MACHINE LEARNING-BASED DEFENCES AGAINST ADVANCED ‘SESSION-REPLAY’ WEB BOTS

SHADI SADEGHPOUR

A DISSERTATION SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE (EECS)
YORK UNIVERSITY
TORONTO, ONTARIO

November 2023

Abstract

The widespread adoption of the Internet has brought about significant benefits for modern society, but has also led to an increase in malicious activities, particularly through the use of web bots. While some bots serve useful purposes, the proliferation of malicious web bots poses a significant threat to Internet security, impacting individuals, businesses, governments, and society as a whole. The emergence of AI-powered web bots capable of mimicking human behavior and evading detection has further exacerbated this problem. This dissertation aims to deepen our understanding of advanced web bots and the web bot attacks that often signal fraudulent online activities. In particular, we focus on session-replay web bots, the latest and most advanced type of web bots, which present an especially difficult challenge in online domains where multiple genuine human users frequently exhibit similar behavioral patterns, such as news, banking, or gaming sites. To achieve our research objectives, we have meticulously curated an extensive dataset encompassing both human and bot-generated data. Additionally, we have developed our own prototype of advanced session-replay bot (the so-called ReBot), which has enabled us to accurately simulate the attacks conducted by this particular category of web bots. Moreover, by infusing randomness into the design of ReBot, we have been able to achieve varying degrees of bot and attack evasiveness. From the defenders perspective, and by leveraging state-of-the-art deep learning algorithms, we have proposed several effective strategies for detection of advanced session-replay bot attacks. One of our proposed techniques deploys the concept of moving-target defence in the form of webpage randomization which is particularly challenging for the attacker to overcome. This thesis also explores the utilization of generative machine learning models for the purpose of generating synthetic bots sessions. The ability to synthesize advance session-replay bots - as opposed to looking for real-world instances of these bots or evidence of their activity in real-world logs - is of critical importance if we are to make timely and effective advances in the field of web bot detection and defence.

Dedication

To my daughter, Niki and my son Kian,

This thesis represents a significant milestone in my life, and I am honored to dedicate it to you both. Throughout this journey, you have been my constant source of motivation and inspiration. Your unwavering support, encouragement, and love have given me the strength to persevere and overcome the challenges that come with pursuing a Ph.D.

As I reflect on the long hours of research, writing, and editing, I am filled with gratitude for the joy and fulfillment that being your parent brings to my life. Your presence reminds me of the importance of balance and the value of pursuing one's passions while never losing sight of what truly matters.

May this thesis be a testament to the depth of my love and appreciation for you both. My hope is that it serves as a reminder of the importance of hard work, determination, and never giving up on your dreams.

To my husband, Mike,

I am deeply grateful for your unwavering love, encouragement, and support during my Ph.D. pursuit. Your presence in my life has been a constant source of comfort and inspiration, and I could not have completed this journey without you. This thesis is dedicated to you as a symbol of my appreciation for all the sacrifices you have made.

With all my heart,

- Shadi, November 2023

Acknowledgements

Throughout the writing of this dissertation, I have been fortunate enough to receive an immense amount of support and assistance, and for that, I am incredibly grateful.

First and foremost, I would like to extend my heartfelt appreciation to my supervisor, Professor Natalija Vlajic, for her unwavering dedication to my success. Professor Vlajic' s invaluable guidance, attention to detail, and insightful feedback have been instrumental in shaping my research and helping me achieve my goals. Her tireless commitment to excellence has inspired me to push beyond my limits and strive for the highest standards in my work. I am truly grateful for her willingness to go above and beyond to provide mentorship and support, which has made a lasting impact on me. I will always be thankful for her belief in my potential and her significant contributions to my growth and development as a researcher. Working under her supervision has been a privilege, and I look forward to continuing to apply the skills and knowledge gained from this experience in my future endeavors.

I would also like to acknowledge members of my supervisory committee and members of the Department of Electrical Engineering and Computer Science (EECS) at York University for their continual support and assistance that I have received during my Ph.D. study. I am especially grateful to Professor Matthew Kyan, and Professor Aijun An for agreeing to serve on my committee members and for providing very useful feedback on my thesis.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ML Machine Learning

AI Artificial Intelligence

AOL America Online

DRL Deep Reinforcement Learning

OSI Open Systems Interconnect

SEO Search Engine Optimization

DDoS Distributed Denial-of-service Attack

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart

cURL Client for URLs

UA User Agent

ATO Account Takeover

NAT Network Address Translation

C&C Command-and-control Server

BHO Browser Helper Object

P2P Peer-to-peer

aIB Agglomerative Information Bottleneck

NN Neural Network

SOM Self-Organizing Map

Modified ART2 Modified Adaptive Resonance Theory 2

MIME Multipurpose Internet Mail Extensions

PCI Principal Component Initialization

BMU Best Matching Unit

GANs Generative Adversarial Networks

SVM Support Vector Machine

MLP Multi-Layer Perceptron

SPRT Sequential Probability Ratio Test

MM Mouse-Move

DD Drag and Drop

PC Point and Click

HLISA Human-Like Interaction Selenium API

GAN Generative Adversarial Network

DTW Dynamic Time Warping

VGG16 Visual Geometry Group (CNN Model)

PCA Principal Component Analysis

t-SNE t-distributed Stochastic Neighbor Embedding

SSOM Spherical SOM

ReBot Replay Bot

ReBotDetector Session-replay Bot Detector

LSTM Long Short-term Memory Networks

RanReBot Randomized Session-replay Web Bot

RanReBotDetector Randomized Session-replay Bot Detector

RanABD Randomization for Advanced Web Bot Detection

MTD Moving Target Defense

Chapter 1

Introduction

1.1 Motivations and Contributions

The proliferation of web bots has ushered in a multitude of malicious activities, posing a significant menace to online security. These activities encompass a wide range of nefarious actions, including web scraping, data mining, the extraction of personal and financial data, brute-force login attacks, digital ad fraud, denial of service attacks, spam, and transaction fraud, among others. Such attacks have raised profound concerns across various industries due to the web bots' capacity to emulate human behavior and operate stealthily, executing their malicious agendas undetected.

This doctoral thesis embarks on an exhaustive exploration of the intricate landscape surrounding the identification of advanced web bot attacks, underscoring the vital role that academic research plays in comprehending and addressing these challenges. A profound understanding of the complex nature of these attacks is essential for formulating robust defense strategies. The thesis commences by delving into the far-reaching repercussions of the escalating bot traffic, the surge in malicious bot activities, and the ever-growing risks posed by automated malevolence. The discussion also offers an insightful overview of successive generations of malicious web bots, illustrating their varying levels of sophistication and the diverse hazards they pose across various business sectors.

In the quest to address these challenges comprehensively, the remainder of this dissertation makes significant contributions from two main perspectives: the defender's and that of the attacker, as illustrated in Figure 1.

From the defender's perspective, in response to the detrimental impact of web bots, our primary focus revolves around the development of effective web bot detection systems to thwart their malicious activities. We initially introduce an unsupervised machine learning-based detection model, which incorporates automated feature selection and validation using real-world server-log data (Figure 1 Point 1). Building upon this success, we delve into the integration of behavioral biometrics techniques, particularly mouse dynamics, to augment the bot detection process. Our

research sheds light on the imperative need for advanced web bot detection techniques to combat the latest and most sophisticated category of web bots, commonly known as session-replay bots. These bots present a formidable challenge, especially on online platforms which are repeatedly visited by the same user(s) and where multiple genuine human users exhibit similar behavior patterns, as commonly observed in news, banking, or gaming sites. To tackle this challenge, we adopt the attacker's perspective and develop an in-house session replay bot (ReBot) in order to be able to simulate attacks conducted by this specific type of web bots (Figure 1 Point 2). We also meticulously curate a comprehensive dataset comprising repeat sessions of a diverse group of genuine human users as well as sessions generated by ReBot. Subsequently, harnessing the power of deep learning algorithms, we formulate a detection model capable of identifying session-replay bot sessions within our dataset (Figure 1 Point 3). Next, by taking the perspective of an advanced adversary, we explore an innovative approach involving "randomization" to intelligently modify the replayed session trajectories generated by our ReBot, making them appear more authentic (i.e., appearing like genuine human-user sessions) and thus harder to detect (Figure 1 Point 4). Leveraging the state-of-the-art deep learning algorithms, we elaborate on effective strategies to counteract these evolving threats (Figure 1 Point 5). This thesis also delves into the utilization of generative models to create synthetic advanced session-replay bots (Figure 1 Point 6).

As the last major contribution, we propose a novel bot detection technique designed to counter malicious AI-powered web bots by employing the concept of webpage randomization (Figure 1 Point 7). This approach makes the task of replicating genuine human user trajectories/sessions far more complex for the adversary, effectively serving as a variant of Moving Target Defense (MTD) strategies. By increasing the level of complexity and uncertainty in the adversary's path, we significantly enhance the probability of successful bot detection and defense.

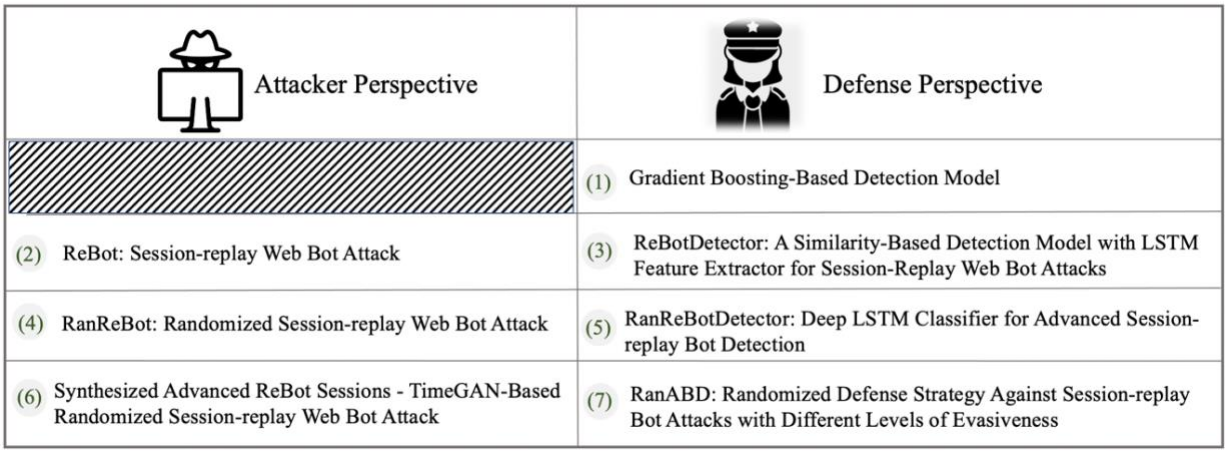


Figure 1. Contributions from attackers and defenders perspectives.

1.2 Two Main Bot-Detection Methodologies Deployed in This Thesis Work

In this thesis work, we address the challenges of web bot detection through two distinct methodologies:

The first methodology exclusively utilizes the access logs collected on the target/victim server. These logs contain information pertaining to the specific pages and resources requested by each user while visiting the given server (i.e., webpages hosted by this server), as well as the timing of these requests. This methodology is thoroughly explored and implemented in Chapter 3. The second methodology centers on leveraging client-side collected behavioral data, such as mouse movements and actions executed by each particular user while visiting one of the target-server's webpages. This approach becomes the primary focus in the subsequent chapters of the thesis – Chapters 4 through 10.

Now, each of these methodologies have their own advantages and disadvantages [1, 2], as outlined below:

Bot detection deploying server-side logs - advantages:

- Availability: Server-side logs are readily available and commonly recorded by web servers, making data collection convenient.
- Scalability: Server-side logs can handle large amounts of data from numerous users, making it suitable for high-traffic websites.

Bot detection deploying server-side logs - disadvantages:

- Limited behavioral information: Server-side logs provide information on page/resource requests but lack detailed behavioral data, making it challenging to discern one human user from another, or a human user from a bot.
- Privacy concerns: Server-side logs may contain sensitive user information, raising privacy concerns, and necessitating careful handling and anonymization.

Bot detection deploying client-side mouse dynamics data - advantages:

- Rich behavioral insights: Collecting mouse dynamics data offers detailed behavioral information, such as mouse movements, clicks, and interactions, providing a more comprehensive view of user behavior.
- Better bot discrimination: Client-side data can help differentiate human users from bots more effectively, as certain mouse movements and interactions are unique to humans.
- Leveraging the complexity of human behavior: To evade a well-designed mouse dynamics-based detection model, adversaries must grapple with the complex task of accurately modeling human behavior. Behavioral biometrics research has demonstrated that modeling human behavior is notoriously challenging [3].

Bot detection deploying client-side mouse dynamics data - disadvantages:

- User consent: Collecting client-side data requires user consent due to privacy considerations pertaining to collection of biometrics-related information (e.g., the way a user moves their mouse cursor on a screen), and some users may be hesitant to grant that permission.
- Data volume and processing: Client-side data gathering can generate a considerable amount of information, which may be challenging to handle and process, particularly for high-traffic websites.

In summary, our choice to pursue the use of mouse dynamics data is motivated by its potential to provide continuous in-depth behavioral insights and be deployed as a non-obtrusive strategy for differentiating bots from human users.

1.3 Dissertation Organization

The remainder of this thesis is structured as follows:

Chapter 2 conducts an extensive exploration of web bots, shedding light on their growing prevalence and ability to mimic human actions. It traces their historical evolution, from basic scripts to advanced algorithms, and scrutinizes the various generations of malicious web bots. The chapter thoroughly examines the wide-ranging impacts of these bots on business operations and industries, underscoring potential risks to customer interactions, data integrity, and marketing initiatives. Additionally, the chapter presents a comprehensive assessment of diverse web bot detection methods, encapsulating key insights and emphasizing the imperative for adopting innovative detection approaches and advanced machine learning techniques.

Chapter 3 introduces an unsupervised web bot detection method that utilizes server logs and integrates the use of Gradient Boosting Technique and Self-Organizing Map (SOM) algorithm. The performance of this integrated system is validated using a real-world dataset obtained during a confirmed large-scale attack on York University, our home institution.

Chapter 4 discusses the potential of utilizing behavioral biometrics techniques, specifically mouse dynamics, to enhance bot detection. It provides a survey of the existing literature on mouse dynamics and bot detection research, specifically covering: data acquisition mechanisms, feature representations, classification methods, experimental protocols, and performance evaluations The chapter also suggests possible directions for future research.

Chapter 5 introduces the ReMouse dataset and presents the results of our analysis of this novel dataset using statistical and advanced machine learning (ML) techniques, including deep and unsupervised neural learning.

Chapter 6 presents ReBot, our prototype of session-replay bot which has been developed to allows us to simulate a wide range of session-replay bot attacks.

Chapter 7 introduces ReBotDetector, our session-replay bot detection system specifically designed to identify/detect ReBot sessions.

Chapter 8 introduces RanReBot, our innovative state-of-the-art solution/enhancement designed to improve the realism of ReBot sessions while maintaining their fundamental structure and intent.

Chapter 9 proposes a detection model named RanBotDetector, our further improved bot detection model based on deep LSTM time-series, for classifying sessions generated by the advanced session-replay bot, RanReBot.

Chapter 10 outlines our novel ‘moving target detection’ approach to design of webpages that can result in more effective detection of session-replay bot attacks.

Finally, Chapter 11 concludes the study, summarizing the contributions made to the field of bot detection, and outlines potential directions for future research.

Figure 2 offers a concise summary of all thesis chapters, offering the reader a clear roadmap of the covered topics and what to anticipate in each chapter.

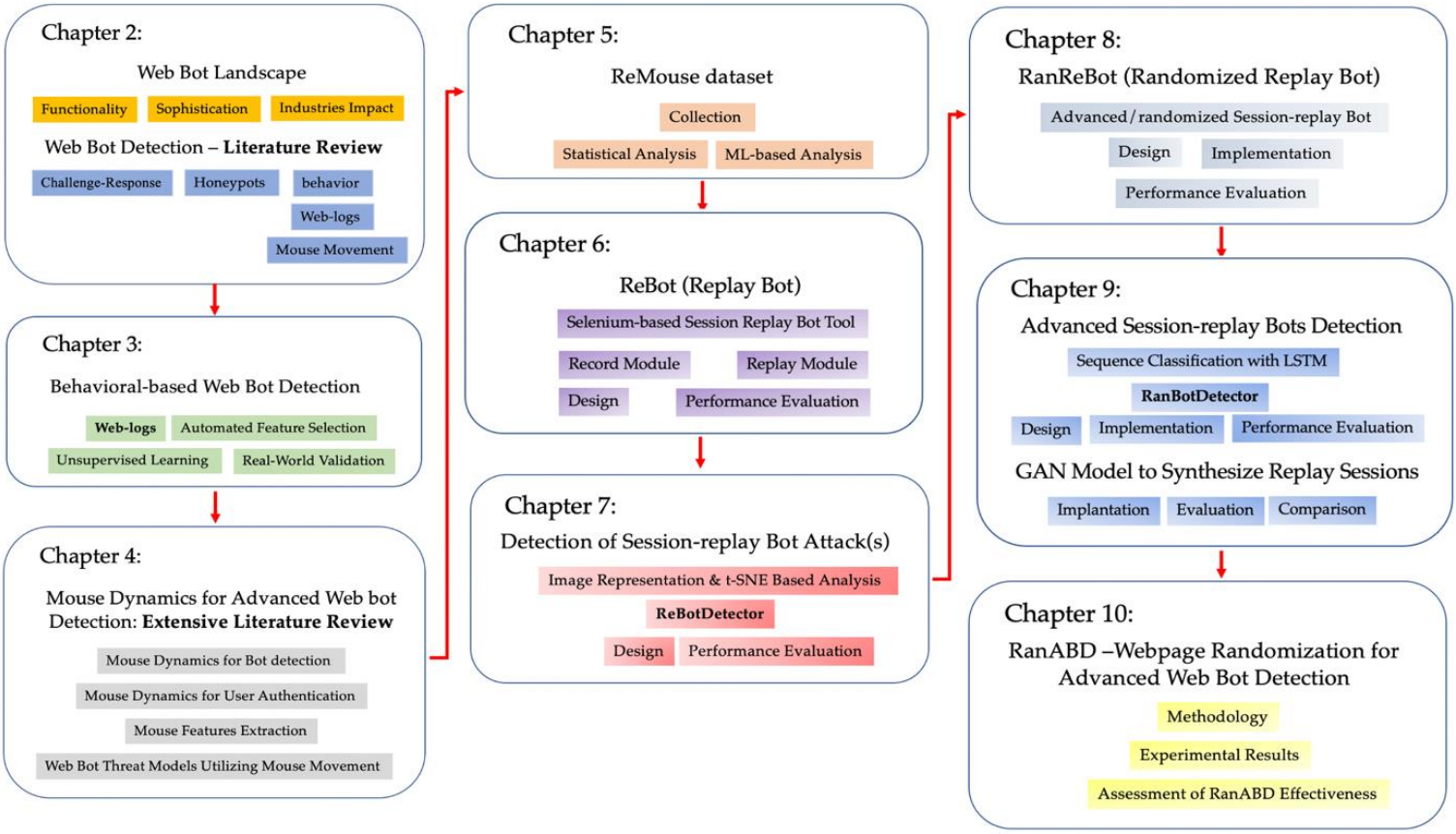


Figure 2. Thesis outline - summary of chapters and contents.

1.4 Peered-Reviewed Scientific Publications

The following are the research outcomes presented in this thesis, encompassing the various publications resulting from our study.

- Sadeghpour, Shadi, and Natalija Vlajic. “RanABD: MTD-Based Technique for Detection of Advanced Session-Replay Web Bots” is accepted for publication at MTD’ 23, 10th ACM Workshop on Moving Target Defense (MTD).

- Sadeghpour, Shadi, and Natalija Vlajic. “RanABD: Webpage Randomization for Advanced Web-Bot Detection”. 2023 IEEE CSNet 2023, 7th Cyber Security in Networking Conference.

- Sadeghpour, Shadi, and Natalija Vlajic. “Analysis of Novel Mouse Dynamics with Repeat Sessions: Helpful Observations for Tackling Session-Replay Bots”. 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC) (pp. 790-797). IEEE.

- Sadeghpour, Shadi, and Natalija Vlajic. “ReMouse Dataset: On the Efficacy of Measuring the Similarity of Human-Generated Trajectories for the Detection of Session-Replay Bots.” Journal of Cybersecurity and Privacy 2023, 3 (1), 95–117.

- Sadeghpour, Shadi, and Natalija Vlajic. "Poster: ReMouse Dataset: Measuring Similarity of Human-Generated Trajectories as an Important Step in Dealing with Session-Replay Bots." Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. 2022.

- Sadeghpour, Shadi, and Natalija Vlajic. "Ads and Fraud: A Comprehensive Survey of Fraud in Online Advertising." Journal of Cybersecurity and Privacy 1.4 (2021): 804-832.

- Sadeghpour, Shadi, and Natalija Vlajic. "Click Fraud in Digital Advertising: A Comprehensive Survey." Computers 10.12 (2021): 164.

- Sadeghpour, Shadi, et al. "Unsupervised ML-Based Detection of Malicious Web Sessions with Automated Feature Selection: Design and Real-World Validation." 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC). IEEE, 2021.

- Madani, Pooria, Natalija Vlajic, and Shadi Sadeghpour. "Mac-layer spoofing detection and prevention in IoT systems: randomized moving target approach." Proceedings of the 2020 Joint Workshop on CPS & IoT Security and Privacy. 2020.

Chapter 2

Evolution and Classification of Web Bots: Understanding the History and Current Landscape

This chapter provides an in-depth exploration of the surging prevalence of web bots, which near if not surpass the numbers of human visitors on present-day websites, and their increasing ability to imitate human behavior, thereby posing significant challenges for detection. It traces the historical evolution of web bots, from basic scripts to sophisticated algorithms, and investigates the different generations of malicious web bots. The chapter sheds light on the significant impact of these bots on business functions and industries, highlighting the threats they pose to customer interactions, data security, and marketing efforts. A comprehensive review of diverse web bot detection techniques is also presented in this chapter, summarizing essential findings and emphasizing the critical need for novel detection strategies and advanced machine learning techniques to effectively counter these evolving threats.

2.1 Introduction

The 2023 Bad Bot report released by Imperva [4] offers invaluable insights into the realm of bots and their profound influence on online activities and generated traffic. This exhaustive analysis meticulously scrutinized bot behavior over the past number of years, revealing captivating trends and disconcerting advancements. Noteworthy among the findings is the revelation that bots were accountable for nearly half (47.4%) of all Internet traffic, marking a substantial 5.1% upsurge from the preceding year. In stark contrast, human Internet traffic witnessed a downturn, plunging to its lowest point in an eight-year span.

Of all the diverse Internet bots, the most disconcerting are the infamous "bad bots" – software applications capable of perpetrating high-speed abuse, misuse, and malicious attacks. The report

unveils alarming findings, indicating that bad bot activity accounted for a substantial 30.2% share of Internet traffic, marking the highest level of such malicious activity recorded in the last decade.

Crucially, the rapid evolution of bots is fueled by the advent of generative artificial intelligence, which is poised to further accelerate this process over the next decade. As a result, the increasing proportion of bot traffic poses tangible risks to businesses, with a potential to cause detrimental impacts on brand reputation, diminished online sales, and heightened security vulnerabilities for web applications, mobile apps, and APIs.

Supporting this concern, the findings from both [4] and [5] underscore a disconcerting reality: a staggering 35% of all attacks witnessed in 2022 were meticulously aimed at application programming interfaces (APIs). Notably, 17% of these targeted assaults originated from bots that harnessed the power of "business logic[1]." These malicious bots adeptly exploit vulnerabilities within APIs, enabling them to surreptitiously pilfer sensitive data and gain unwarranted access to vital accounts. Evidently, this emerging trend poses a burgeoning and pressing concern for organizations across the board.

As cybercriminals intensify their focus on attacking API endpoints and application business logic through sophisticated automation, the predictions [4] estimate a significant escalation in business disruptions and financial impacts associated with bad bots in the coming years.

This chapter aims to explore the far-reaching implications of the burgeoning bot traffic, the surge in bad bot activity, and the escalating risks posed by malicious automation. The chapter is organized as follows: Section 2.2 delves into the history and characteristics of web bots, providing insights into their evolution and functionalities. In Section 2.3, an overview of different generations of malicious web bots is presented, highlighting the varying levels of sophistication and threats posed by these bots. Section 2.4 focuses on the impact of malicious bots on various business functions and industries, exploring the detrimental effects on customer interactions, data security, and marketing efforts. Section 2.5 presents a comprehensive review of various web bot detection techniques, highlighting key findings, and underscoring the urgent need for novel detection strategies and advanced machine learning techniques to effectively combat these evolving threats. Finally, Section 2.6 concludes the discussion, summarizing the key findings and emphasizing the

[1] A business logic attack is a type of assault directed at the flaws present in an application's design and implementation. These weaknesses can be manipulated by attackers to exploit legitimate functions and achieve nefarious objectives, like unauthorized access to user accounts and theft of sensitive data. In contemporary software development, APIs play a crucial role, yet their vulnerability to bad bots exploiting business logic vulnerabilities is significant if not properly secured [4].

need for novel detection strategies and machine learning techniques to effectively counter the growing dangers arising from the use of web bots by a wide range of malicious actors.

2.2 History and Characteristics of Web Bots

The history of web bots can be traced back to 1988, and to the advent of Internet Relay Chat (IRC) bots such as those used within the Hunt the Wumpus game platform or Bill Wisner's Bartender bot [6]. These early IRC bots provided automated services to users and sat in channels to prevent servers from shutting down due to user inactivity. It was not until 1994 that the first web crawlers (i.e., web bots) were created. The first such bot (used to index webpages) was created by AOL (America Online) in 1995 and purchased by Excite in 1997. Soon after, several commercial web crawlers became available such as Lycos, Infoseek, Excite, AltaVista, and HotBot [7].

While the early web bots were generally used for benign purposes, over time various types of malicious web bots started emerging to ultimately become the most dominant and active bot category of the present-day Internet [8].

There are several different definitions of what can be classified as an Internet/web bot.[2] For example, Radware states that "bots are automated programs created to execute repetitive tasks", Wikipedia says "bot is a software application that runs automated tasks (scripts) over the Internet", and Netaces describes automated traffic as "any set of legitimate requests made to a website that is made by an automated process rather than triggered by a direct human action".

According to [9], web bots are generally grouped into the following main categories and respective sub-categories:

1. **Good Web Bots** are legitimate bots whose activities might be beneficial to businesses as well as individuals. Some specific sub-categories of good web bots and the functions they perform are listed below:

    • Website monitoring bots monitor websites' availability and system health. An example of a bot in this category is Pingdom [10].

[2] It should be stressed that the term "Internet bots" refers to a broad family of malicious programs that target layer 3 (network layer), layer 4 (transport layer), or layer 7 (application layer) of the Open Systems Interconnect (OSI) model. However, the application layer bots are the only bots explicitly capable of mimicking human behavior, and as such are the main focus of this research. For that reason, in the remainder of this chapter, we use the terms 'application-layer bots' and 'web bots' interchangeably.

• Aggregator bots collect information from websites and notify users or subscribers about news or events. An example of this type of bot is Feedly [11].

• Backlink checker bots confirm the inbound (referrer) URLs that a website receives so that marketers can understand trends and optimize their pages accordingly. SEMRushBot is an example of this type of bot [12].

• Partner bots execute tasks and functions on transactional websites. An example being PayPal IPN [13].

• Social networking bots are deployed by social networking platforms to add visibility to their webpages and drive overall user engagement. Facebook bots are an example of this type of bot.

• Search engine bots, which are also known as web crawlers or spiders, crawl through websites in order to index their pages and make them available/accessible on the respective search engine. Without them, most online businesses would struggle to define their brand value and attract new customers. Bots in this category include: GoogleBot, Bingbot, and Baidu Spider.

2. **Bad Web bots** are programmed to perform various malicious tasks on the WWW. They work evasively and are mainly used by scammers, cybercriminals, and other nefarious parties involved in a variety of illegal activities. Bad bots are automated programs that do not follow (i.e., respect) any rules. Mostly unregulated, they have a specific malicious objective which they are trying to accomplish. According to [9] and [14], some general sub-categories of bad web bots are:

• Scraper bots collect/steal large amounts of information from websites. They are scripted to look for specific data, including product reviews, breaking news, prices, customer names, product catalogues, or even user-generated content on community forums. By scraping the content off a website and then posting it somewhere else, bots can negatively affect the search engine's ranking of this websites and/or the products it advertises. By scraping and posting content elsewhere, bots can also have a negative impact on the companies that invest budget and resources into creating original digital content.

• Scalper bots are designed to automatically capture and purchase goods and have a high-speed checkout process. They make bulk purchases. For example, they buy hundreds of tickets immediately after opening of a booking and then sell them through reseller

websites for a price considerably higher than the initial ticket price. It is very common for scalper bots to mimic human behavior in order to avoid detection.

• Spam bots (also known as content spammers) inject messages into the user-controlled areas of a website, such as forums, guestbooks, bulletin boards, and reviews or comments sections associated with news articles. They arrive in the middle of users' conversation and insert messages with unwanted advertisements, links, and banners. Such insertions often frustrate real users who participate in forums and comment on blog posts, and potentially drive them away from the given forum or bulleting board. Moreover, spam bots may insert malicious links to direct users to phishing sites in order to trick them into revealing sensitive information such as bank account numbers and passwords.

• Session-replay bots are automated software programs designed to mimic and replicate the actions of human users as they interact with web applications, websites, or online services [1]. These bots record and replay user sessions, including mouse movements, clicks, and keystrokes, in an attempt to collect data, gather insights, or engage in potentially malicious activities such as credential theft, fraud, or unauthorized data harvesting.

• Click bots are purposefully designed for engaging in click fraud, a deceptive practice involving the generation of artificial clicks on webpages [15]. While basic click bots merely access a webpage and click a link, sophisticated ones mimic real user behavior with mouse movements, random pauses, and varying timings between clicks. Click bots aim to appear legitimate to evade detection. According to ClickCease [2], a prominent click fraud prevention company, these bots contribute significantly to fraudulent clicks, potentially up to 50% of all fraudulent clicks on online ads. As a countermeasure, click fraud campaigns often utilize botnets - networks of devices, each running a copy of the click bot, with different IP addresses to make the clicks seem like they come from individual users. The increasing sophistication of click bots poses a serious threat to online advertisers and publishers, making their detection and prevention challenging.

It is worth noting that both good and bad bots can regularly make requests from real browsers and execute JavaScript code intended to validate users as humans. Sophisticated web bots can also bypass modern detection mechanisms such as CAPTCHA (Completely Automated Public Turing

test to tell Computers and Humans Apart)[3] either by using artificial intelligence, bruce-force systems, or with the help of human agent farms.

One of the most significant threats posed by web bots in today's digital landscape is the phenomenon known as a "botnet attack". A botnet is a large network of bots (i.e., malicious programs), each running on a compromised computer while being controlled by a remote command-and-control center (the botnet operator). Commonly, the word botnet creates an image of a Distributed Denial-of-service (DDoS) attack. However, in reality, botnets can carry different payloads and can be used in different types of attacks. For example, they can be used to extract cryptocurrency from infected devices, or to cover up other attacks or an illegal activity. Some bots (i.e., botnets) can be utilized as email relays for massive spam campaigns. Ultimately, the threats emerging from botnets are only limited by the creativity of their creators [16].

2.3 An Overview of Different Generations of Malicious Web Bots

Web bots have evolved rapidly from their origins as simple scripting tools with command-line interfaces to modern-day, complex programs that leverage full-fledged browsers and are able to mimic human behavior (e.g., navigate a website or application, move the mouse, touch and drag objects, etc.).

In this section, we provide a more detailed overview of four generations of malicious web bots (as commonly classified by the cybersecurity community), all of which can still be found in use today [9].

- First-Generation Bots are basic scripts that send requests such as cURL [17] from a small number of IP addresses. (Client for URLs, or cURL, [17] is a command-line tool for getting or sending files using URL syntax.) These bots cannot store cookies or run JavaScript code (i.e., they do not have real web browser functionality), and can be easily detected and mitigated by blacklisting their IP addresses and UAs, as well as combinations of IPs and UAs. They are mostly used for scraping, carding, and spamming.
- Second-Generation Bots leverage headless browsers (such as PhantomJS [18]), and unlike first-generation bots can store cookies and execute JavaScript code to automate control of a

[3] https://en.wikipedia.org/wiki/CAPTCHA

website. These bots are used to conduct DDoS attacks, scraping and spamming campaigns, as well as to skew web analytics or conduct ad fraud. However, they can be effectively detected using their browser and device characteristics including the presence of certain JavaScript variables, frame forgery, sessions, and cookies. Once identified, these bots can subsequently be blocked based on their fingerprints. Another way of detecting these bots is by analyzing their click-path through the target website as they often exhibit significant discrepancies relative to the click-path of ordinary (human) users/visitors.

• Third-Generation Bots can operate in full browser mode and are capable of executing human-like interactions such as simple mouse movements and keystrokes. However, they are typically unable to exhibit subtle and unique randomness that is characteristic of human online behavior[4]. They are commonly used to execute DDoS attacks, API abuse, ad fraud, and account takeover fraud. An interaction-based user behavioral analysis approach could help in identifying these bots as they generally follow a programmatic sequence of URL traversals.

• Fourth-Generation Bots, the most advanced category of bad bots, are capable of mimicking more complex mouse movements of a human and engaging in humanlike click-path patterns. Namely, the developers of this category of bots engage in behavioral hijacking (i.e., recording real users touching and swiping behaviors on hijacked mobile apps or websites) to fully simulate human behavior on websites or apps. These bots can also change their UA (user agent) and cycle through thousands of IP addresses. All of this makes the process of detecting the fourth-generation bots extremely challenging. They are typically employed in ad fraud, account takeover, API abuse, and DDoS attacks. Figure 3 shows the key behaviors of bad bots by generation.

In our study, we consider our adversaries as web bots specifically designed to interact with websites and online platforms, exhibiting humanlike behavior. These malicious web bots closely mimic human behaviors on the majority of everyday websites (e.g., news agencies, banking, social media, etc.). According to our definition, these advanced web bots are maliciously programmed to

[4] These limitations are primarily attributed to programming constraints, as the level of randomness generated by the bot is restricted by the code and algorithms devised by its developer. In contrast, human behavior is influenced by a myriad of factors, rendering it intricate and challenging to be fully replicated through programming alone. Moreover, the predictable patterns exhibited by these bots arise from their adherence to pre-defined scripts or instructions, while human online behavior is shaped by emotions, intentions, and external stimuli, resulting in a diverse array of unique and unpredictable actions.

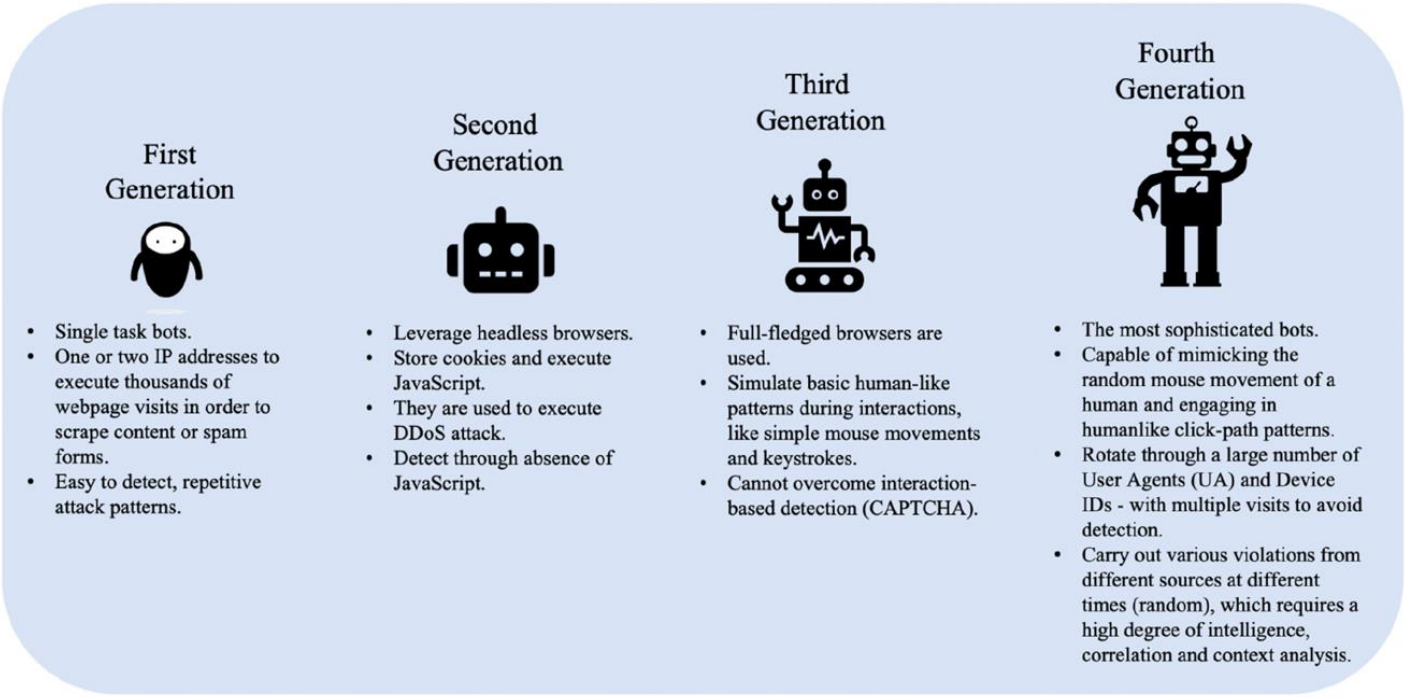


Figure 3. The evolution of bots.

ensure their activities are indistinguishable from those of genuine users, enabling them to navigate online environments seamlessly.

2.4 Impact of Malicious Web Bots on Various Business Functions and Industries

There are a wide variety of activities involving automated traffic that can be used to exploit businesses across all industries. NETACEA [14] believes that, regardless of the industry targeted with these attacks, the core of each bot attack is due to one of the following three motives: money, data, and stock.

2.4.1 Impact of Malicious Web Bots on Different Business Functions

We categorized the main types of web bot attack techniques that are used to exploit business logic under seven broad categories [7, 9, 14][5]:

[5] The references cited in this section predominantly originate from industry studies and white papers due to several key reasons. First and foremost, industries possess direct access to real-world data, including logs, network traffic, and

• Web scraping (Scraping of pricing, content and inventory information): This is a technique of extracting different types of information from websites, such as product prices and news content, which can be costly if extracted without consent. For example, nefarious competitors scrape prices and product lists to attract the other business' customers. They effortlessly steal whatever pieces of content they are programmed to find in order to sabotage the (victim) retailer 's sources of income. Attackers also scrape unique content (and duplicate exclusive content) of an online business to negatively impact their search engine optimization (SEO) efforts.

• Cart Abandonment and Inventory Exhaustion: Merchants usually leave items in the shopping cart for about 10 to 15 min before concluding that the buyer has abandoned the purchase. After this period, the items are released and placed back into the available inventory. Competitors' bots put hundreds of items in shopping carts and abandon them later to limit real consumers from buying products. That sets the grounds for a decline in sales, distorted conversion rates, and ultimately a damaged brand reputation.

• Application DDoS: These types of attacks look for functionality areas that are 'weak points' of the target application. This can be an area that involves high CPU usage, integration with third-party systems, or complex database activity such as search, registration, availability checking, or real-time booking requests. Malicious web bots automate their requests to those areas of the website until the website reaches its limit and fails or is unable to carry out normal transactions with legitimate customers. These attacks specialize in utilizing rotating IP addresses and legitimate user agents (to conceal the bots' identities) and are usually launched via large botnets.

• Scalping Products and Tickets: Malicious bots can be programmed to actively buy valuables goods such as consumer electronics and resell them for a considerably higher price. Bots can pick up tickets for popular events as soon as they go on sale.

---

incident reports. This access equips them with invaluable insights into the latest tactics employed by malicious web bots, allowing for a more immediate and informed response to evolving cybersecurity threats. The practical experience of cybersecurity professionals within the industry, dealing with web bot attacks on a daily basis, significantly contributes to a nuanced and realistic understanding of the threat landscape. In contrast, academic researchers, while possessing deep theoretical knowledge, may lack day-to-day exposure to the practical challenges faced by industry professionals. Furthermore, accessing real-world data for researchers in academia proves to be a challenging endeavor. Essentially, the amalgamation of industry white papers' access to real-world data, close ties to cybersecurity incidents, focus on practical solutions, timely reporting, and practical experience results in a more comprehensive and up-to-date comprehension of web bot attacks. This practical insight complements the theoretical contributions of academic research, creating a well-rounded and informed perspective on the intricate nature of web bot activities in different industry contexts.

• Card Cracking: Fraudsters use bots to test thousands of stolen credit card numbers against merchant payment processing. Since the stolen card owner can report a fraudulent transaction and request a repayment, the sites targeted with card cracking attacks will ultimately suffer financial losses (due to issued refunds), legal penalties, and lousy trading history. In extreme cases, frequent carding activities and too many refunds may force the merchant to disable credit card payments altogether.

• Fake Account Creation: Criminals use bots to create fake accounts and commit various forms of cybercrime. Some of the activities that can be carried out after creating such accounts include: misusing the 'first-time-buyer' bonus, using a free product trial awarded to a new account, using multiple accounts to attack the inventory of websites that only allow logged-in users to store items, content spamming, money laundering, malware distribution, and skewed research and SEO.

• Account takeover (ATO): Account takeover bots focus on gaining control over user accounts within a system and accessing people's personal data for use elsewhere. Credential stuffing and card cracking/credential cracking are amongst the commonly used ATO techniques, and each uses automated bots to gain brute force entry to an account. In the credential cracking attack model, multiple username and password combinations are attempted until a successful combination is discovered. Credential stuffing as an alternative approach involves taking known lists of email and password combinations and determining if they are further valid for alternative sites. After the credentials are authenticated, attackers can extract money or other financially valuable items (e.g., loyalty rewards) from within that account. They can also harvest personal data for use/sale elsewhere.

2.4.2 Impact of Malicious Web Bots on Different Industries

The presence of bad bots is a pervasive issue affecting all industries. These malicious bots possess the capability to execute various harmful actions at an unprecedented speed, surpassing what a human could achieve. As a result, they have become an ideal tool for high-speed abuse, misuse, and attacks. While certain bad bot use cases, such as content scraping and account takeover, are observed across multiple industries, there are also industry-specific use cases. For instance, scalping, which significantly impacts online retailers and entertainment ticketing services, is a type of bad bot activity that is particularly prevalent in these sectors.

The top five industries that faced the largest share of bad bot traffic in 2022, along with their respective percentages, are as follows [4]: Gaming (58.7%), Telecom and ISPs (47.7%), Community and Society (41.1%), IT and Services (40.0%), and Digital Advertising (38.0%).

This section reviews some of the negative impacts that bad bots have on certain industries.

- Threat in Gambling and Gaming: Account take over and credential stuffing are the two most common techniques that gambling, and gaming companies suffer from because each account contains cash or loyalty points that can easily be transferred to other users and emptied if compromised.
- Threat in Telecommunication and ISPs: The Telecom and ISPs sector experienced a slight increase in bad bot traffic, rising from 46.9% in 2021 to 47.7% in 2022 [4]. This industry encompasses mobile ISPs, residential ISPs, hosting providers, and others. Bad bots engage in various malicious activities within this sector, including scraping sensitive customer data and carrying out brute force login attacks to take over user accounts. Due to the sector's heavy reliance on continuous availability and its sensitivity to downtime, bad bots target it by inundating its infrastructure with a massive number of requests, disguising themselves as legitimate users. This overwhelms the system and disrupts services, leading to potential financial losses and customer dissatisfaction. Moreover, bot traffic can distort website analytics, resulting in misguided decision-making based on inaccurate data.
- Threat in Community and Society Web Domains: Bad bots accounted for 41.4% of the traffic on community and society websites [4]. Among the most prevalent issues in this industry are spam bots, also referred to as Fake News Spam and Comment Spam [19]. These malicious bots are responsible for spreading fake news, propagating propaganda, and concealing malware within clickbait links. Additionally, the community and society sector includes numerous non-profit organizations that accept donations on their websites. Bots exploit these donation pages to test stolen credit card numbers, posing significant financial challenges and burdens that many non-profits cannot afford.
- Threat in IT and Services: Malicious bot attacks pose significant risks to the IT and Services industry, capable of causing severe disruptions in business operations. These attacks can freeze inventory, crash customer service systems, suspend orders, and cripple IT infrastructure, potentially leading to revenue loss and even business closures. In 2022, this industry experienced 40% of its traffic originating from bad bots, highlighting the

severity of the issue [4]. The negative impact of bad bots in this sector spans from technical problems to fraud and security threats.

- Threat in Digital Advertising: Digital advertising is known as a multi-billion-dollar industry that uses very sophisticated methods to ensure that the maximum value is extracted [7]. Fraudsters use botnets to generate fake clicks and obtain fraudulent digital ad impressions. Fake traffic artificially increases advertising costs. Malicious automated traffic also performs retargeting fraud to illegally generate revenue from invalid traffic to publishing sites. Such attacks sabotage the advertising network's efforts to connect them to quality inventory. It also prevents marketers from reaching a wider audience. Bad bots generate invalid traffic, which negatively affects the brand reputation of an advertising network and undermines its claim to provide reliable media for a media buying environment. Over and above that, skewing of analytics and other metrics by bad bots would result in invalid business decisions and a large amount of marketing and advertising expenditure being squandered, often in a matter of hours [20].

Below are several other industries that are also affected by bad bot traffic:

- Threat in Finance: Banks, financial service providers, and insurance companies are counted as high-value targets for fraudsters. In recent years, botnet attacks have progressively ramped up the rate and extent of fraud in these industries. The types of botnet attacks on financial institutions include: account takeover, DDoS attacks, and content scraping. However, credential stuffing and card cracking are the two most common techniques used by attackers in the financial services domain [9].
- Threat in Education: Malicious bots can be employed to look for research papers, class availability, and access user accounts in educational institutions [21]. The unauthorized scraping of research papers not only violates copyright laws but also undermines the integrity of academic publishing. By exploiting class availability systems, these bots gain unfair advantages, hindering legitimate students' chances of enrolling in high-demand courses. Moreover, attempts to access user accounts raise concerns about data breaches and privacy violations.
- Threat in E-commerce: E-commerce companies receive a wide range of bad bot attacks. For example, malicious bots sent to third parties by competitors can crawl/collect information from these websites to post them elsewhere or even (re)sell them. Furthermore,

malicious bots can not only steal new listings, but they can also fill web forms with bogus details. In general, their activities include price and content scraping, account takeovers, credit card fraud, and gift card abuse [22].

- Threat in Travel: Card cracking in the travel industry results in the theft of valuable and monetizable frequent flyer miles that are subsequently sold for a profit. Bad web bot aggregators plague travel sites for travel lists, prices, and trends that can be used to inform and offer competitive package deals. In the airline sector, bots are employed to reserve seats on a flight for up to 20 min (the time-window allowed until the reservation is paid for) [22]. During this time, genuine customers are shown that there is a reduced or no availability on the given flight. The perpetrators then can try to sell these seats for a profit.
- Threat in Healthcare Industry: In 2022, healthcare websites encountered a substantial threat, with approximately 31.7% of their traffic originating from malicious bots [1]. These malicious bots pose several risks to the healthcare sector, including data breaches, where they take over user accounts to access sensitive medical records, which can then be sold on the dark web or used for fraudulent activities. Bad bots also pose a threat through Distributed Denial of Service (DDoS) attacks, overwhelming healthcare systems and hindering patient access to information and services. Additionally, these bots can spread misinformation and spam, potentially leading to harmful outcomes such as misdiagnosis and mistreatment.

In our research ([23], [24]), we conducted a focused examination of the digital advertising industry to understand the impact of malicious bot traffic on this particular sector. The motivation for our investigation stemmed from the significant global financial losses caused by digital advertising fraud, with marketers wasting billions in online ad spend each year. The estimated financial impact ranges from 35 to 100 billion U.S. dollars between 2018 and 2023, and in the United States alone, it was projected to reach 81 billion U.S. dollars in 2022 [20]. The concern expressed by marketers worldwide regarding ad fraud underscores the need for effective tracking tools and stricter regulations in this domain.

Our two comprehensive studies, [23] and [24], provided in-depth coverage of various facets of online advertising fraud. In particular, we explored digital advertising platforms, revenue models, and classified ad fraud based on the human actors involved. Additionally, we conducted an in-depth investigation of click fraud, a critical issue characterized by the generation of false

clicks and impressions, leading to substantial revenue losses for businesses in the digital advertising industry. Our research provided an extensive overview of the state of click advertising fraud, highlighting the use of click-botnets as the most prevalent means of conducting click-fraud. We also surveyed the most representative click-fraud countermeasures, ultimately pointing to the fact that existing machine learning techniques were inadequate in effectively countering the rapidly evolving and mutating strains of click bots.

As a result of our findings, we have recognized the imperative to address a wide range of bad bots, including those involved in click fraud. Consequently, our research has shifted its focus to the detection of malicious web bots and a thorough investigation into human-like web bots, allowing us to gain a deeper understanding within this field of study.

2.5 An Overview of Malicious Web Bot Detection Techniques

The challenge of web bot detection revolves around accurately distinguishing between human visitors and automated bots. In the past, web bot detection relied on examining the signature of the visitor's request, including request headers, and checking for support of JavaScript, cookies, and web sessions. However, with tools like Selenium[6], present-day bots are very much capable of mimicking these signatures and features, making traditional methods less effective [25].

One widely known technique for web bot detection is the use of challenge-response techniques, such as CAPTCHAs. CAPTCHAs are Turing tests that present visual challenges or auditory challenges for the visually impaired users [26]. The assumption is that humans can extract information from distorted images or audio, while bots cannot. However, attackers have developed techniques to bypass some popular CAPTCHA challenges, including the use of public speech-to-text engines to bypass Google's reCAPTCHA. Despite being a popular solution, CAPTCHAs have faced criticism, especially from individuals with disabilities who may struggle with these tests, and from those who simply find these tests to be disruptive and time-consuming [27].

Another category of web bot detection techniques involves the use of Honeypots [28]. Honeypots are intentionally created webpages or links designed to "trap" web crawlers by making them perform certain actions that regular human visitors cannot see or perform. However, honeypot

[6] http://www.seleniumhq.org/.

techniques for bot detection have various drawbacks, such as their very limited effectiveness against sophisticated web bots [29].

To address all these issues, current research focuses on using machine learning-based detection techniques rather than relying solely on rule-based methods. The first step in utilizing machine learning models for the purposes of web bot detection requires that individual user sessions be extracted/represented from either server web logs or from in-browser mouse movement captures (behavioral analysis techniques).

Web bot detection approaches based on server logs analysis typically use "traditional" machine learning algorithms. Each visitor's web server session(s) are first extracted from web logs, and various measurable properties and characteristics of their behavior are calculated. These properties include access frequency to specific webpages, types of accessed web content (e.g., HTML, text, JavaScript, images, CSS), overall access patterns, and HTTP errors produced. Calculated feature values are then used as input to train machine learning models, which can then classify new visitors as bots or humans [30].

Web bot detection approaches based on in-browser mouse movement captures generally model each user's session/profile using high-level mouse-cursor actions like clicks, point-and-click, and drag-and-drop. These actions, along with their properties such as movement distance, duration, and efficiency, are subsequently used to train the classification model of choice [31]. An alternative way of modeling a user's session/profile is by representing their respective mouse-cursor trajectory in the form of an image, which is then directly used to train the classification model.

In summary, the categorization of web bot detection techniques can be effectively divided into three primary categories:

1) Challenge-response techniques, which may also incorporate machine learning.
2) Honeypots.
3) Behavioral analysis-based techniques, often utilizing machine learning, including web log and mouse/keyboard biometrics approaches.

In the remainder of this section, we provide in-depth overview of each of these three main categories of web bot detection techniques.

2.5.1 Challenge-Response Techniques

Distinguishing between human users and Internet bots in computer interactions is a challenging task. While bots can be programmed to perform tasks more accurately than humans, they may struggle with simpler tasks that come naturally to human users. This observation formed the basis for the development of the so-called challenge-response human verification schemes, such as CAPTCHA [32].

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) is a security mechanism developed to distinguish between human users and automated bots on websites [26]. It was initially introduced in 2003 by Luis von Ahn et al. [33]. CAPTCHAs play a pivotal role in safeguarding against automated bot attacks, thereby upholding the security and reliability of online systems.

The initial CAPTCHAs were primarily text-based, requiring users to decipher distorted characters to verify their human identity. In 2010, Motoyama et al. [34] conducted a comprehensive investigation into CAPTCHAs and their solvers, evaluating eight CAPTCHA-solving services against CAPTCHAs from 25 popular websites. Their study unveiled a success rate exceeding 70% for these services within a 20-second timeframe on most websites. The authors argued that defenders were starting to gain an advantage over cost-effective automated software solvers. Despite the significant costs associated with these solvers, their performance remained unsatisfactory. Consequently, image-based CAPTCHAs, including variations that involve text, rose to prominence as a more effective solution. Apart from the familiar text and image-based CAPTCHAs, a variety of other CAPTCHA types have also emerged [27, 35]. These include audio-based CAPTCHAs, tailored for users with visual impairments. These users are required to transcribe or understand spoken words or digits. Another innovation is the video-based CAPTCHA, which presents video content and prompts users to identify objects or patterns within the videos.

Math-based CAPTCHAs present users with mathematical problems that must be solved to pass the challenge. Slider CAPTCHAs engage users with interactive sliders or drag-and-drop elements, requiring them to manipulate the slider across the screen to complete the task. For a more interactive approach, game-based CAPTCHAs incorporate actual games into the verification process. Sensor-based CAPTCHAs utilize data from hardware sensors, often embedded in mobile devices equipped with gyroscopes or accelerometers.

Finally, behavior-based CAPTCHAs, as discussed in [35], are designed to assess user behavior and interactions to accurately classify users as either human or bot. In this regard, reCAPTCHA v3[7] falls under this category, employing behavioral analysis to evaluate user interactions without the need for explicit challenges, thus delivering a seamless and user-friendly experience. Indeed, the reCAPTCHA system comprises three versions: reCAPTCHA v1, v2, and v3. The original reCAPTCHA v1 was text-based and has been discontinued since 2018. reCAPTCHA v2 comes in two types: the first one presents users with a checkbox labeled "I'm not a robot" and requires them to select specific objects from a set of nine candidate images. The second type, known as reCAPTCHA v2 Invisible[8], eliminates the need for the checkbox but requires users to be associated with a button or invoked programmatically. This version analyzes various factors like IP addresses, cookies, and mouse movements to assess the risk and determine if additional challenges are necessary for verification. In contrast, the latest iteration, reCAPTCHA v3, does not involve explicit challenges but rather assigns a score to the user, enabling a more nuanced classification as either human or machine [35].

Additionally, there are Arkose Labs CAPTCHA [36], which combines AI and interactive elements like puzzles to offer a comprehensive approach to bot detection, and hCaptcha [37], which is similar to reCAPTCHA v3, utilizing behavioral analysis and offering adjustable difficulty levels for explicit challenges.

CAPTCHA technologies have undoubtedly evolved significantly to combat AI-based threats, but they are not without drawbacks. Unfortunately, several significant limitations still exist in this technology that can hinder their effectiveness in detecting bad bots. For example, one of these limitations is the fact that complex CAPTCHAs may frustrate legitimate users, leading to a poor user experience. Moreover, accessibility concerns arise with certain CAPTCHA types, like audio-based ones, which may not be accessible to users with disabilities. Additionally, advanced AI algorithms can bypass certain CAPTCHAs, compromising their effectiveness in safeguarding websites. The maintenance overhead of CAPTCHAs is another challenge, as regular updates are necessary to counter evolving AI attacks, making the CAPTCHA upkeep process very much resource-intensive. Furthermore, there is a risk of misclassification, as overly stringent CAPTCHAs might misidentify genuine users as bots, leading to false positives and potential user frustration.

[7] https://www.google.com/recaptcha.

[8] https://developers.google.com/recaptcha/docs/display.

While reCAPTCHA v3 offers a more seamless and frictionless user experience, it requires website owners to implement the score threshold effectively in order to strike the right balance between security and satisfactory user experience [27].

To maintain this delicate equilibrium, continuous improvements and novel approaches are necessary to effectively thwart sophisticated bot attacks while ensuring a seamless browsing experience for legitimate users. Ongoing research and innovation in CAPTCHA technologies are vital to stay ahead of rapidly evolving AI-based threats and provide robust security measures for online platforms.

2.5.2 Honeypots

Honeypots are deliberately created webpages or links that are designed to 'trap' web crawlers by making them perform certain actions (e.g., clicking on a hidden resource or a link) that generally cannot be performed/seen by regular human visitors. In [38], McKenna presented a model for detecting and classifying web robots using honeypots. In particular, the author constructed hidden resources, including PDF, DOC, and HTML files, using the CSS rule "*display:none*." The *sandtrap*, which is a server-side PHP script, executed the honeypots to catch crawlers. The author employed a "one-strike" rule for classification, where bots failing to check the robots.txt file or comply with its directives were classified as malicious, while those accessing the honeypot and complying with directives were benign. The model's performance was evaluated using logs from an academic website, and unfortunately it was found that honeypots were not effective in detecting sophisticated bots using deep-crawling algorithms with query generation.

In [39], the authors present Aristaeus, a system that aims to distinguish between good and bad bots based on their browsing activity. They conducted experiments to collect a large-scale dataset of automated browsing activity from various sources and analyzed the behavior of bots in terms of their request rates, user agent strings, and other characteristics. They found that bot behavior varied widely depending on the type of bot, with some bots exhibiting highly irregular behavior while others closely mimicked human browsing activity. Based on their analysis, the authors developed a machine learning-based classification model that achieved high accuracy in distinguishing between good and bad bots. They also proposed the use of honeysites as a means of capturing additional information about bot behavior and identifying new types of bots. The authors

conclude that their approach offers a promising means of detecting and mitigating the harmful effects of bad bots on the internet.

In 2015, a specialized system named Lino [40] was created to detect web crawlers through targeted data collection. Lino simulates a vulnerable webpage, ensnaring web crawlers in the process. Machine learning techniques, including Support Vector Machine (SVM) and decision tree C4.5, were applied to identify web crawlers by analyzing selected features contributing to visitor behavior classification. The top four influential features were identified: 1) Post data, indicating whether the client interacted with the fake form on the Lino system, 2) Session change, detecting alterations to the session identifier during the user's session, 3) Session duration, measuring the length of the session in seconds, and 4) Robots, assessing whether the user accessed the robots.txt file that defines robot conduct rules. Despite its potential, the detector exhibited a high false-positive rate in robot detection, potentially hampering web browser performance and user experience.

Our survey of honeypot-based detector approaches shows that a majority of these studies exhibit various limitations. These drawbacks encompass: a tendency to adversely impact the browsing performance of legitimate users, a propensity for yielding high false-positive rates, and a general lack of effectiveness when faced with more advanced and sophisticated web bots.

2.5.3 Behavioral Analysis-based Techniques

Behavioral analysis-based techniques have gained significant traction in both commercial applications and academic research as effective tools for distinguishing human behavior from automated actions across diverse contexts. In the realm of contemporary web bot detection strategies, commercial solutions often incorporate two primary methods: rule-based web bot detection, leveraging browser fingerprinting techniques, and behavior-based web bot detection, which scrutinizes visitor behavior encompassing mouse movements and browsing speed[9] [2]. Upon identifying a visitor as a potential bot through these methods, website administrators can implement

[9] Rule-based web bot detection, centered on browser fingerprints, encompasses a range of elements such as font detection, plugin enumeration, webGL fingerprinting, and the analysis of unique automation software strings within JavaScript variables [42, 43]. More advanced fingerprinting techniques have even emerged, capable of extracting nuanced properties like instruction-set architecture and memory allocator usage [44]. Nonetheless, it's important to note that certain limitations have been observed in current commercial fingerprint-based bot detection tools [42]. For instance, specific techniques may be tailored to particular automation tools and versions, resulting in the potential time-consuming maintenance of fingerprint lists [42, 45]. Additionally, these techniques can be circumvented by modifying automation tool fingerprints or substituting regular browsers for browsing automation software [42, 46].

various responsive actions, ranging from visitor blocking and content alteration to requesting human verification through visual challenges [41].

With the rise of AI-based techniques, companies offering bot protection solutions have adapted their offerings accordingly. Among the top six commercial bot protection solutions [47], which include Akamai[10], DataDome[11], Imperva[12], Radware[13], Cloudflare[14], and Vercara[15], Imperva and Vercara stand out for their expertise in utilizing advanced behavioral analysis and advanced machine learning algorithms as part of their solution.

However, a notable challenge within the realm of commercial web bot solutions is the frequent lack of transparency and adoption of closed-source approaches, which hinders researchers from analyzing or experimenting with their algorithms. This limitation obstructs independent verification and stifles innovation in the field. Consequently, gaining insights into detection models often requires exploring academic literature authored by researchers.

Conversely, within academia, researchers have employed diverse detection models based on behavioral analysis techniques to differentiate malicious bots. They have predominantly focused on two key approaches. The first approach revolves around utilizing server-side logs as the primary data source, focusing on analyzing information recorded in server access logs, including details about specific pages and requested resources, along with timing. The second approach involves leveraging client-side collected behavioral data, monitoring a user's interactions, such as tracking mouse movements and actions as they navigate through a website.

In the upcoming subsections, we will delve into a comprehensive review of proposed models based on these two distinct approaches as outlined in the literature.

2.5.3.1 Web Log-based Detection Models

Access to web content is facilitated through the client-server network processing, utilizing the HTTP protocol over the TCP/IP suite. Clients, such as web browsers operated by human users or web bots, interact with the server by sending HTTP requests. The server processes these requests and responds accordingly (see Figure 4) [48]. Typically, a user session on a website involves

[10] https://www.akamai.com/products/bot-manager.
[11] https://datadome.co/press/5-best-bot-protection-solutions-and-software-for-2023/.
[12] https://www.imperva.com/.
[13] https://www.radware.com/.
[14] https://www.cloudflare.com/en-ca/.
[15] https://vercara.com/.

multiple HTTP requests for different pages and embedded objects. To maintain user sessions and differentiate between legitimate users and bots, additional mechanisms like cookies[16] are used. Each HTTP request made by a client is recorded in a standard server access log file, which contains various fields, such as client IP address, timestamp, requested resource, HTTP method, status code, data volume, referrer, and user agent. The standardized format of server logs enables the preprocessing of log entries to extract individual request fields, facilitating the reconstruction of user sessions. A user session is characterized by a sequence of HTTP requests made by a client during a single visit to a website. Through analysis of the request fields, various statistical features can be derived, such as the total number of requests, number of page requests, percentage of page requests, percentage of successfully processed requests, among others [48].

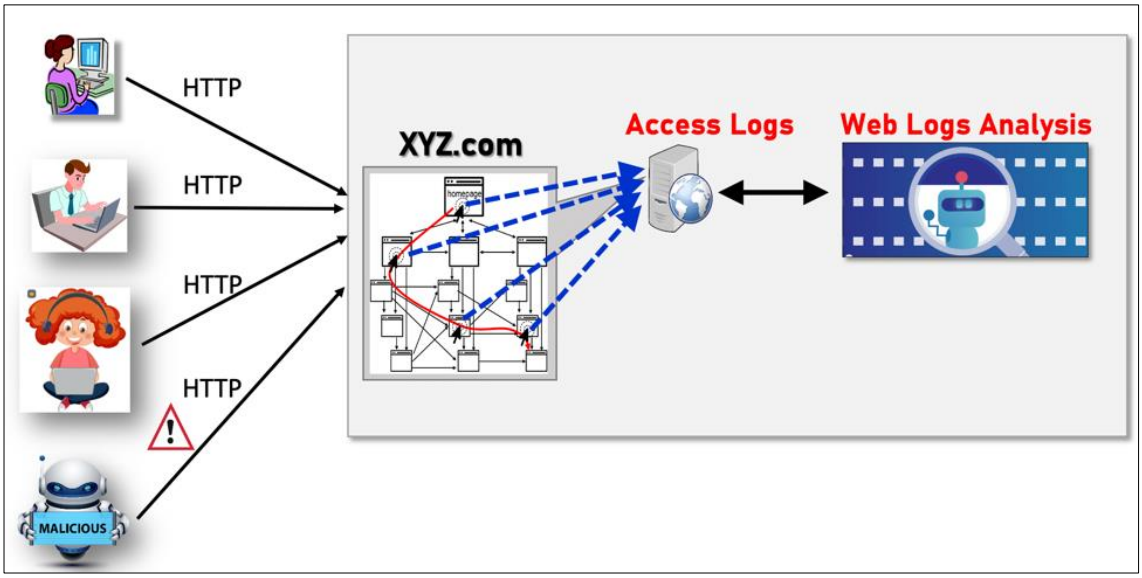


Figure 4. HTTP requests recorded in server access logs.

By extracting relevant features from the server log data and accurately representing user sessions, machine learning (ML) models can effectively identify patterns in both bot and human sessions. This capability extends to scenarios with or without labeled data, encompassing both supervised and unsupervised learning approaches. *Supervised learning* involves training the

[16] HTTP cookies are small pieces of data stored on a user's web browser by websites to remember preferences and track user interactions. They improve the user experience and help websites function efficiently [49].

algorithm on a labeled dataset, where input data and corresponding correct outputs are provided. The goal is for the algorithm to learn a mapping between inputs and outputs, enabling accurate predictions on new, unseen data. On the other hand, *unsupervised learning* trains the algorithm on an unlabeled dataset, where correct outputs are not provided. The algorithm seeks to find patterns or structures within the data without explicit guidance, such as clustering similar data points together [50].

Figure 5 illustrates the two primary machine learning-based approaches for web bot detection, which are classification and clustering. In some cases, pre-labeled training samples may be necessary for supervised data-mining algorithms to learn a classification model tailored to a specific dataset.

The literature reports several machine learning algorithms used for web bot detection, including Support Vector Machines [25, 51, 52], Random Forests [25, 53], Adaboost [25, 53], Multi-Layer Perceptron Classifiers [25, 51, 52, 54], Neural Networks [51, 55, 56], Ensemble Methods [25, 53], and Logistic Regression [55, 57]. Another popular method is the Bayesian network [58]. Haidar et al. [59] proposed a two-class Boosted Decision Tree (BDT) to identify malicious bots by analyzing website navigation behavior. The authors claimed that their model could be retrained as web bots evolve.

The primary drawback of bot detection solutions that employ supervised learning lies in their strong reliance on the session labeling strategy. The process of assigning ground truth labels to sessions, used in experimental analysis, introduces inherent errors. This is mainly because bots continuously evolve in terms of their sophistication and capabilities, outpacing the updates to expert knowledge-based databases of known bots' user agents and IP addresses typically used for session labeling. As a result, supervised learning solutions may lag behind in capturing the latest bot behaviors, indicating the potential advantages of adopting unsupervised learning approaches.

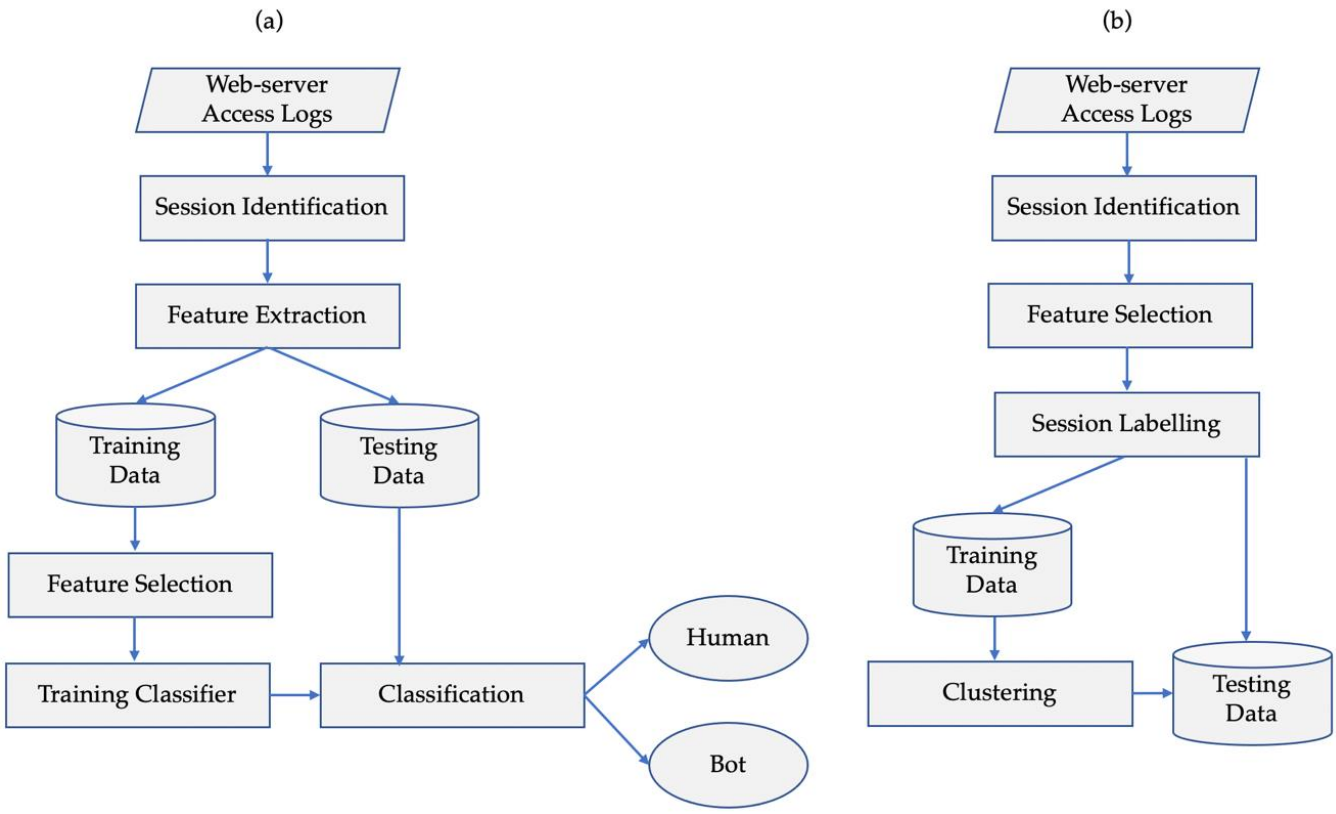


Figure 5. Web bot detection approaches that are based on web logs rely primarily on machine learning algorithms, a) the use of classification algorithm, b) clustering algorithm.

In recent years several methods were proposed to cluster user sessions in the context of distinguishing bots from legitimate users. Clustering-based models include: PSO-based clustering [60], SOM (Self-Organizing Maps) [61, 62], Modified ART2 (Adaptive Resonance Theory 2) [62], DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [63], MCL (Markov Clustering) Algorithm [64], K-means and Graded Possibilistic c-means [51], and Agglomerative Information Bottleneck [48].

Zabihi et al. [63] employed the Density-Based Spatial Clustering of Applications with Noises (DBSCAN) algorithm to categorize users into two clusters: one dominated by humans and the other by bots. Some outlier sessions were considered as 'noise' and remained unclassified. They considered 14 session features and eventually selected four features for clustering using the t-test. The clustering quality was evaluated using supervised-oriented metrics, with mean entropy and

purity scores of 0.024 and 0.966, respectively. However, unclassified sessions were not taken into account during the assessment.

Similarly, in the study conducted by Hamidzadeh et al. [65], an unsupervised learning approach was combined with feature selection to partition web bots and humans into clusters. They utilized Fuzzy Rough Set (FRS) theory to overcome the curse of dimensionality and reduced a 30-element feature set to 6-9 features. The Self-Organizing Map (SOM) algorithm was then applied to cluster the sessions. The clustering results were evaluated using entropy, purity, G-mean, and Jaccard measures, with mean rates of 0.37, 96.3%, 94.7%, and 88.3%, respectively. However, the study found that less than 60% of malicious bots, on average, were correctly assigned to bot-dominated clusters.

Both studies concluded that augmenting unsupervised classification with a feature selection stage has the potential to improve classification performance rates. This was further supported in another study [66], where Principal Component Analysis (PCA) was utilized to reduce the initial 40-feature set at the input of the K-means algorithm, aiming to separate bots from humans. The experimental analysis revealed that using the top principal components as new information-rich session features or selecting a subset of the most significant features indicated by PCA resulted in an increase in clustering purity from 94% to 96% for ten clusters.

In two other studies [51, 67], unsupervised learning techniques, namely K-means and Graded Possibilistic c-Means (GPCM), were applied for microclustering of bots and humans on an e-commerce site. Based on 22 session features, the model achieved recall and accuracy rates of 0.98 for binary classification of new user sessions.

The analysis of the literature indicates that applying unsupervised learning techniques to identify web robots is a relatively new and rapidly developing research area. However, the differentiation of web traffic patterns has been primarily focused on binary discrimination of humans from bots or a general, unclear division of bots into benign and malicious categories. Only one study, to the best of our knowledge [48], addresses the problem of web traffic patterns for multiple human and bot categories. They propose a novel approach that combines feature selection and unsupervised learning of HTTP-level traffic patterns to develop a user session classification model. The agglomerative Information Bottleneck (aIB) algorithm and other reference algorithms are employed for session clustering. The model is then used to classify new sessions into various profiles of bots and humans and label the sessions accordingly. Extensive experimental studies

based on real server log data demonstrate the ability of aIB clustering to distinguish user profiles and confirm the high performance of the classification model in terms of accuracy, F1, recall, and precision.

Despite their popularity, using web logs to detect bots poses several challenges, including complexity and time consumption, unavailability of server logs, difficulty in labeling positive cases in datasets with uncertain data, inability to detect and block web bots in real-time, and detecting only previously known bots. Therefore, feature selection and dimension reduction are crucial for developing high-performing web bot detection algorithms.

2.5.3.2 Mouse Movement-based Detection Models

Biometric-based methods provide a less intrusive and more continuous way to distinguish between human users and web bots. Namely, modern behavioral biometrics technologies detect bots through their actions rather than with interruptive user-unfriendly challenges.

In general, biometrics technology can be classified into two main types: static biometrics and behavioral biometrics [3]. Static biometrics modalities, such as fingerprints, face, voice, iris, retina, and palmprint, are physical features of a human that can be used to identify, verify, and authenticate that particular individual. Behavioral biometrics, on the other hand, are dynamic inputs such as keyboard typing and mouse dragging generated/produced by a human user that can also be used for the purposes of user identification and authentication.

When it comes to behavioral biometrics, it is well demonstrated that people perform differently when they hold or swipe a computer mouse, or type on a screen or keyboard, and these can be useful sources of data for user authentication, fraud detection, as well as bot detection systems. Modern-day behavioral biometrics technologies are capable of measuring a broad range of data inputs with a high level of accuracy and precision throughout a user session [3]. Examples of measurable behavioral-based parameters/actions include: key press, key or mouse-click sequence, key or mouse-click pressure, mouse-cursor motion, mouse-cursor acceleration, hit zone[17], and key flight[18]. Mouse movement related actions in particular have received increasing attention as a data source for detection of web bots for two main reasons: 1) the prevalence of key/keyboard actions during a user's interaction with a typical webpage is rare (if present at all),

[17] Hit zone offers the coordinates of touch screen events – or, locating exactly where the user touched the screen.
[18] Key flight is the time between two key press events.

2) most people have their own unique style of mouse usage that distinguishes them from other users as well as automated bots.

The conventional method for gathering mouse movements is illustrated in Figure 6 [2]. This involves embedding a JavaScript file on each webpage, which continuously records and stores the browser's mouse movements along with their timestamps[19]. The data is sent back to the server either when the visitor clicks the mouse or periodically, every few seconds. The collected data forms a sequence that includes all the points where the visitor performed mouse movements, along with their respective timestamps. By analyzing this data, we can distinguish and characterize different behavioral patterns for humans and bots.

To date, mouse movement has been used in many different applications, such as identifying genders [68], online learning and educational technology applications [69], social cognitive theory [70], finding reading patterns on websites [71], enhancing psychological science [72], and user authentication and identification tasks [73, 74], as well as for bot detection [1, 75].

In our research, we have made a substantial and valuable contribution to the field of malicious web bot detection through the application of behavioral analysis, with a specific focus on mouse movement. This approach has proven to significantly improve the accuracy and efficiency of detection methods. Our findings and contributions are extensively discussed in Chapter 4 of our study, where we delve into the methodology, present detailed experimental results, and highlight the outcomes of our investigations.

[19] A more comprehensive insight into the process of collecting mouse movement data will be provided in Chapter 5.

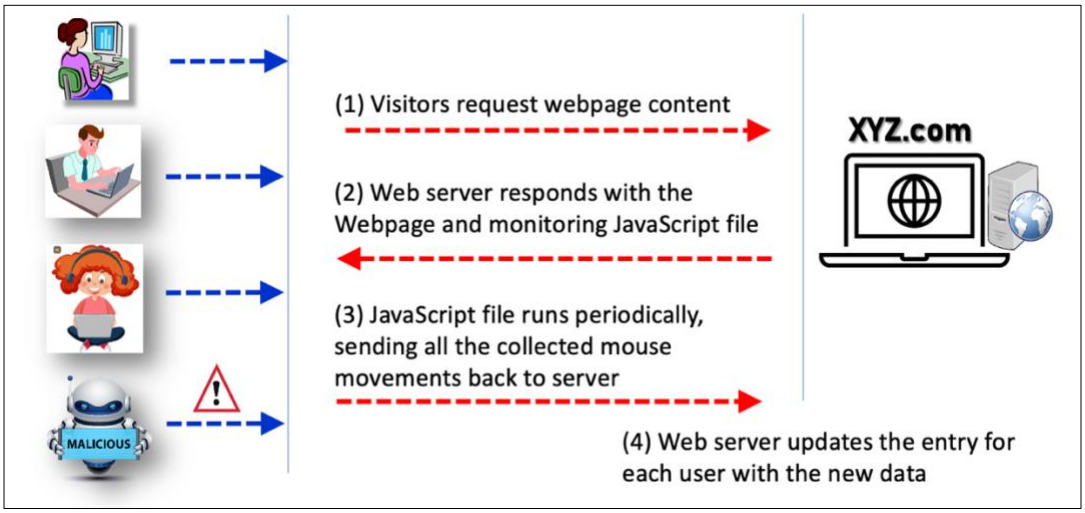


Figure 6. Mouse movement collection process.

2.6 Conclusions

In this chapter, we explored the rising prominence of web bots and their increasing sophistication in mimicking human behavior, presenting challenges for detection. We traced the historical evolution of web bots and their impact on various industries, highlighting significant threats to customer interactions, data security, and marketing efforts. Our review of web bot detection techniques emphasized the need for innovative strategies and advanced machine learning based solutions to effectively counter these evolving threats.

In the following chapter, we introduce an unsupervised machine learning-based detection model that incorporates a fully automated feature selection procedure and has been validated on a real-world dataset. This model was developed as the first step (i.e., attempt) forward in our efforts to design an effective system capable of combating and mitigating the adverse effects of modern-day malicious bots.

Chapter 3

Unsupervised ML-Based Detection of Malicious Web Sessions with Automated Feature Selection: Design and Real-World Validation

In this chapter we address the challenge of distinguishing human web sessions from those generated by malicious bots based on server log data, and we emphasize the importance of appropriate feature selection in the dataset preprocessing phase. Furthermore, we propose the use of the Gradient Boosting Technique to automatically identify the most significant web session features, and we then integrate these features into an ML system for web session classification based on the unsupervised Self-Organizing Map algorithm. We validate the proposed system using a real-world server-log dataset. The obtained experimental results demonstrate high effectiveness of our approach in identifying malicious web sessions while also providing valuable insights into the nature and scope of the observed attack.

3.1 Introduction

As web bot technologies continue to evolve in their sophistication while becoming ever-more evasive, the task of separating human web sessions from those generated by malicious bots also becomes increasingly more challenging. To date, many research studies have proposed the use of advanced ML-based methods for automated differentiation between web bot and genuine human sessions. (Some of these techniques have been surveyed in Chapter 2.) Unfortunately, most of the existing works that rely on the use of ML-based techniques for web bot detection based on server logs tend to overlook the importance of adequate feature selection during the dataset preprocessing stage. Namely, instead of making the process of feature selection automated and optimized to each particular dataset, these studies generally resort to the use of the same fixed set of hand-picked web-session attributes. It is well known, however, that suboptimal approach to feature selection is

likely to result in suboptimal performance of the respective ML algorithm and, consequently, of the entire system.

In this chapter, we present several key contributions of our work. Firstly, we propose using the Gradient Boosting Technique to automatically identify the most significant web session features (from an extensive initial list of features) for any server-log dataset. Secondly, we integrate this automated feature selection technique into a web session classification system deploying the unsupervised Self-Organizing Map algorithm. Thirdly, we validate the performance of our integrated system on a recent real-world dataset collected during a large-scale attack on our academic institution. The experimental results demonstrate high effectiveness of our proposed system in identifying malicious web sessions and provide valuable insights into the nature and scale of the conducted attack.

The chapter is organized as follows. The related work is reviewed in Section 3.2, while Section 3.3 introduces our Apache Spark log analyzer and the extensive initial list of 119 web-session features implemented in it. Section 3.4 introduces the real-word dataset which our analysis was based on, and it also presents our gradient boosting feature (GBF) selection method which was employed to identify the most significant of the initial 119 features. This section also discusses the experimental results obtained using our proposed GBF feature-selection approach on the deployed dataset. Section 3.5 discusses the motivation for using SOM algorithm in our analysis and presents the web-session mapping and clustering results acquired using a 2D SOM algorithm. Sections 3.6 and 3.7 highlight some additional interesting observations of our research. Finally, Section 3.8 closes the chapter with a summary of the most significant findings and possible directions for future work.

3.2 Related Work

In this section, we provide a concise survey of a restricted set of literature that specifically addresses the issue of feature selection in server-log datasets, which is directly relevant to our research as presented in this chapter. However, for a more comprehensive and extensive overview of the broader existing literature on web bot detection, we refer readers to Chapter 2, Section 2.5.

In [48], Suchacka et al. investigated the issue of how to detect and differentiate between advanced web bots and humans. They proposed a novel approach that combines feature selections and an unsupervised machine learning technique to cluster user sessions with the agglomerative

Information Bottleneck (aIB) algorithm. They applied the Fisher Score algorithm and a subset of the most relevant features based on experimentally driven clustering entropy rates. They demonstrated that most genuine users had similar online behaviour and their sessions were partitioned into a very small number of subgroups. In contrast, web robots showed a variety of navigational patterns and their sessions were spread across several clusters.

In terms of identifying simple web bots from advanced bots, which try to hide their main purpose and operation, Iliou et al. [25], presented a detection model which was a combination of rule-based and machine learning techniques. They used 23 different detection attributes for each user session and incorporated 4 classification algorithms. They generated the ground truth to train their model using an automatic annotation mechanism. It was demonstrated that their model was able to examine the fingerprint of the visitor (the agent name) and check its IP to see if it has shown malicious activity using an external honeypot.

Over and above that, addressing the problem of detecting malicious and non-malicious website visitors using unsupervised learning method has been studied by Stevanovic et al. [76]. They examined the use of two unsupervised neural network (NN) learning algorithms for the purpose of web log analysis. Particularly, they applied Self-Organizing Map (SOM) [77] and Modified Adaptive Resonance Theory 2 (Modified ART2) [78] to discover the relative differences and/or similarities between malicious web crawlers and other non-malicious visitor groups. They defined 10 different detection attributes that identify and distinguish between automated and human visitors to a website.

Considering different web bot detection models in the previous related works, the goal of our research was to investigate the behavior of web bots using a much wider range of detection features within a server-log dataset, with the ultimate hope of being able to identify and examine a much broader range of malicious bots. According to our knowledge, our work is one of the most comprehensive real-world dataset based bot-detection studies to date, utilizing Apache Spark and the Gradient Boosted Feature Selection algorithm. The utilized real-world dataset was collected during a confirmed large-scale attack on our academic institution, providing practical insights for the wider cybersecurity community.

3.3 Spark Log Analyzer

Apache Spark is an open-source framework for analyzing and modeling structured and unstructured data at scale [79]. One of the most beneficial features of Spark is its in-memory cluster computing that increases the processing speed of associated applications. In the course of our study, we developed an Apache Spark based server-log analyzer. The purpose of this analyzer was to: (1) scan the entries in the provided web logs and find unique visitor sessions, and (2) analyze the found sessions with the aim of extracting their key features and building their feature-vector representations.

In the remainder of this section, the details of how our Spark analyzer identifies individual user session representations within a provided server-log file will be discussed. The actual set of utilized user session features will also be reviewed.

3.3.1 Session Identification

The first objective of our Apache Spark log analyzer is to perform session identification by grouping together all HTTP requests that have originated from the same IP address and that carry the same user-agent string. Following that, a timeout approach is applied to break this grouping into unique sessions. Subsequently, the Spark tumbling windows function is used to discretize a day into 6-hour buckets[20] in order to ensure scalability. A time-window of 30-min is then applied to distinguish between different sessions corresponding to the same user/IP.

3.3.2 Features

From the previous related studies, and as summarised in [30], web session attributes can generally be divided into six classes: content, frequency, sequence, size, time, and workload. These classes depict the type of information that one could extract and interpret by analyzing an identified web session in a server-log dataset. Now, the purpose of most attacks on web servers is to cause an overload of their communication and/or processing resources. Consequently, it can be said that the "frequency", "time", and "content" attributes comprise the largest portion of attributes that would

[20] The 6-hour window in the Apache Spark log analyzer was selected through empirical experiments to capture diverse user behaviors effectively throughout the day. This duration strikes a balance between granularity and efficiency, considering varying rates of user website access. The choice, validated through empirical testing, aims to facilitate practical session identification, aligning with diverse user engagement patterns.

likely be found in web sessions generated by malicious bots during an attack on a web server. Considering that the goal of our work was to build a highly accurate and effective system capable of detecting malicious web sessions, we have adopted most of the web-session features from those three classes of attributes. Additionally, to improve the detection capability, our system also identifies and includes a number of new features based on different MIME (Multipurpose Internet Mail Extensions) types found in an HTTP request. A MIME type is a standard that indicates the nature and format of a document, file, or assortment of bytes, and is used by browsers to determine how to process the respective HTTP request [80]. We believe that deploying this type of information/feature is useful in capturing the differences in traffic patterns of web bots and legitimate web users. A sample of web-session features that were implemented in our Spark-based log analyzer is shown in Table 1. The complete list of the MIME type based features (features 22 to 119) can be found at http://www.cse.yorku.ca/~shadisa/.

Table 1. The set of user session features.

	Feature and Description
1	*n_requests*: the number of requests in one single session.
2	*average_size:* the average size of the object returned to the user, measured in bytes.
3	*total_size:* the total size of the object returned to the user in one single session.
4	*count_of_robots_txt:* if user accessed robot.txt file or not.
5	*fraction_of_consecutive_sequential:* the ratio of requests that have the same depth (i.e. parent segments) in one session.
6	*std_of_request_depth:* it calculates the amount of variation of the requests' depth in a session, (the standard deviation of the size of the segments).
7	*fraction_of_missing_referer:* the percentage of unassigned referrer field requests.
8	*fraction_of_malformed_request:* the percentage of Bad Requests in a session, e.g., malformed request syntax, invalid request message framing, etc.
9	*fraction_of_informational*:it is the percentage of the requests that have been received but are still being processed.
10	*fraction_of_success:* the type of HTTP-Status Codes, success, the percentage of the actions that are successfully received, understood, and accepted.
11	*fraction_of_redirect:* the type of HTTP-Status Codes, redirect, the percentage of the actions that must be taken in order to complete the request.
12	*fraction_of_client_error:* the type of HTTP-Status Codes, client error, the percentage of the requests containing incorrect syntax or that could not be fulfilled.
13	*fraction_of_server_error:* the type of HTTP-Status Codes, server error, the percentage of the server that failed to fulfill an apparently valid request.

14	*fraction_of_GET:* the percentage of HTTP requests of type GET.
15	*fraction_of_POST:* the percentage of HTTP requests of type POST.
16	*fraction_of_HEAD:* the percentage of HTTP requests of type HEAD.
17	*fraction_of_DELETE:* the percentage of HTTP requests of type DELETE
18	*fraction_of_PATCH:* the percentage of HTTP requests of type PATCH.
19	*fraction_of_OPTIONS:* the percentage of HTTP requests of type OPTION.
20	*fraction_of_CONNECT:* the percentage of HTTP requests of type CONNECT.
21	*session_durartion_in_seconds:* the total time (in seconds) between the first and the last HTTP request of a session.
22-119	*fraction_of_mime_image/jpeg:* the ratio of webpage to image requests in a session, *fraction_of_mime_application/pdf:* the ratio of webpage to pdf requests in a session, *fraction_of_mime_text_css:* the ratio of webpage to txt_css requests in a session, etc.

3.3.3 Session Labeling

Once the feature-vector representation of an identified web-session is generated, our Spark log analyzer pre-labels this feature-vector as belonging to one of the following five categories: Normal Traffic, Suspicious Traffic, Malicious Traffic, Known Crawler, and Unknown Traffic. It is important to note that these labels are not used before or during the learning process. Instead, they are (only) intended to later on help us understand the final result of the unsupervised clustering performed on all identified web-sessions (i.e., respective features vectors), as well as to help us assess the clustering performance.

The algorithmic steps of the session pre-labelling process are outlined below:

1) Any feature vector that corresponds to a web session whose user agent string matches a user agent string of a well- behaved crawler and a session that has an IP of a well- behaved crawler, is labelled as a Known Crawler.

2) Any feature vector that corresponds to a web session whose user agent string matches a user agent string of a malicious crawler or a session that has a malicious IP address is labeled Malicious Traffic (Log analyzer maintains a table of blacklist IPs and a table of user agent fields of all known, both malicious and well-behaved, web crawlers from the data found in repositories [81], [82], and [83]). In addition, any session that would normally belong to a human visitor (i.e., normal traffic) except it accesses either the ‘robots.txt’, ‘security.txt’, or ‘sitemap.xml’ files, is also labelled as Malicious Traffic.

3) Any feature vector that does not match 1) and 2), and its user agent string also does not match a user agent string of a known browser, or it contains a typographical mark as a username is labelled as Suspicious Traffic.

4) Any feature vector that corresponds to a web session whose user agent string matches a user agent string of a known browser, and the session does not access the 'robots.txt', 'security.txt' or 'sitemap.xml' files is labelled as Normal Traffic.

5) All other web sessions are labelled as Unknown Traffic. The overall dataset labelling process is also depicted in Figure 7.

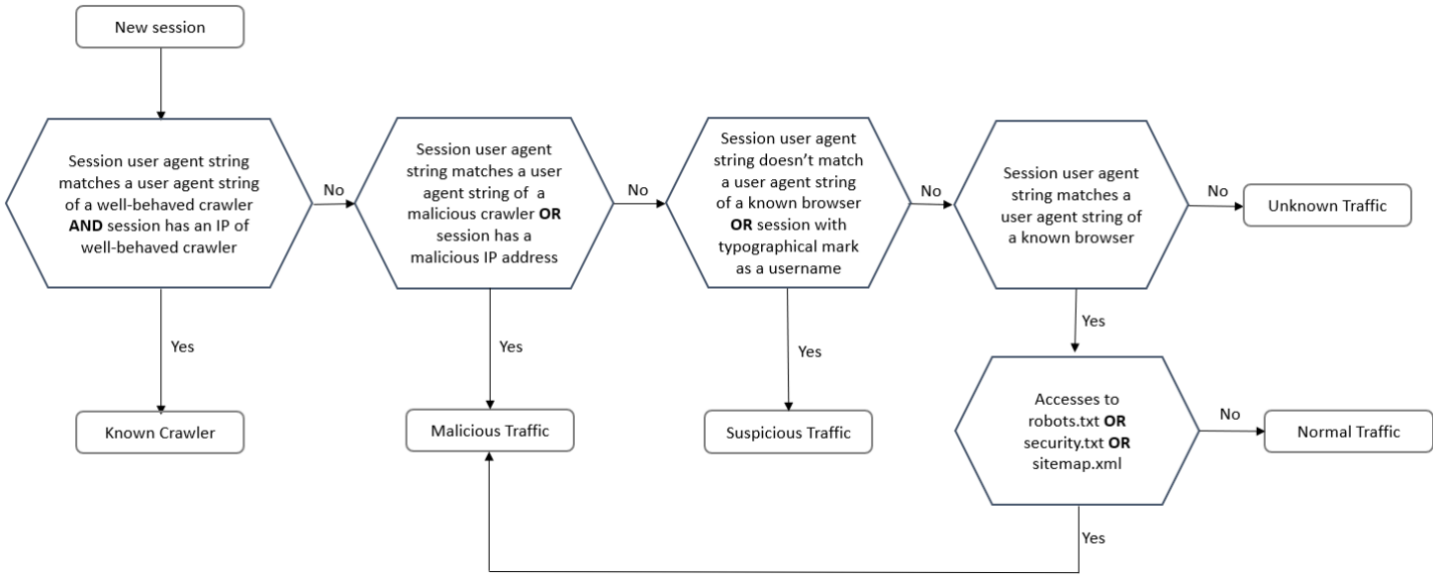


Figure 7. Dataset labelling flow chart.

3.4 Server-log Dataset and Feature Selection using Gradient Boosting

This section gives an overview of the real-world server (i.e., access) logs dataset that was made available to use for the proposes of this research, as well as the gradient boosting feature

selection method utilized to identify the most significant web-session features pertaining to the given dataset.

3.4.1 Dataset

The log file/dataset which our work has focused on, and which we have fed to the Spark log analyzer, was provided by the York University's EECS department. (In the remainder of this thesis we will refer to the dataset contained in this file as YorkU-EECS dataset.) The log file contains detailed information about web requests into the domain www.cse.yorku.ca recorded during a 6-week interval – from March 29, 2020, to May 10, 2020. The logs are known to be tampered with a multi-vector attack which happened on May 01, 2020. Each file entry contains information in the following order from left to right: IP address of the source of the request (e.g., 46.229.168.138), the timestamp of the request (e.g., 26/Apr/2020:04:49:04 -0400), the HTTP method (e.g., GET), the file on the server that was requested (e.g., /teaching/docs/javaapi/index.html?javax%2Fxml%2Ftr ansform%2FURIResolver.html), the response code from the server (e.g., 200), the size of the data retrieved from the server (e.g., 2908 bytes) and the user-agent file. Table 2 lists the total number of requests and the distribution of different sessions in the EECS department's log files as per labeling process of our Spark log analyzer previously described (see Figure 7).

Table 2. The distribution of dataset.

	Number of Sessions
Total number of requests	291032
Total number of sessions	205884
Total number of Normal Sessions	152025
Total number of Malicious Sessions	9051
Total number of Known Crawler Sessions	92
Total number of Suspicious Sessions	44704
Total number of unknown visitor Sessions	12

3.4.2 Feature Selection using Gradient Boosting

Ideally, there are four conditions that should be satisfied by a feature selection method applied to any dataset intended for the training of an ML-based model, if this ML model is to produce meaningful and accurate results [84]. The first condition is that the method is able to identify non-linear feature interactions. The second condition is that the method is capable of reliably extracting the most relevant features. The third condition is that the method scales linearly with the number of features and dimensions in any given dataset. The final condition is that the method allows the incorporation of a predefined sparsity structure. In this work, we have applied a notable feature selection algorithm, Gradient Boosted Feature Selection (GBFS) [84], which satisfies all four of these requirements. GBFS is flexible, scalable, and remarkably straightforward to implement as it is based on a modification of Gradient Boosted Trees. An advantage of using gradient boosting is that after the boosted trees are built, it is comparatively straightforward to retrieve importance scores for each attribute. The importance score indicates the level of usefulness for each feature in the construction of the boosted decision trees within the model.

After applying the GBFS to our dataset, we calculated the importance value for each attribute in the dataset, allowing attributes to be ranked and compared to each other. GBFS was implemented in our system using GB python library [85]. The plot in Figure 8 shows 11 features that have ended up receiving the highest importance values out of the entire original feature set (consisting of 119 initially selected features, as previously discussed). Figure 8 clearly shows that the “fraction _of _GET” has the highest and that the “fraction _ of _ mime _ application _xml” has the lowest importance score among the top 11 scoring features.

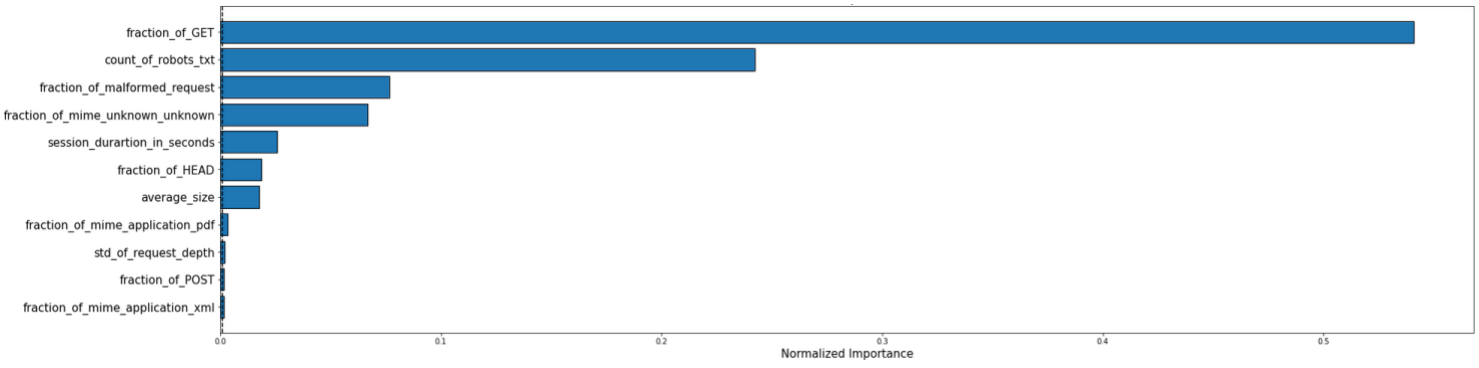


Figure 8. Feature importance plot.

3.5 Dataset Evaluation using SOM Algorithm

In this section, we delve into the assessment of YorkU-EECS datasets described in Section 3.4.1 using the Self-Organizing Map (SOM) algorithm.

3.5.1 SOM Algorithm

The self-organizing map (SOM) is an unsupervised ML technique widely used for the purposes of data visualization and data exploration [86]. Notably, it is an artificial neural network which use unsupervised (competitive) learning to produce a low-dimensional representation of high dimensional data by "fitting" a grid of nodes to the input dataset over a fixed number of iterations. Neurons (also called nodes or reference vectors) can be typically assembled in a single 2-dimensional grid of a rectangular or hexagonal shape. SOM produces a discretized representation of the input space that preserves topological properties of the input data set and is robust to statistical anomalies [76]. Through dimensionality reduction and topology preservation, it can help us uncover different patterns and categories of data in large datasets. In practice, the quality of the final results produced by a SOM is greatly influenced by a number of factors, including: the initial weight of the map's neurons, the neighborhood function, the learning rate, the sequence of training vectors and the number of learning iterations [87]. In general, SOM initialization approaches are grouped into two classes: random initialization and data analysis-based initialization. In the first method, a number of learning attempts with different initial configurations are usually made, and the best among them is adopted in the end. However, in the second approach, certain statistical data analysis methods are applied beforehand to establish the best initial configuration. The linear principal component analysis (first eigenvectors corresponding to the largest eigenvalues of the empirical covariance matrix) is a popular method to select the initial SOM weights. In this work, we have utilized the principal component initialization (PCI) method to construct the initial SOM configuration.

3.5.2 Training SOM and Visualization

The experimental verification of our proposed approach for web session classification and detection, utilizing the available YorkU-EECS dataset, was performed using the SOM implementation in Python by a package called Tfprop_somp [88]. The data cleaning, preprocessing

(e.g., removing zero values) and visualization work were performed using several other popular python packages[21]. SOM neurons were arranged in an 11-by-11 (experimentally) hexagonal composition. The input vectors were normalized before being fed to SOM. It should be noted that, even though SOM algorithm is typically used for the purpose of data visualization and dimensionality reduction, the Tfprop_somp Python package provides an additional feature which enables automated identification of the main groups/clusters within the formed map using K-means clustering algorithm. We applied the K-means algorithm to all input data to gain a preliminary understanding of the data distribution. This process involves clustering all input data. Through this, we establish correlations between each cluster and the corresponding number of hits in each neuron, as visualized in the BMU (Best Matching Unit) visualization (refer to the next section). This methodology provides valuable insights into the distribution of hits within distinct clusters, facilitating the analysis of patterns and relationships within the dataset. Figure 9 displays the results obtained by detecting the main clusters in the SOM model/map trained on our input dataset, with the number of clusters being set to 5 to match our 5 assumed categories of web traffic: normal traffic, suspicious traffic, malicious traffic, known crawlers, and unknown crawlers. The numbers labeled on the cluster map are ID numbers in which each cluster can be correlated to a traffic category in our training dataset. From a statistical perspective, it can be stated that the neurons of cluster ID 1 were mostly fired by 'normal sessions', and the neurons of cluster ID 4 were predominantly fired by 'suspicious sessions'. However, the neurons of cluster ID 2, 3 and 5 were fired by a mix of sessions from different user groups, and thus warranted a further investigation. In the following sub-section, the approach used, and the results obtained trying to acquire more clarity on the data content of different clusters within the trained SOM, including clusters 2, 3 and 5, is explained.

[21] Such as numpy, pandas, scipy, matplotlib, seaborn, and sklearn [89].

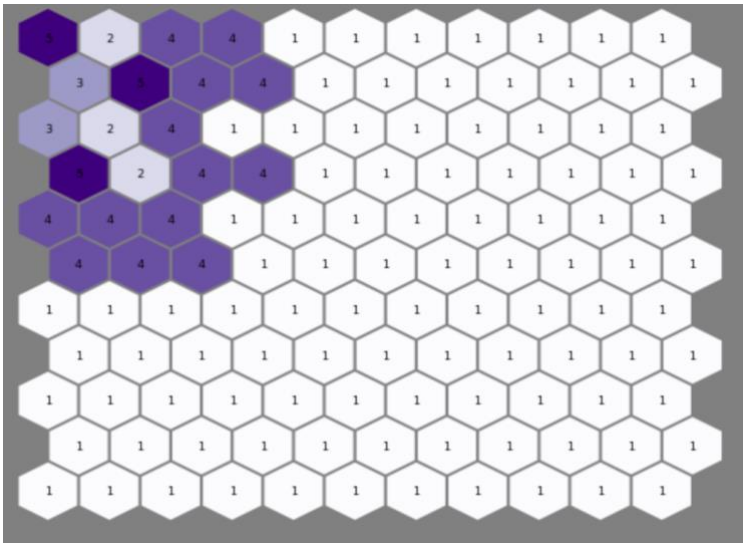


Figure 9. Unsupervised clustering visualization, all traffic heatmap.

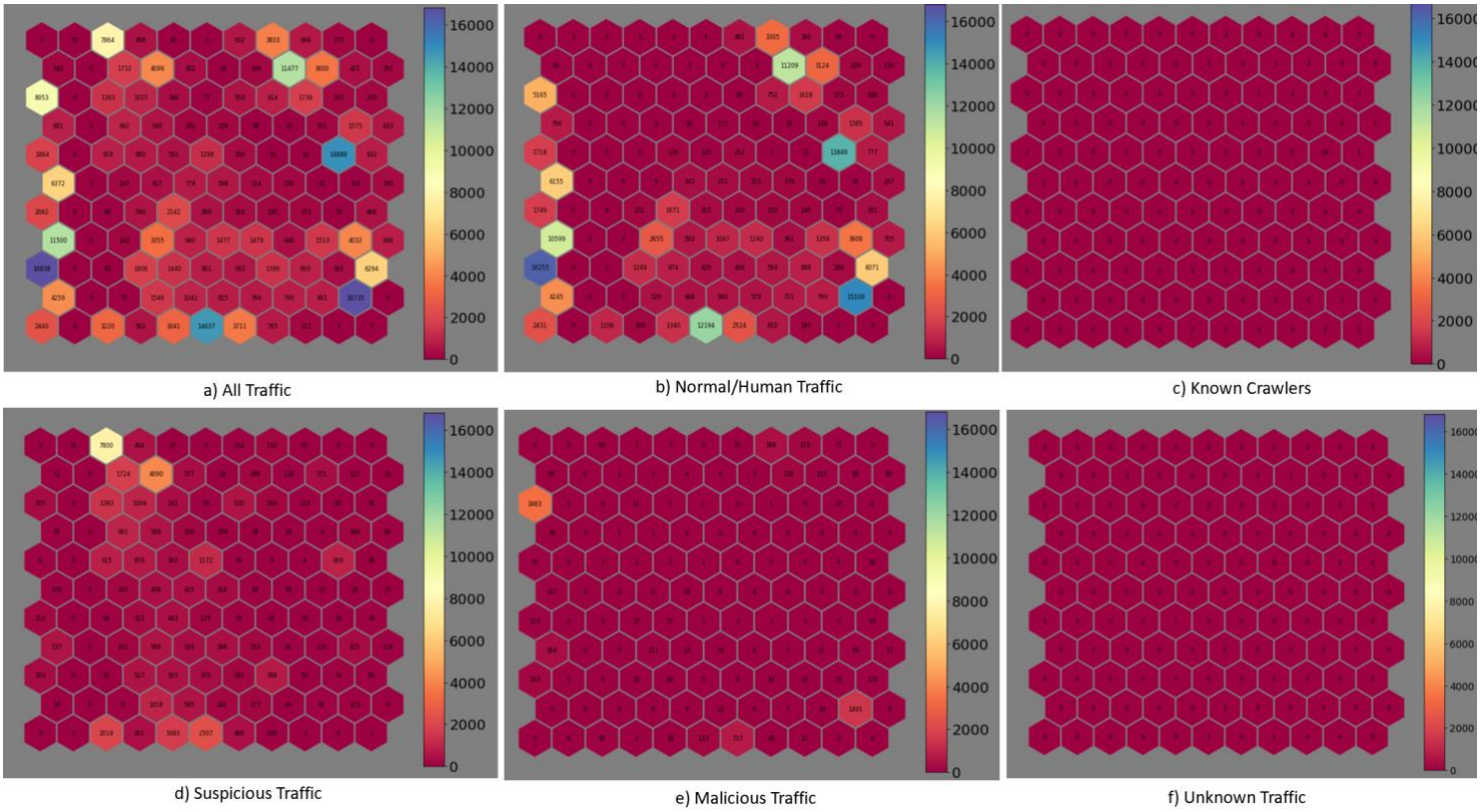


Figure 10. BMU heatmap of all sessions.

3.5.3 BMU Visualization

Using SOM, each sample in the training set has a corresponding BMU (best matching unit) in the map. This is the cell/unit that the given sample is most similar (i.e., closest) to. We applied this concept for all the sessions in the training dataset to identify which cells/neurons of the trained SOM are their respective BMU. Subsequently, the heatmap visualization was applied to display the obtained result, as shown in Figure 10. The color bar of the heatmap was set in a way that the blue color represents the highest number of sessions that hit the given neuron on the map and the red color represents the lowest number. Figure 10.a) belongs to the BMU of all training sessions. Figures 10.b) to 10.f) show the BMU (i.e., neuron hits) for sessions that were labelled as belonging to Normal Traffic, Known Crawler, Suspicious Traffic, Malicious Traffic, and Unknown Traffic, respectively.

From the obtained maps, the following conclusions can be drawn:

• Normal traffic: Figure 10.b) represents the distribution of normal traffic and appears very compatible with the cluster ID 1 in Figure 9.

• Malicious Traffic: Shown in Figure 10.e) is the distribution of malicious traffic. As can be seen from this figure, there are 2 hexagonal bins (i.e., neurons), which are the BMUs of the largest portion of the malicious sessions – one neuron placed in the upper left corner of the map matching 3463 malicious sessions (i.e., hexagon bin colored orange) and the other neuron placed in the lower right corner matching 1492 malicious sessions (i.e., hexagon bin colored bright red).

A detailed comparison of Figures 10.b) and 10.e) reveals the following:

a) The upper-left neuron/hexagon in Figure 10.e), in addition to being the BMU for 3463 malicious sessions, is also the BMU for 5165 normal sessions, as shown in Figure 10.b). However, based on the cluster map in Figure 9, this neuron belongs to cluster ID 3 and is completely separated from cluster ID 1, which comprises the majority of normal sessions. Our detailed investigation of these specific sessions (as they appear in YorkU-EECS file) has revealed a crucial fact: they are indeed malicious traffic attempting to bypass website security by not accessing the ‘robots.txt’, ‘security.txt’, or ‘sitemap.xml’ files and/or falsifying the value of the user agent string field.

b) The lower-right neuron/hexagon in Figure 10.e), in addition to being the BMU for 1492 malicious sessions, is also the BMU for 15109 normal sessions, as shown in Figure 10.b). Furthermore, based on the cluster map from Figure 9, this neuron belongs to Cluster ID 1 in Figure 9 and therefore shares similarities with other neurons that have attracted the majority of normal sessions. Upon further investigation of these specific sessions (as they appear in YorkU-EECS file), it was revealed that they are either sessions of harmless web spiders or of regular human users.

• Suspicious traffic: The distribution of suspicious traffic is depicted in Figure 10.d). Clearly, the upper left corner of the map compromises the BMUs for the majority of suspicious sessions. (Recall, we consider a session to be suspicious if either its user agent string does not match a user agent string of a known browser or it contains a typographical mark as a username). It is interesting to observe that the Cluster map in Figure 9 recognizes these sessions as an independent cluster with cluster ID 4. An in-depth investigation of these sessions (as they appear in YorkU-EECS file) was conducted in order to analyze their underlying characteristics, will be further discussed in Section 3.5.4 b). Additionally, it can also be noticed in Figure 10.d) that there are some suspicious sessions that fired the neurons in the region dominated by normal traffic in the lower left corner of the map. Our in-depth investigation of these session uncovered an interesting reason behind their misplacements: they belonged to legitimate users attempting to log in to certain YorkU servers/services with mistyped usernames. The misplacement of these sessions suggests that, preferably, the website's security system administrator would conduct further investigation of any session that ends up falling into ‘suspicious sessions’ cluster/category, as failed login attempts could be caused by benign mistakes of honest/regular users, but they could also be an indication of a brute force login/password attack.

• Known crawler: Figure 10.c) shows the distribution of well-behaved web crawler sessions. There is an overlap between fired neurons in this map and normal sessions in Figure 10.b). This overlap is likely the result of the statistical dominance of training-data corresponding to human sessions (see Table 2). (Recall from Table 2, there were only 92 sessions in the dataset labeled as Known Crawler.) Namely, as pointed out in [76], the SOM algorithm tends to produce results that are dependent on the input data density; such

that higher density data clusters tend to ‘win-over’ a greater number of SOM neurons, regardless of their inter-cluster variance.

• Unknown traffic: According to the initial definition of unknown traffic from Section 3.3.3, these sessions deploy user agent strings that are not known and do not match well-behaved nor malicious traffic according to [81] and [82]. They also do not access the ‘robots.txt’, ‘security.txt’ or ‘sitemap.xml’ files. A comparison between Figures 10.b) and 10.f) shows that there is a significant overlap between fired neurons in the respective maps. This leads us to conclude that most unknown sessions are likely normal traffic generated by benign users/visitors likely deploying unconventional browsers (i.e., means of webpage retrieval)[22].

3.5.4 Abnormal Traffic Analysis

In this sub-section, we take a closer look at two particular groups of sessions from our dataset – malicious and suspicious – in order to analyze their underlying characteristics and identify the features that could assist web intrusion detection systems in differentiating this traffic from normal/human sessions.

a) Malicious Sessions: Table 3 displays the IP addresses associated with web sessions from our YorkU-EECS dataset that are labelled as malicious, including:
   - The type of worldwide malicious activities associated with these IP addresses as reported by [90, 91],
   - Respective geolocations,
   - Respective DNS names,
   - Number of malicious sessions with these IP addresses observed in the YorkU-EECS dataset from March 29th to May 10th.

These sessions belong to the upper-left corner of the map in Figure 10.e). Upon closer inspection of YorkU-EECS logs, it was evident that these sessions were generated by attackers who tried to ‘brute force’ login into EECS system through EECS website. Even though it cannot be determined with absolute certainty whether these sessions were generated by malicious humans or

[22] We chose to discard the small amount of unknown traffic in our analysis to streamline resources and focus on known patterns of normal and malicious behavior. This decision was made to reduce noise and false positives, optimizing our efforts for a more efficient and targeted analysis.

malicious bots, in our analysis they were mostly identified as bad web bots. It is encouraging, nevertheless, that no matter their actual origin, all these sessions were correctly identified as 'malicious' using our proposed detection model[23].

b) Suspicious Sessions: Another in-depth analysis was conducted on suspicious traffic, which is located on the upper-left corner of the map in Figure 10.d). Our in-depth analysis revealed that most of the suspicious sessions - which have user agent string that either did not match a user agent string of a known browser or contained typographical mark as their username - came from India, Pakistan and the African countries of Zambia, Ghana, Kenya, Uganda, and Nigeria. We also took a closer look at this particular group of sessions, using more resources [92, 93], and we found that there exists a limited number of borderline "bad activities" associated with some of the observed IP addresses from this group (see Table 4). This led us to conclude that in a real-world system deploying our solution a further inspection by the website's security system administrator may be required to examine/prevent any possibility of illegitimate activities caused by this particular (gray-area) category of traffic.

[23] It is worth noting that these malicious bots adhered to the rules we had previously defined (refer to Section 3.3.3).

Table 3. Examples of malicious traffic in the dataset.

Originated IP Addresses	Bad activities Reported	Geographical Location of the IP Addresses	DNS name	Number of Identified Malicious Sessions in EECS Dataset
111.202.100.81, 111.202.100.82, 111.202.102.102, 111.202.101.92	Malicious brute force vulnerability hacking attacks	China	chinaunicom.com	6
123.125.67.157, 123.125.67.158, 123.125.67.160, 123.125.67.164, 123.125.67.221, 123.125.71.14, 123.125.71.47, 123.126.113.248, 123.126.68.144	Malicious brute force vulnerability hacking attacks, Bad Web bot	China	chinaunicom.com	63
220.181.51.112, 220.181.51.115, 220.181.51.116, 220.181.51.119, 220.181.51.120, 220.181.51.122	Brute force attack stopped by firewall, Bad Web bot	China	chinanetcenter.com	23
39.101.184.55, 39.101.205.97, 39.104.235.66	Attack against Apache (too many 404s) , Scanning unused Default website or suspicious access to valid sites from IP marked as abusive, Brute force attack stopped by firewall	China	aliyun.com	54
46.229.173.68, 46.229.168.129 – 46.229.168.163	Brute force activity, Malicious traffic/form submission, Attacks websites by trying to access known vulnerable of plugins, brute-force of backends or probing of administrative tools	Netherlands	datawebglobalgroup.com	3212
103.131.71.180 – 103.131.71.197, 103.131.71.36 – 103.131.71.53	Brute force activity	Vietnam	coccoc.com	386
125.209.235.167 –125.209.235.169, 125.209.235.172 –125.209.235.186	Bad Web bot	South Korea	naver.com	43
178.154.200.11, 178.154.200.25, 178.154.200.55, 178.154.200.154, 178.154.200.176, 178.154.200.188, 178.154.200.197, 178.154.200.211, 178.154.200.239	Brute force attack to crack Website login password	Russia	yandex.net	23
213.180.203.42, 213.180.203.157, 213.180.203.168, 213.180.203.169	Brute force attack	Russia	yandex.net	5
216.244.66.196, 216.244.66.198, 216.244.66.246	Brute force, Bad Web bot, login attempts	USA	wowrack.com	108

Table 4. Examples of suspicious traffic in the dataset.

Originated IP Addresses	Bad activities Reported	Geographical Location of the IP Addresses	DNS name	Number of Identified Suspicious Sessions in EECS Dataset
183.131.110.104	Spam attack	China	chinatelecom.com.cn	110
103.217.176.34, 103.255.5.98	Attempts against non-existent wp-login, SSH brute force attack, Spam attack, XSS Attack	Pakistan	tes.com.pk	4
160.238.74.253, 103.110.169.60	Brute force attack , Unauthorized IMAP connection attempt	India	apsfl.in myhsb.in	5

3.6 Classification Performance

We evaluated the performance of our model for 2-class classification, 'normal/human traffic' vs. 'malicious traffic', on our YorkU-EECS dataset using parameters including accuracy, precision, recall, and f-measure[24]. Given the substantial size of our dataset, we opted for a 50-50 split for training and testing, which proved sufficient for learning the class aspects and preventing over-fitting [94]. The model achieved a detection accuracy of 99.987% with f-measure values as 99.993% and 99.889% (refer to Figure 11 and Table 5) for normal and malicious traffic. Clearly, this result demonstrates a superb classification performance with an exceptional low rate of false positives. Furthermore, as depicted in Figure 11, the overwhelming amount of 'normal' traffic in the training dataset did not dominate the learning task – which is evident from high precision and recall values associate with 'malicious' class.

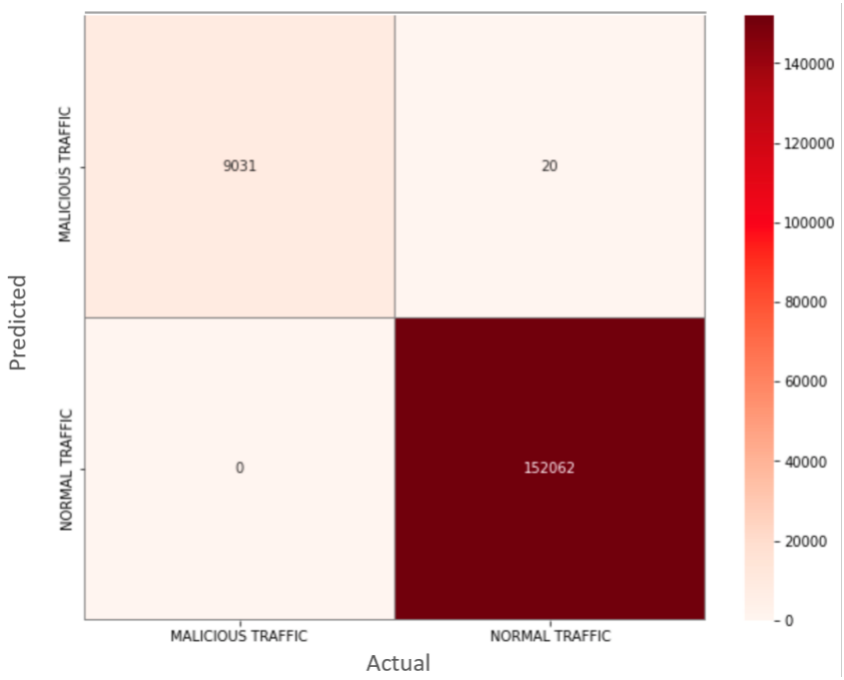


Figure 11. Confusion matrix for 2-class classification.

[24] The classifier is specifically tailored for a two-class classification, distinguishing between normal and malicious traffic. It's important to provide clarity on the handling of the other traffic types. Suspicious traffic, identified as malicious through in-depth analysis in the second step of our examination (refer to Table 4), is appropriately labeled as malicious. On the other hand, instances related to legitimate users with mistyped usernames attempting to log in to specific YorkU servers/services are labeled as normal. As stated earlier, any traffic falling into the unknown category is disregarded. Hence, the classifier is trained exclusively on these two classes.

Table 5. Precision, recall, and f-measure for 2-class classification.

	Precision	Recall	F1-score
Malicious Traffic	1.000000	0.997790	0.998894
Normal Traffic	0.999868	1.000000	0.999934

3.7 Geolocation of Malicious Traffic

In this section, we present our results obtained using Choropleth Map in order to identify the geolocations of different sessions appearing in YorkU-EECS dataset. A Choropleth Map is a map composed of colored polygons, and it shows the spread or impact of certain phenomena across a geographical area [95]. Figure 12 depicts the distribution of the total traffic captured in the YorkU-EECS logs on a world Choropleth map, while Figure 13 depicts the distribution of malicious traffic only. As it could have been expected, given the geolocation of our University and EECS department (Toronto, Canada), the majority of all recorded web sessions have originated from Canada and USA. However, according to Figure 13, the majority of recorded sessions that are identified as 'malicious' have originated from USA, Russia, and China. Such results do not come as a big surprise, as these three countries rank highest in producing malicious cyber activity, according to [96]. Shown in Figure 13, the color bar was set in a way that the yellow color represents the highest value on the map and the dark blue color represents the lowest value. Correspondingly, it can be said that the approximation for the number of malicious sessions generated from China is between 1000 and 2000. This number increases between 2000 and 3000 for the malicious traffic generated from Russia. However, we have the highest number of 6000 (i.e., number of malicious sessions) originating from the USA.

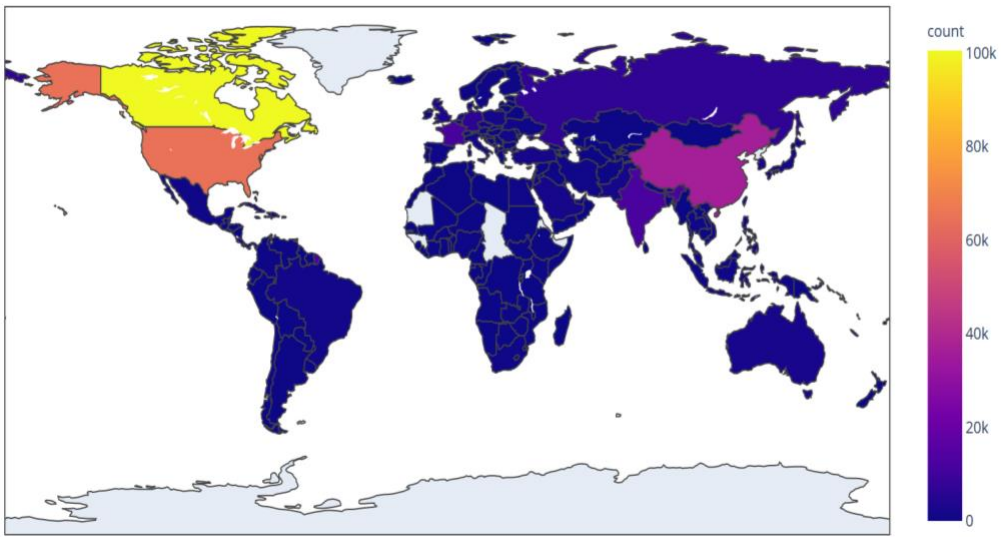


Figure 12. World Choropleth Map, total traffic.

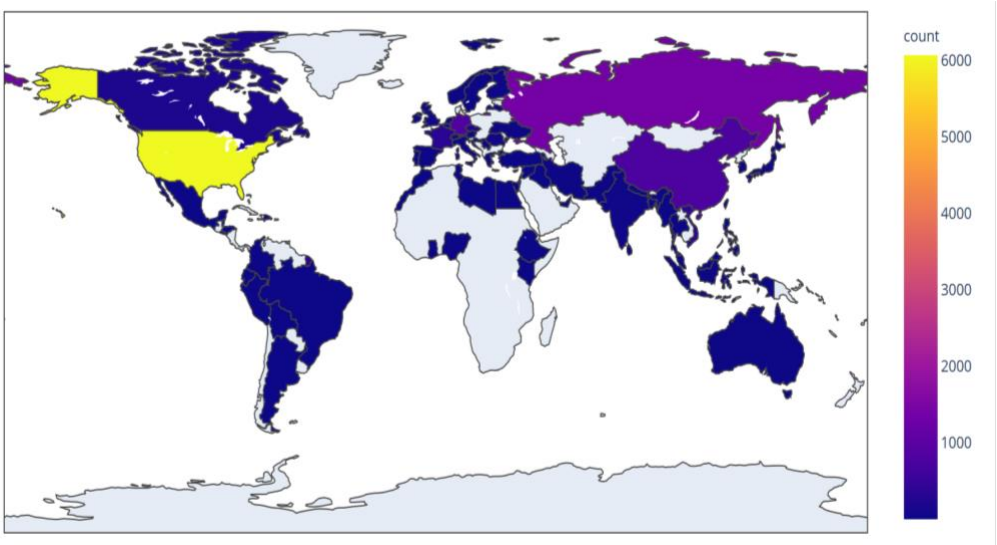


Figure 13. World Choropleth Map, malicious traffic.

3.8 Conclusion

In this chapter, we described our use of unsupervised machine learning algorithm, Self-organizing Map (SOM), to detect malicious web bots. We introduced a novel approach to extract features from content-type parsing, and we applied gradient boosting to rank these features based

on their actual importance. When applied to a real-world dataset, our model effectively distinguished among five categories of traffic (i.e., types of users).

In the upcoming chapter, we will explore the issue of advanced web bots and investigate the use of behavioral biometrics, specifically mouse dynamics, as a non-intrusive security measure. This approach is motivated by the limitations of using server access logs alone to detect advanced malicious bots. Namely, there are many cases where server/access logs may lack crucial information about user behavior, as client-side interactions like mouse movements and clicks are often not captured. To tackle these challenges, behavioral biometrics show promise in enhancing web bot detection capabilities.

Chapter 4

Mouse Dynamics for Advanced Web bot Detection: Extensive Literature Review

Today's web bots are becoming increasingly sophisticated and successful in mimicking human behaviors, making their detection solely based on server access logs particularly challenging. This chapter explores the research on using client-side collected mouse dynamics to detect web bots, as this type of data has proven to be very effective in facilitating successful detection even of the most sophisticated web bots. The chapter provides an in-depth examination of mouse dynamics as a form of behavioral (i.e., biometrics) data, it also offers an overview of detection techniques based on mouse dynamics to-date, and it discusses common threat models corresponding to different levels of web bot evasiveness.

4.1 Introduction

The primary challenge for many businesses is detecting and mitigating bot threats before any actual damage is done (i.e., server crashing, revenue being lost, etc.). Although various methods have been proposed to differentiate bot traffic from human traffic, including methods that utilize mouse dynamics data, there is a notable lack of comprehensive surveys and evaluations of the techniques in this specific domain. The main objective of this chapter is to provide a detailed survey and comparison of the existing studies on mouse dynamics-based biometrics and identify promising future research directions pertaining to the use of mouse dynamics for the purposes of bot detection.

The chapter is structured into five sections. Section 4.2 provides an overview of web bot detection techniques, emphasizing the importance of biometric analysis using mouse movement features. Section 4.3 compares the effectiveness of various mouse movement features in detecting malicious bots. Section 4.4 analyzes popular threat models and evaluation scenarios pertaining to web bot attacks. Section 4.5 summarizes the main contributions of this chapter and highlights some possible future research directions.

4.2 Biometrics Analysis (Mouse Movement) For Purpose of User Authentication and/or Bot Detection

In recent years, researchers have turned to the use of behavior analysis (specifically mouse dynamics) as a potential means of user authentication and/or web bot detection. The fundamental idea behind this approach is to analyze whether the mouse movement data produced by a specific user aligns with patterns that are typical of (other) human users. While web bots can generate individual (i.e., sporadic) mouse events in the same manner as humans, it is challenging for them to perform a wider/longer sequence of mouse operations in an entirely human-like manner. As such, mouse dynamics data can serve as a useful indicator to differentiate bots from human users.

Several studies have explored the use of mouse movement analysis for purposes of bot detection and user authentication. Tables 6 and 7 summarizes the performance of various methods in this general line of research as proposed in the literature.

Table 6. Comparison on existing bot detection proposal methods.

	Ref., Year	Type of Web Bot Detected	Technique Deployed	Key Points and Remarks	Accuracy
Bot Detection	[75], 2017	Bots with statistical attack ability[25]	Deep neural network approach	- A new representation method for mouse movement data is proposed that converts every mouse movement into an image. - CNN models are then used to automate feature learning from mouse movement data.	96.2%
	[50], 2020	Web Scrapers, Form Injecting Bots, Automatic Registration Bots, XRumer, Magic Submitter, AutoIt, BotChief	A combined model of unsupervised and supervised ML techniques including K-Nearest Neighbors algorithm and Naïve Bayes classifier	- The unsupervised model is applied to extract the cluster centers as a representative data point, and the supervised model to classify the unknown data points into either human users or web bots.	92%
	[97], 2020	XRumer, Comment Anywhere, Visual Web Ripper, Integromat, AutoIt, Form-Spamming Playback Bot	Timing Pattern Analysis, Movement Pattern Analysis, Pressure Pattern Analysis, and Error Pattern Analysis.	- Four phases web forensic framework is developed to guide forensic examiners in their expedition to verify if the crime is done using automated bots. - Two unique access loggers are developed to extract the web access patterns such timing, movement, pressure and error patterns, as well as extracting the patterns such as Inter	N/A

[25] In [75], the author explains that a statistical attack operates on the fundamental concept of estimating the probability density functions of features using data from a group of individuals. This information is then leveraged to create forgeries by selecting feature values with the highest probability.

				Request Delay, Entropy of Inter Request Delay and Standard Deviation of Inter Request Delay.	
	[98], 2020	Human Mimic Bot	Support Vector Machine classifier	- Neuromotor features are extracted from human and synthetic mouse trajectories. - A classifier is then trained for bot detection. - The proposed generators can be also helpful for other HCI applications.	93%
	[2], 2021	Moderated Web Bots, Advanced Web Bots	A combined model of web log data (with an ensemble classifier) and mouse movement data (with CNNs)	- The detection model performs a decision-level fusion to take advantage of the complementarity between the two modules based on their different granularity in capturing the distinctive temporal characteristics of the browsing behavior and mouse movements.	100%
	[99], 2017	DDoS attack	A developed classification algorithm	- The detection approach employs web user dynamism by assessing mouse movement and right-click actions. Through the analysis of a dataset comprising 11,055 applications using a Java classification algorithm, it achieves a remarkable 100% accuracy in discriminating between genuine users and DDoS attacks, relying on the identified characteristics.	100%

Table 7. Comparison on existing user authentication proposal techniques.

User Authentication	**Ref., Year**	**Countermeasure/Strategy**	**Key Points and Remarks**	**Accuracy/FP/ FN/ FRR/ FAR/ AUC[26]**
	[100], 2015	The classification algorithm based on distance measures adapted from Kolmogorov-Smirnov non-parametric test	- A user identity recognition system is introduced that can recognize users in small groups (e.g., those who share the same computer) and groups containing hundreds of users.	With 100 users, 85%, a pool of 1500 users, 51%
	[73], 2017	Weighted multi-classifier voting technique – SVM as the basis classifier	- A novel identification approach is presented that uses a weighted multi-classifier voting technique, which combines statistical features and procedural features to accurately characterize users' mouse behavior. - Two experiments are conducted with a dataset from 12 users. The model obtained an accuracy of 84.1% in the identification experiment with an authentication time of 90s.	FRR=5.5% FAR=8.8%

[26] Accuracy (ACC): The ratio of correct predictions to the total number of predictions, providing an overall measure of model correctness, False Positive (FP): Instances wrongly classified as positive by the model when they are actually negative, False Negative (FN): Instances wrongly classified as negative by the model when they are actually positive, False Rejection Rate (FRR): The proportion of actual positive instances that are incorrectly rejected by the model, False Acceptance Rate (FAR): The proportion of actual negative instances that are incorrectly accepted as positive by the model, Area Under the Curve (AUC): The area under the Receiver Operating Characteristic (ROC) curve, representing the model's ability to distinguish between classes. [104].

	[101], 2019	CNN network - deep learning	- A continuous identity authentication method is proposed based on mouse dynamics behavior and deep learning to solve the insider threat attack detection problem. - Human-mouse dynamics behaviors are mapped into pictures to characterize a user's unique mouse behavior characteristics. - A 7-layer CNN network is then used to train the mouse behavior pictures datasets.	FRR=2.94% FAR=2.28%
	[102], 2019	Deep neural network including 1D-CNN, 2D-CNN, LSTM, and a hybrid CNN-LSTM	- Various deep learning architectures for mouse movement sequences classification, including convolutional networks, recurrent networks, and a hybrid model that combines convolutional and recurrent layers, are studied to simplify the feature extraction process. - To train the 2D-CNN model, the mouse movement sequences are plotted as images while the rest of the models receive time-series sequences of features. - Experimental evaluations among various deep learning architectures show that the multi-label 2D-CNN model outperforms other deep learning architectures.	AUC=0.96%
	[103], 2019	Semi-supervised classifier including Domain-based (One-Class Support Vector Machines) and distance-based (k-nearest neighbor and Local Outlier Factor)	- A semi-supervised learning method using a novel feature extraction technique for authentication via mouse dynamics is proposed. - Two new features named Beginning of Action (BoA) and Finishing of Action (FoA) are introduced. - The results confirm that LOF has outperformed all other classifiers. - The experiments show that the proposed model's performance is comparable with the models based on supervised methods.	EER=0.26 AUC=0.78%
	[105], 2020	One-dimensional convolutional neural network	- A novel 1D-CNN model for user authentication based on mouse dynamics is suggested. - The model learns features from raw data and then applies convolutional filters to extract relevant features. -To improve the result, the authors replace the raw coordinates with directional velocities. - Employing transfer learning (learning the data representation on an independent large dataset) can improve the performance of the authentication system as the authors say.	AUC=0.98%
	[74], 2021	One-Class Support Vector Machine (OCSVM)	- A new dataset, SapiMouse, is presented to be used in training and evaluating user authentication and bot detection tasks. - A new user authentication model is also introduced and evaluated on this new dataset. - To learn mouse movement features from raw data, a Fully Convolutional Neural Network (FCN) is applied. - One-Class Support Vector Machine (OCSVM) is then employed for the user authentication task. - The best performance value of the model is 0.94 AUC for 15s of data.	AUC=0.94%

	[106], 2021	Ensemble learning and frequency domain analysis	-A verification study is presented that focuses on behavioral biometrics with mouse dynamics. -The study considers each movement as a signal and enhances the vision of patterns with signal processing techniques. -The characteristics of the mouse usage data in the frequency domain are analyzed to understand the target user's periodic behaviors. -Various techniques of combining consecutive possibilities from action sequences are examined. -The ensemble learning methods are used to learn user behavior against bots.	EER=7.46% AUC=96.47%

4.2.1 Mouse Dynamics for Bot Detection: Related Work[27]

This sub-section provides a brief survey of the works that specifically focus on the use of Mouse Dynamics for the purpose of bot detection. Wei et al. proposed a deep neural network approach for detecting malicious web bots by using mouse movement data in the form of images [75]. Namely, they proposed a new representation method to first convert mouse movements into images, and then they deployed convolutional neural network (CNN) models to extract/learn features automatically. The study employed ResNet [107], a classical CNN model, to determine whether a user is a bot or a human. The experimental results have shown that their proposed system achieved a high accuracy of 96.2% in detecting bots with statistical attack ability, compared to traditional detection methods that use hand-crafted features or recurrent neural networks (RNNs) with less than 30% accuracy in detecting bots.

In their study, Acien et al. [98] aimed to explore the potential of behavioral biometrics to differentiate between human users and bots. They proposed a novel bot detector, BeCAPTCHA-Mouse, that utilizes a neuromotor model of mouse dynamics to generate a new feature set for the classification of human and bot data. The learning framework of the BeCAPTCHA-Mouse model consists of both real and synthetically generated mouse trajectories. The authors introduced two new mouse trajectory synthesis methods: a function-based method based on heuristic functions and a data-driven method based on Generative Adversarial Networks (GANs). The generator uses a Gaussian noise input to synthesize human-like trajectories. The proposed model characterizes each mouse trajectory with a fixed-size feature vector, which is then classified using a Support Vector Machine classifier. The experimental results showed that the proposed features, when used with

[27] Related works on mouse dynamics for user authentication/identification are available in Appendix A.

multiple classifiers and learning scenarios, were effective in detecting bot trajectories with a high effectiveness, achieving 93% accuracy.

Iliou et al. [2] proposed a state-of-the-art approach for detecting malicious bots by combining two detection modules: one that uses web logs and another that leverages mouse movements. The idea is to capture the temporal and spatial properties of both so as to create a more robust detection framework that is difficult to evade. Each module has its own classifier, and the model performs a decision-level fusion to take advantage of their complementarity. The authors evaluated the framework using a test web server with human visitors and simulated malicious bots of different levels of evasiveness. The results showed that combining web logs with visitors' mouse movements is more effective in detecting advanced web bots that try to evade detection.

Rahman et al. [50] proposed a web detection module to identify the presence of human users on web applications. The proposed model uses new biostatistics features, including Input-Source, Click-Pressure, Horizontal Scrolling Amount, Vertical Scrolling Amount, Horizontal Scrolling Speed, Vertical Scrolling Speed, and Entropy of Inter Request Time. The model utilizes two machine learning algorithms: an unsupervised algorithm for extracting cluster centers and a supervised algorithm for classifying unknown data points into human users or web bots. The divisive hierarchical clustering algorithm is used to extract representative data points and transform the dataset into a reduced form, and k-NN and Naïve Bayes classifiers are used to detect unknown and known web bots, respectively. The proposed model achieved a high accuracy of 92% in distinguishing human users from web bots.

Pozzana et al. [108] conducted a study to investigate the behavioral dynamics of bots and humans on Twitter to determine if they differ. They used the Twitter Search API[28] to collect a large Twitter dataset and separated bots and humans before analyzing their activity sessions. The authors measured distinct quantities that captured user behavior and contrasted the results between bots and humans[29]. They found that short-term behavioral trends in humans, which are associated with cognitive origins, were absent in bots due to the automated nature of their activity. The authors developed a set of predictive features[30] that can separate human and bot activity sessions and evaluated the performance of a machine learning framework that leverages these features. The

[28] https://dev.twitter.com/rest/public/search
[29] https://botometer.iuni.iu.edu/
[30] Such as session ID - session the tweet belongs to, the position of the tweet in the session, length of the session, retweet, reply, the numbers of mentions, hashtags, URLs contained in the tweet, and the text length.

results showed that the session features could increase the performance up to 14% in AUC metrics, suggesting that features inspired by cognitive dynamics can be helpful indicators of human activity signatures.

The study described in [99] explores the potential use of mouse movement and right-click behavior in detecting DDoS attacks. The researchers suggest that the dynamism between the user and the system can indicate whether a request to the server is genuine. They propose that click events, as unique events, can be used to unequivocally identify real users, as malicious actors or web bots may make requests without scrolling the mouse or using click events. A classification algorithm is developed in Java and is used to evaluate both mouse movement and right-click behavior, achieving 100% efficiency in distinguishing between real users and robots.

Rahman et al. [97] presented a web forensic framework to help forensic examiners[31] investigate cybercrimes that involve the use of bad bots. The proposed framework consists of four phases: Timing Pattern Analysis, Movement Pattern Analysis, Pressure Pattern Analysis, and Error Pattern Analysis. The authors evaluated the framework by developing a bot crime scenario and presenting detailed forensic procedures and technical reports for bot crime investigation. The results showed that the proposed framework could be effective in identifying and visualizing bot activities on web applications, providing valuable insights for forensic examiners and researchers in developing advanced web bot detection systems.

Overall, as we have seen through our comprehensive literature review, numerous researchers have diligently tackled the challenge of identifying and countering malicious web bots within the realm of the Internet. They have put forth a range of techniques aimed at detection. However, as our survey has revealed, a significant gap persists in the specific and more narrow domain of session-replay bots. These particular bots utilize recording a genuine human user's browsing session by subsequently replaying it. The ultimate objective of session-replay bots is to successfully evade detection by closely mimicking authentic human behavior and eluding suspicion of abnormal activities. To date, session-replay bots remain a greatly understudied research area, and there is almost no published works on the given topic.

[31] As a branch of digital forensics, web application forensics deals with searching and collecting the evidence material found on web servers. It identifies and traces back cyber-attacks on web applications to their originator by analyzing the different server logs, database logs, and browser logs.

4.3 Comparison of Different Mouse Features in Detecting Web Bots

Many of the works surveyed in the previous sections highlight the fact that mouse dynamics can not only be very helpful in verifying the validity of human users or detecting the presence of web bots, but it also yields extremely cost-effective solutions since the mouse is an integral input device of any computer system and does not require the use of additional extra software or hardware components [111]. In this section, we present a comparison of various mouse features utilized in different research studies for detecting web bots.

In general, recordable mouse actions can be divided into four categories: Mouse-Move (MM), Drag and Drop (DD), Point and Click (PC), and Silent (no movement) [103]. The characteristics of 'mouse dynamics' (i.e., Mouse Dynamic Signature [112]) of a specific user can be described as a set of factors/features derived from these four recorded mouse actions, together with their respective timestamps, over a period of time. Examples of 'derived' mouse dynamics features include: mouse movement speed, trajectory, clicking tendencies, drag and drop actions, mouse hovering, and instances of silence. Studies have shown that individual users tend to exhibit unique patterns/characteristics in these features, showcasing diverse mouse movement speeds, trajectories, and clicking frequencies. The mouse dynamics characteristics of individual users are commonly referred to as 'user signature'. Studies have also shown that users' (mouse dynamic) signatures can play a significant role in distinguishing authentic human users from automated bots.

Previous studies have adopted two main approaches for extracting features from mouse movement data in the context of bot detection. One approach involves manually designing hand-crafted features and feeding them into shallow machine learning models. On the other hand, researchers have explored the use of deep learning models, such as convolutional networks and recurrent networks, to automate the feature learning process for mouse movement sequence classification [102], [105]. However, Hu et al. [101] have argued that solely extracting features from basic mouse actions may only capture a limited aspect of user behavior. To address this limitation and preserve the richness of human-generated features, they propose a novel approach where all basic mouse actions are mapped to images. These images are then utilized as inputs to a Convolutional Neural Network (CNN) in the domain of deep learning.

Some of the most widely-used mouse dynamics features, along with short descriptions outlining their key characteristics and the corresponding studies examining their use, are presented

in Tables 8 and 9. The distribution of selected features among different research studies is shown in Figure 14.

Table 8. Description of extracted features from users' mouse movements and click actions.

ID	Feature Names	Short Description
1	Neuromotor features	Distinctive features in human-mouse movements such as input pulse (covered distance), initialization time (displacement in the time axis), log-temporal delay, impulse response time of the neuromotor system, starting angle of the stroke, ending angle of the stroke[32]
2	Cursor trail length/ movement feature of the mouse/distance	Total distance (in pixels) traveled by the cursor on a webpage, the distance between two mouse positions
3	Curser speed/ velocity	The average cursor speed (in pixels per second) as a function of trail length and movement time
4	X and Y coordinate	The coordinates of the cursor on the screen
5	Left-click action	The click action is made up of two actions: pressed and released. This feature detects the pressed and released action of left mouse click
6	Right-click action	The click action is made up of two actions: pressed and released. This feature detects the pressed and release action of right mouse click
7	Double-click	Two consecutive times on the same coordinate (X, Y) becomes a double-click action
8	Drag operations	Drag operations occur when the mouse is not released immediately after being pressed but is instead moved a certain distance before being released. This action, characterized by the initial press, subsequent dragging, and eventual release, constitutes a drag operation
9	Directional features[33]/ angle of movement	The direction of the end-to-end line, direction of movement at a given timestamp
10	Mouse wheel up or down	Continuous mouse wheel down or up
11	Type of movement action-MM, PC, DD	MM describes a mouse movement between two screen locations. PC is a point-and-click (press action and release action), moving the mouse to a point and then clicking one of the mouse buttons. DD is drag and drop action, which starts with a left mouse button pressed event, followed by a sequence of mouse drag events, and ends with a left mouse button released event
12	Acceleration	Change in cursor velocity per unit time
13	Curve feature of mouse movement	The maximum and minimum offset distance from the ideal mouse trajectory and their corresponding positions
14	Mouse action histogram	Statistical Feature - Mouse-move, left/right/double click, point and click/double-click, drag and drop, point-click-point, and mouse-wheel actions histogram
15	Distribution of action elapsed time	Distribution of action (i.e., mouse-move, left/right/double click, point and click/double-click, drag and drop, point-click-point, and mouse wheel) elapsed time[34]
16	Traveled distance correlation	Traveled distance (in a straight line and curve) and ratio of two distances

[32] Stroke as the set of points between two mouse clicks. For more information see [98] and Section 2.1.2.
[33] See Figure 5 in [31].
[34] The elapsed time is the time spent by the user to perform an action; it depends on the traveled distance and the type of the performed action [113].

17	Distribution of movement directions and average movement speed per direction	The directions of mouse movement (end-to-end line direction and direction at a specific timestamp) and calculating the average movement speed for each direction in various actions such as mouse-move, point and click/double-click, and point-click-point
18	Movement elapsed time histogram	The distribution of time taken for mouse-move actions and point and click/double-click actions, it visualizes the frequency of different time intervals for these specific mouse movements
19	Distribution of cursor positions on the screen	Distribution of cursor positions on the screen for mouse-move, left/right/double click, point and click/double-click, drag and drop, point-click-point, and mouse-wheel actions histogram
20	Mouse action duration/elapsed time	The timestamp difference between the last and first mouse-move events
21	Displacement	The segment length between two points, the starting point and the end point, of a mouse movement[35]
22	Displacement angle	The angle of displacement for Point and Click action
23	Move efficiency	The displacement over distance[36] (for Point and Click action)
24	Timing entropy	The sequencing of event intervals associated with the actions of a specific user[37]
25	Scrolling amount in X-direction/ horizontal scrolling amount	The distance covered by mouse cursor while user scrolls the webpage in X-direction
26	Scrolling amount in Y-direction/ vertical scrolling amount	The distance covered by mouse cursor while user scrolls the webpage in Y-direction
27	Scrolling speed in X-direction/ horizontal scrolling speed	The speed of mouse while user scrolls the webpage in X-direction. It can be obtained by dividing the scrolling amount in X-direction by time taken.
28	Scrolling speed in Y-direction/ horizontal scrolling speed	The speed of mouse while user scrolls the webpage in Y-direction. It can be obtained by dividing the scrolling amount in X-direction by time taken.
29	Click pressure	The amount of pressure applied when a user clicks the mouse buttons
30	Input-source	The type of device that generated the event
31	Right-click time of mouse	The time period between the mouse down and the mouse up action on right-click
32	Left-click time of mouse	The time period between the mouse down and the mouse up action on left-click
33	Click duration	Delay between pressing and releasing the mouse button
34	Pause to click	Time between click and last movement event
35	Pause after click	Delay between click and next movement event
36	Stay operation	The interval between two mouse operations

[35] For a mouse movement from the starting point to the end point, displacement is the segment length between the two points, and distance is the actual length traversed [1].
[36] The statement "move efficiency is displacement over distance" means that move efficiency is a measure of how far an object or entity has moved (displacement) relative to the total distance it has traveled. In other words, it assesses how effectively an object reaches its final position compared to the entire path it took to get there. To calculate move efficiency, the displacement is divided by the total distance traveled. Displacement is the straight-line distance from the initial position to the final position, while the total distance is the actual path length taken by the object [1].
[37] Timing entropy refers to the measurement of unpredictability or irregularity in the intervals between events within a user's behavior sequence [1].

37	Scroll operation	The action of vertically moving the visible content of a webpage or interface, by a mouse device, either upwards or downwards.
38	Jerk	Change in mouse acceleration per unit time
39	Angular velocity	The angular velocity of the mouse pointer, i.e., the rate of change of the mouse pointer's orientation or direction over time, it quantifies how quickly the mouse pointer rotates or changes its heading while in motion
40	Minimum, maximum, mean, variance, skewness, kurtosis of a movement action	Minimum, maximum, mean, variance, skewness, kurtosis of a mouse movement action – MM, PC, DD, quantify various statistical aspects of the specific movement, These statistical measures provide valuable information about the distribution, shape, and spread of the mouse movement data, aiding in distinguishing different types of actions
41	Mean frequency, mean power, peak powers, peak power frequencies of a movement action	Mean frequency, mean power, peak powers, peak power frequencies of a mouse movement action – MM, PC, DD, represent different spectral characteristics of the movement, These measures provide insights into the frequency distribution and power distribution of the mouse movement data, allowing for a better understanding and differentiation of various types of actions
42	Straightness trajectory	The measure of how closely a mouse movement follows a straight line between the press and release actions. It is calculated as the ratio of the straightness of the trajectory to the curviness, representing the degree of deviation from a straight line during the mouse movement journey
43	First and second click time	The first click time refers to the time elapsed between the first two clicks, while the second click time is the time difference between the last two clicks
44	First, second, third, and fourth interval time	The first interval time is the time between the first and third actions, the second interval time is the time between the first and fourth actions, the third interval time is the time between the second and third actions, and the fourth interval time is the time between the second and fourth actions
45	Mean, std, min, max of horizontal velocity, vertical velocity, velocity, acceleration, jerk, angular velocity of mouse movement	Mean, standard deviation, minimum, maximum of horizontal velocity, vertical velocity, velocity, acceleration, jerk and angular velocity of a mouse movement action
46	Number of points (events) in a move action	The count of mouse events recorded during that action
47	Sum of the angles	The cumulative changes in trajectory angles during the mouse movement between two specific actions, it quantifies the overall angular deviation of the mouse pointer as it moves from one point to another
48	Largest deviation	The maximum distance between the points along the mouse trajectory and the straight line connecting the two endpoints. It measures the farthest the mouse pointer deviates from a direct path between the starting and ending positions during the movement
49	Number of sharp angles[38]	The count of angles with a measure less than 0.0005° (degrees) in the mouse movement trajectory
50	Movement offset	The distance between the practical mouse trajectory and the ideal mouse trajectory
51	X-speed against distance	The mouse movement speed compared to traveled distance in abscissa direction

[38] See Figure 5 in [31].

52	Y-speed against distance	The mouse movement speed compared to traveled distance in ordinate direction
53	Average speed against distance	Average mouse movement speed compared to cumulatively traveled distance
54	X-acceleration against distance	The mouse movement acceleration in comparison to the traveled distance in the abscissa (horizontal) direction
55	Y-acceleration against distance	The mouse movement acceleration in comparison to the traveled distance in the ordinate (vertical) direction
56	Acceleration against distance	Average mouse movement acceleration compared to accumulatively traveled distance
57	X-acceleration	The mouse movement acceleration in abscissa direction
58	Y-acceleration	The mouse movement acceleration in ordinate direction
59	Beginning of Action (BoA)	The beginning of the movement before the first time point where the acceleration falls below zero
60	Finishing of Action (FoA)	The finishing of the movement after the last time point where the acceleration is below zero
61	Curvature change rate (min, max, mean, SD, variance)	The minimum, maximum, mean, standard deviation, and variance of the curvature change in the mouse movement trajectory. Curvature change is calculated as the change in curvature per pixel traveled, divided by the length of the path from the origin point
62	Angle of curvature	The curvature of the mouse movement trajectory, it provides insight into the direction and degree of curvature exhibited by the mouse pointer's path
63	Jerk along X-axis	The change in mouse acceleration per unit time in X-direction
64	Jerk along Y-axis	The change in mouse acceleration per unit time in Y-direction
65	Jerk over the mouse (x-y) plane	The acceleration of the mouse movement along the X and Y axes, it quantifies how the mouse pointer's acceleration changes over time as it moves across the screen
66	Jerk over the mouse (x-y) plane (min, max, mean, SD, variance)	The minimum, maximum, mean, standard deviation, and variance values for jerk in both the X and Y directions, the acceleration of the mouse pointer changes as it moves across the screen

Table 9. Utilization of different features extracted from users' mouse actions in previous published works actions.

Feature Names	**Literature**																			
	[73]	[74]	[1]	[75]	[99]	[2]	[50]	[97]	[98]	[101]	[106]	[102]	[105]	[103]	[100]	[111]	[31]	[114]	[115]	[116]
Neuromotor features									✓											
Cursor trail length/ movement feature of the mouse/distance	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓		✓	✓	✓	✓	
Curser speed/ velocity	✓	✓	✓								✓	✓	✓	✓			✓	✓	✓	✓
X and Y coordinate		✓		✓		✓						✓						✓	✓	✓

Left-click action	✓		✓							✓				✓					✓	
Right-click action	✓		✓		✓					✓				✓					✓	
Double-click	✓									✓				✓						
Drag operations	✓									✓										
Directional features/angle of movement	✓										✓			✓			✓		✓	
Mouse wheel up or down	✓																			
Type of movement action-MM, PC, DD	✓																✓		✓	
Acceleration	✓										✓			✓			✓		✓	
Curve feature of mouse movement	✓										✓								✓	
Mouse action histogram	✓																			
Distribution of action elapsed time	✓																			
Traveled distance correlation	✓																			
Distribution of movement directions and average movement speed per direction	✓																			
Movement elapsed time histogram	✓																			
Distribution of cursor positions on the screen	✓																			
Mouse action duration/elapsed time			✓								✓					✓	✓		✓	

Displacement			✓																	
Displacement angle			✓																	
Move efficiency			✓																	
Timing entropy			✓																	
Scrolling amount in X-direction/ horizontal scrolling amount							✓	✓		✓										
Scrolling amount in Y-direction/ vertical scrolling amount							✓	✓		✓										
Scrolling speed in X-direction/ horizontal scrolling speed							✓	✓			✓		✓	✓		✓	✓		✓	
Scrolling speed in Y-direction/ horizontal scrolling speed							✓	✓			✓		✓	✓		✓	✓		✓	
Click pressure							✓	✓												
Input-source							✓													
Right-click time of mouse								✓												
Left-click time of mouse								✓												
Click duration															✓					
Pause to click															✓					
Pause after click															✓					
Stay operation										✓										
Scroll operation										✓										
Jerk											✓			✓			✓			
Angular velocity											✓			✓			✓		✓	
Minimum, maximum,											✓			✓						

mean, variance, skewness, kurtosis of a movement action																			
Mean frequency, mean power, peak powers, peak power frequencies of a movement action											✓			✓					
Straightness trajectory											✓			✓			✓		✓
First and second click time											✓			✓					
First, second, third, and fourth interval time											✓			✓					
Mean, std, min, max of horizontal velocity, vertical velocity, velocity, acceleration, jerk, angular velocity of mouse movement																	✓		✓
Number of points (events) in a move action																	✓		
Sum of the angles																	✓		
Largest deviation																	✓		
Sharp angles																	✓		
Movement offset																✓			
X-speed against distance																✓			

Y-speed against distance																✓				
Average speed against distance																✓				
X-acceleration against distance																✓				
Y-acceleration against distance																✓				
Acceleration against distance																✓				
X-acceleration																✓				
Y-acceleration																✓				
Beginning of Action (BoA)														✓			✓			
Finishing of Action (FoA)														✓						
Curvature change rate (min, max, mean, SD, variance)																			✓	
Angle of curvature																			✓	
Jerk along X-axis																			✓	
Jerk along Y-axis																			✓	
Jerk over the mouse (x-y) plane																			✓	
Jerk over the mouse (x-y) plane (min, max, mean, SD, variance)																			✓	

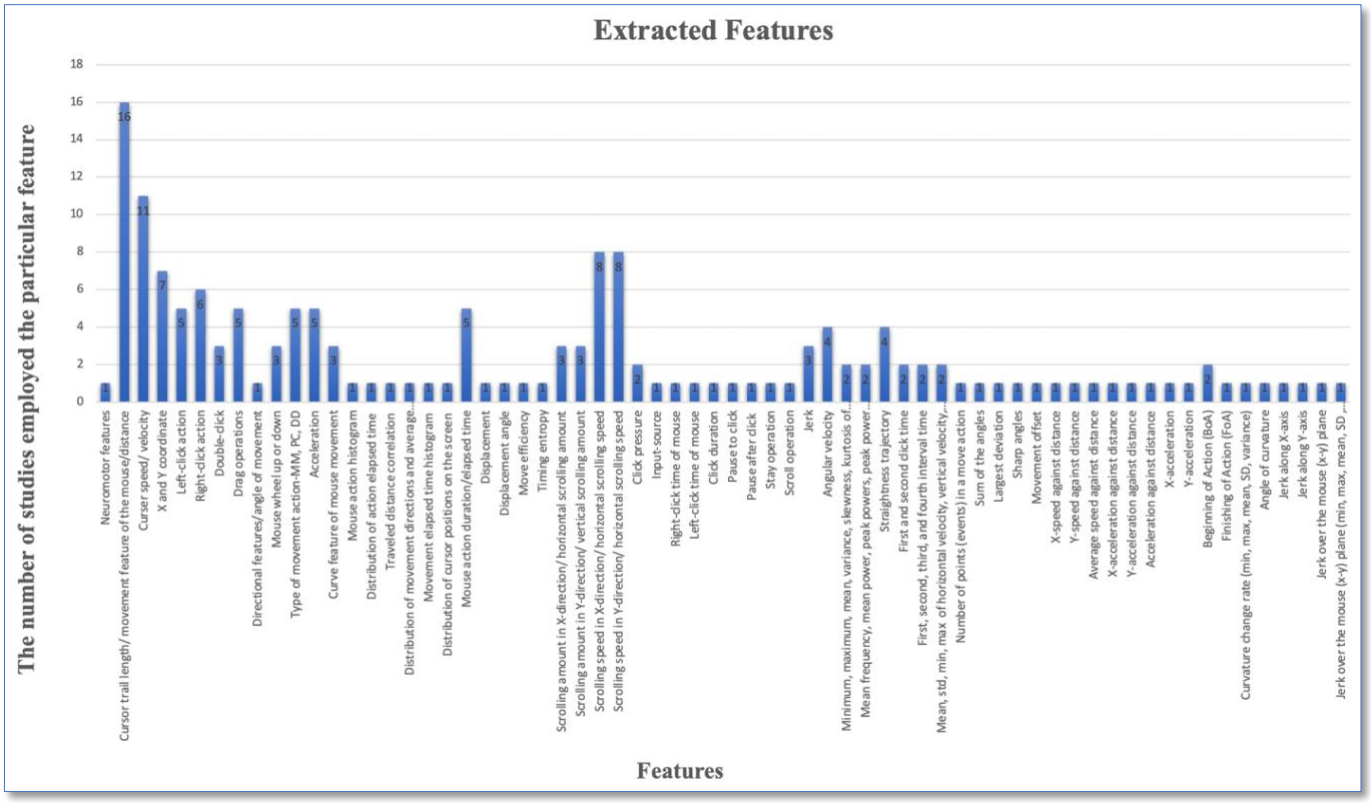


Figure 14. Distribution of detection features among different research studies.

4.4 Web Bot Threat Models Utilizing Mouse Movement

Emulating or generating bot behavior is a critical consideration in web bot detection. Understanding the significance of replicating bot behavior is essential for developing effective strategies to detect and mitigate potential threats. Adversaries can attempt to deceive web bot detectors by sending synthetic traces of mouse movement directly to the server. This can be achieved through various strategies, such as using a generative model of human-mouse dynamics to send a bogus trace or replaying previously recorded human interactions with the target webpage [117]. In addition to various approaches, various tools can also be employed to actually generate web bot traffic, ranging from simple automated browsing environments like *wget* or *curl* to more full-fledged browsers controlled programmatically with libraries like Selenium [39]. The threats posed by botnets are limited only by the creativity of their creators. In this section, we will briefly

review some of the most common approaches to designing and running web bots found in the literature.

4.4.1 Method 1: Software-based Bots

This section describes various software tools that can be used to engineer (i.e., build) malicious web bots.

- Selenium[39] browser automation software: This open-source tool can automate and integrate web browsers into advanced web bots with a benign-looking browser fingerprint and human-like behavior [2].
- XRumer: An automated software tool designed for spamming of online forums and comment sections of web applications. It can register and write posts in forums [97, 98].
- Visual web Ripper: A visual tool used for web harvesting and web scraping. It scans information present on a website, extracts the targeted data, and store this date in XML files, databases, or CSV files [97].
- AutoIt[40]: A Windows-based scripting language that can imitate user actions such as mouse movement, mouse clicks, and keystrokes. It can surf websites using the mouse and keyboard [97].
- Integromat: A tool that can connect web applications and transfer and transform data. It supports different applications using HTTP/SOAP and JSON modules, allowing users to connect to any of them [97].
- Comment Anywhere: An automated program capable of searching relevant pages similar to the user's product description. It saves advertisers' time and money by advertising the content and getting a positive user response [97].
- Form-Spamming Playback Bot: A malicious program used for automatic account registration. It records the actions of human users interacting within a webpage using a mouse and keyboard, and later masquerades as the human user by playing back the recorded traces on the webpage [97].

[39] https://www.seleniumhq.org/. For this research, the authors assumed that the fingerprint generated by Selenium is indistinguishable from a browser fingerprint.
[40] http://www.autoitscript.com/site/autoit/.

- Form Inject Bot: A PHP cURL script that sends an HTTP request to the server hosting the blog page where the bot plans to post comments. The bot then injects content into form fields[41], creates a syntactically legal HTTP response with the HTML form data as the body, and sends it to the submission URL at the server [1].
- Human Mimic Bot: Configured based on AutoHotkey[42] script, an open-source Windows program designed to automate the Windows GUI and general scripting. It opens a blog page in the browser and uses OS API calls to generate keystroke and mouse events. It mimics human browsing behavior, including movement and clicks, scroll, drag and drop, and type keys, fooling older detection methods [1].
- Replay Bot: This bot records a human's actions while filling out a form and then replays those actions on form submission pages to impersonate the human. It utilizes the Global Mouse and Keyboard Library for Windows[43], which enables both recording and replaying capabilities [1]. It should be noted that this Replay Bot falls under the category of blog bots and is distinct from the Replay Bot (ReBot) that will be introduced in Chapter 6, which is a session-replay bot that mimics complete browsing sessions, including mouse movement trajectories and click actions on various websites.
- Magic Submitter: An automated software that submits web content to other websites, forums, etc. It can enhance a webpage's rank in a short time [50].
- BotChief: A software creation tool that automates several online tasks, such as creating massive accounts on websites, web scraping, and web form submission. It can analyze web data, synchronize online accounts, and upload and download website data [50].
- Human-Like Interaction Selenium API (HLISA): A new interaction library for Selenium that provides more human-like interaction and allows Selenium-based bots to hide identifiable behavior [118].

In addition to the above enlisted software tools, other resources that can be used to generate humanlike mouse movement can be found on GitHub, such as Ghost Cursor[44],

[41] For example, <input type="text" name="email" /> is the text field to enter email address. This bot can recognize fields and fill in appropriate content.
[42] http://www.autohotkey.com/. A similar bot tool that may generate simple human behavior is AutoMe. (http://www.asoftech.com/autome/).
[43] http://www.codeproject.com/KB/system/globalmousekeyboardlib.aspx.
[44] https://github.com/Xetera/ghost-cursor.

NaturalMouseMotion[45], BezMouse[46], wind-mouse[47], Humanlike mouse move[48], ClickBot[49], and pyclick[50].

The emergence of automation DevTools such as Puppeteer[51] and Playwright[52] provides adversaries with the capability to run browsers identical to those used by end-users, thereby evading bot detection systems and seamlessly blending in with the vast volume of internet traffic. As a result, they can carry out malicious activities with ease [119, 120].

While some of the mentioned tools excel in simulating human-like mouse behavior, their design caters primarily to specific types of web bots, often limited to a narrow subset of replay bots tailored for tasks on blog webpages or those involving user form submissions. Consequently, their functionality falls short of addressing the comprehensive requirements of our research objectives. This leads us to the conclusion that despite the availability of numerous tools, there remains a noticeable absence of readily deployable (i.e., plug-and-play) bots customized to precisely meet our specific research needs.

4.4.2 Method 2: Knowledge-based Bots

In this section, we delve into the realm of theoretical trajectory synthesis techniques, which are knowledge-based methods used to develop web bots capable of generating human mouse trajectories.

- The probability density function-based bots: This category of bots relies on estimating probability density functions based on a dataset of human actions. The core concept behind these bots is to analyze and predict the most probable feature values, which are then used to generate forgeries. In the design of these bots, particularly, kernel density estimation is applied. It is employed to calculate the probability density estimate for two key aspects of human mouse movement sequences: the step size (the distance between two consecutive

[45] https://github.com/JoonasVali/NaturalMouseMotion.
[46] https://github.com/vincentbavitz/bezmouse.
[47] https://github.com/arevi/wind-mouse.
[48] https://github.com/khabibr/human_like_mouse_move.
[49] https://github.com/amSangi/ClickBot/.
[50] The library can be used to generate mouse trajectories based on BÈzier curves. https://github.com/patrikoss/pyclick.
[51] Puppeteer, introduced by Google, is a powerful interface in NodeJS for automating tests and various tasks using the Chromium browser engine.
[52] Playwright, created by Microsoft, is an automation framework that allows developers to test across browsers, further democratizing the adoption of sophisticated automation tools across the globe.

points in mouse movement) and the event interval (the time difference between two consecutive points in mouse movement). These probability density estimates are crucial in synthesizing bot mouse movement sequences. They ensure that the step size and event interval series generated for the bots closely mimic the distribution found in human actions.

- The shape and velocity function-based bots: These bots generate various mouse trajectories that mimic human movements by changing the parameters of shape functions such as linear, quadratic, and exponential. These functions are combined with different velocity profiles such as constant, logarithmic, and Gaussian to produce human-like mouse trajectories. For further information, refer to [98].
- Impersonation bots utilize two common distributions, including the uniform distribution and the normal distribution, to generate sequences of varying step and time sequences. These distributions are governed by heuristics, which are a set of rules or procedures, that determine the parameters for generating the time and step sequence values (where "step sequence" refers to a series of steps or movements with associated lengths and directions used in simulating mouse movements [121].

It's noteworthy that these bots, while capable of mimicking human-like mouse trajectories to some extent, come with significant limitations: They often lack the complexity, the natural randomness and variation found in genuine human behavior, resulting in repetitive and easily distinguishable actions. Bots relying on heuristic functions face challenges in adapting to different scenarios and user behaviors, thus reducing their flexibility in replicating real-world interactions. Moreover, these web bots encounter difficulties in evading security measures, particularly in scenarios where diverse and unpredictable behavior patterns are essential for bypassing security mechanisms designed to detect and thwart automated bot activities.

4.4.3 Method 3: ML-based Bots

This section introduces two advanced web bots developed using machine learning techniques: deep autoencoders and GAN (Generative Adversarial Network). The objective of these bots is to mimic human behavior as closely as possible, including response times, typing dynamics, phrasing, and mouse trajectories.

- SapiAgent: is a bot designed to generate human-like mouse trajectories by using deep learning. It employs a deep autoencoder and a novel training algorithm to produce more

realistic mouse movements compared to conventional autoencoders and Bézier curves[53]. During the training process, an autoencoder learns two functions: an encoder, which transforms the input data into a latent code, and a decoder, which reconstructs the input from the latent code. SapiAgent is trained to generate realistic trajectories and is then used to create synthetic trajectories that mimic human trajectories [122].

- GAN-based bot: is a type of machine learning-based bot that uses a Generative Adversarial Network (GAN) consisting of two neuronal networks, a Generator and a Discriminator, to generate synthetic mouse trajectories similar to human ones. The Generator generates fake samples (mouse trajectories) that are very similar to the real ones, while the Discriminator has to distinguish between real and fake samples. The topology of both Generator and Discriminator typically consists of two LSTM layers followed by a dense layer, and the dense layer of the Discriminator is used to classify fake and real mouse trajectories [98].

While these bots employ machine learning to adapt dynamically, they demonstrate sophistication in devising novel behaviors and strategies. However, these may not consistently replicate authentic human actions. Their primary objectives often revolve around exploiting system vulnerabilities, executing complex fraudulent activities, and engaging in malicious actions. Despite their proficiency in deception and adaptability, their capacity to faithfully replicate nuanced human-like behavior may be limited in comparison to session-replay bots.

In addition, GAN-generated bots are accompanied by inherent constraints, primarily rooted in the substantial computational costs involved in their training. The resource-intensive nature and high computational demands of GAN training give rise to practical challenges that constrain their applicability in real-world scenarios where considerations of efficiency and cost-effectiveness hold paramount importance. The substantial computational requisites can impede their widespread adoption, necessitating a meticulous evaluation of the trade-offs between capabilities and resource allocation. Therefore, due to the unavailability of a session-replay bot tool and in response to the limitations of the proposed model, we have developed an innovative session-replay bot tool named "ReBot." This Selenium-based session-replay web bot tool is capable of recording and replaying human browsing behavior on the web to mimic human actions, which will be discussed in Chapter 6.

[53] https://www.redblobgames.com/articles/curved-paths/#arcs.

Furthermore, to facilitate a fair comparison between ReBot's replayed sessions and those generated by a generative model, substantial efforts have been invested, which will be described in Chapter 8. A distinctive generative model named ‘TimeGAN’ has been employed for this purpose. TimeGAN was utilized to evaluate the effectiveness of tools like ReBot in executing session-replay bot attacks.

It is imperative to emphasize that the synthetic replay sessions were created using TimeGAN, a generative model specifically designed for time-series data. In contrast to traditional GAN-based models, such as SapiAgent and GAN-based bots in this section, which may encounter challenges in capturing the distribution of features at individual time points and the intricate relationships among features over time in time-series data, TimeGAN demonstrates proficiency in these aspects. Section 9.4 provides a comprehensive summary of our findings.

4.5 Discussion & Conclusion

This chapter has extensively examined the realm of advanced web bots and explored cutting-edge detection techniques centered around behavioral biometrics, particularly mouse dynamics, which offer a seamless and unintrusive layer of security. Our classification of the published research to-date into two main categories - bot detection models and user authentication systems - reveals the potential synergy between authentication technologies and bot detection systems. Through an amalgamation of literature analysis and practical exploration, a roster of established mouse-derived features for behavioral detection of bots (vs. humans) has been identified – out of which neuromotor attributes extracted from complete mouse actions within a session have demonstrated remarkable efficacy in distinguishing human users from web bots. The innovative application of mapping user mouse actions to images, followed by automatic extraction and modeling of these images using deep learning's CNN, has showcased promising results in bot detection. In addition to CNN, other ML approaches that are shown to be effective in detection of web bots include Support Vector Machine, Random Forest, Adaboost, and the Multi-Layer Perceptron.

However, the present techniques of web bot detection based on mouse dynamics exhibit some obvious limitations, predominantly focusing on specific bot types and behaviors, with minimal attention on 'replay bots'.

In the upcoming chapters, our attention will be directed towards the emerging threat posed by advanced malicious bots that mimic human behavior, with a specific emphasis on a particular category of web bots commonly referred to as 'session-replay bots'. Our initial investigation involves a comprehensive analysis of statistical consistencies and variations present within ‘repeat’ browsing sessions of legitimate users. In Chapter 5 we specifically delve into analysis of mouse movement trajectories contributed by a diverse range of authentic human users on a designated target webpage.

Chapter 5

ReMouse Dataset: Analysis of the Novel Mouse Dynamics Dataset with Repeat Sessions

The pros and cons of web bot detection utilizing server-logs are reasonably well known and researched. Yet, there are only a handful of studies that have looked into the use of client-side mouse-base biometrics for the purposes of bot detection. Moreover, the particular problem of session-replay bot detection appears to be completely overlooked by both groups of researchers - those pursuing server-log vs. client-side biometrics-based detection. The research presented in this and subsequent chapters of this thesis aims to close the obvious gap in the existing web bot detection research/literature, and specifically focus on the detection of session-replay bots utilizing mouse-based biometrics data. Chapter 5 summarizes the first stage of this line of our research, in which we have sought to obtain a better understanding of genuine human behavior in application scenarios where repetitive actions (i.e., repeating of the same set of online tasks) is inherently present or required. A better understanding of how humans behave when repeating the same online task(s) is the key precondition for developing systems capable of detecting session-replay bots. To acquire this understanding, we have created an actual online platform/environment through which we have collected real-world user behavior (mouse dynamics) data, which we have named 'ReMouse dataset'. The dataset includes detailed captions of repeat sessions generated by the same human user(s), making it the first of its kind and with broader relevance for future studies on session-replay bots. As part of our research, the ReMouse dataset has been analyzed using statistical and advanced machine learning-based methods, including deep and unsupervised neural learning. The most important findings of our research suggest that: a) two different human users generally do not produce the same or similar-looking sessions when performing the same or similar online task, and b) even the repeat sessions generated by the same human user tend to be sufficiently distinguishable from one another.

5.1 Introduction

Mouse dynamics analysis is a burgeoning area of research in behavioral biometrics that has gained significant attention in recent years. The attractiveness of mouse dynamic analysis stems from the fact that monitoring of mouse movements is a low-cost and non-intrusive way to authenticate and identify users, and can be relatively easily extended and utilized for the purposes of web bot detection. To advance research in this specific application area, we introduce the ReMouse dataset – a dataset collected through our own online interactive platform that has been deployed on MTurk (https://www.mturk.com/), and which we have made publicly available to the wider research community on IEEE DataPort [123]. As one of the most important contributions of this chapter, we not only describe the online platform that has been developed and used to collect the ReMouse dataset, but we also present the results of our statistical and machine learning-based analysis performed on this dataset.

It should be pointed out that previous studies on mouse dynamics have examined the significance of different mouse movement characteristics for the purposes of user identification/authentication (such as hesitation patterns, random and straight movements, etc.). Some of these studies have also investigated the deployment of various machine learning methods in user identification/authentication systems (refer to Appendix A). However, as shown in Chapter 4, a common drawback of these studies is the fact that they have often relied on a limited number of manually selected features from their respective mouse movement datasets. As the second important contribution of this chapter, and to avoid the pitfalls of manual feature extraction processes, we propose to tackle the problem of mouse trajectory classification by using a deep neural network (convolutional neural network) that utilizes all of the raw mouse movement data. That is, instead of handpicking the most important features for a set of mouse movement trajectories, we let the convolution neural network identify these features in an unsupervised manner.

The third important contribution of this chapter is our deployment of mouse movement analysis specifically for the purposes of detection of the latest and most advanced category of web bots – i.e., bots that are capable of impersonating human behavior in terms of mouse movement. As explained in the earlier chapters, this latest and most advanced generation of human-mimicking malicious bots is capable of programmatically replaying a browsing session, including the mouse movement trajectory, that was previously executed (and recorded) by a genuine human visitor on

a given target/victim website [1, 124, 125]. We believe that some of the main aspects of our research, such as the analysis of statistical similarities and differences between browsing sessions (mouse movement trajectories) generated by the same or by a group of different genuine users on the same target webpage, are of critical importance for the creation of more effective techniques for detection of session replay bots.

The chapter is structured as follows: Section 5.2 provides a comparative overview of the existing publicly available mouse dynamics datasets, including our novel ReMouse dataset. In Section 5.3, we introduce our web platform that was used to collect the ReMouse dataset. The results of the statistical analysis of the ReMouse dataset are presented in Section 5.4, while Sections 5.5 and 5.6 summarize our approach and main findings obtained on the ReMouse dataset using advanced ML techniques. Section 5.7 concludes the chapter and provides directions for future work.

5.2 Related Work - Mouse Dynamics Datasets

The existing research studies focusing on the problem of mouse movement analysis generally deploy two different strategies to acquiring human-generated mouse trajectory data - they either use/rely on existing publicly available datasets (e.g., [109, 110, 126, 127, 128, 129]) or they collect their own. When it comes to the actual process of collecting a mouse movement dataset (as described in all previous related works), two main approaches stand out: (1) the authors either create a 'guided environment', where the users are asked to perform a specific (same) task with the mouse, or (2) the authors create a 'non-guided environment', where users are not guided (i.e., instructed) on how to perform a particular task [130].

In the category of currently publicly available mouse movement datasets, among the most commonly studied ones are: Balabit [109], DFL [110], Bogazici [126], SapiMouse [127], the Attentive Cursor dataset [128], and Chao Shen [129]. In the remainder of this section, we provide a brief description of each of these datasets.

5.2.1.1 Balabit Dataset

Published in 2016, the Balabit dataset falls in the category of 'non-guided environment' datasets and includes mouse pointer positioning and timing information for 10 users working over remote desktop clients connected to a remote server. During data collection, users were asked to

perform their regular daily activities. Mouse events were stored in tuples containing the following data items: timestamp, pressed button, mouse state and mouse pointer coordinates. The primary purpose of collecting the Balabit dataset was to learn how the involved users utilize their mouse so as to be able to protect them from unauthorized usage of their accounts. Both training and test data are presented as sessions in the dataset; however, it is worth noting that the test sessions of Balabit dataset are much shorter than the training sessions.

5.2.1.2 Bogazici Mouse Dynamics Dataset

The Bogazici dataset, published in 2021, also falls into the category of 'non-guided environment' datasets and comprises mouse usage behavior patterns of 24 users gathered over a one-month period. The data collection participants were selected from different positions in a software company in order to acquire different patterns of user behavior while interacting with different programs and tools in the office environment. Each user's machine was loaded with a specially designed program that would launch at startup and would collect the user's mouse movements without being tied to a specific task and without preventing the user from performing their regular daily activities. The specific information contained in the dataset include: mouse action type, timestamp, spatial coordinates, button, state and application window name. The dataset was collected for the purpose of training several neural network and deep learning models, which were then deployed to identify/verify the involved users.

5.2.1.3 The Attentive Cursor Dataset

This is a large-scale 'guided environment' dataset of mouse cursor movements during a web search task, and the set was collected in 2020 for the purposes of inferring a user's attention and demographic information. Nearly 3000 participants were recruited from the FIGURE EIGHT (https://www.figure-eight.com) crowdsourcing platform. Using an injected custom JavaScript code, the authors captured the real-world behavior of individuals completing a transactional web search task. The captured information includes the following data items: mouse cursor position, timestamp, event name, XPath of the DOM element related to the event and the DOM element attributes (if any).

5.2.1.4 SapiMouse Dataset

The dataset was collected at Sapientia University in 2020 and also falls into the category of 'guided environment' datasets. It contains mouse dynamics data from 120 subjects (92 males and 28 females between 18 and 53 years of age). Using a JavaScript web application running on the user's computer, mouse movements were sampled by an event-driven sampling technique. The participants were asked to perform four different actions, and each was associated with geometric shapes in a webpage, including right and left clicks and drag and drop actions. In the dataset, two files were associated with each participant, with each file corresponding to one- and three-minute-long sessions, respectively. Individual lines in the two files capture information pertaining to executed/recorded mouse events, such as mouse cursor position, button type, event type (move, drag, press or release), and respective timestamp. The authors have presented their user authentication results obtained on this dataset in [127].

5.2.1.5 Chao Shen Dataset

This 'non-guided environment' dataset was collected in 2017 and consists of mouse dynamics information pertaining to 28 users, with each user completing at least 30 separate data sessions over a two-month period. Each session consisted of about thirty minutes of the respective user's mouse activity. In the dataset, each mouse operation was represented as a tuple of multi-attributes (action type, application type, screen area and window position) and their respective timestamps. The dataset was collected for the purpose of continuous user authentication.

5.2.1.6 DFL Dataset

This dataset was collected in 2018 from 21 participants in a non-guided environment. The participants were asked to install a background service on their computers (which collected their mouse activity data) and perform their daily activities. The dataset contains the following information about the users' mouse activities: timestamp, button (left, right, no-button), state (move, pressed, released, drag) and coordinates. The dataset was used to evaluate a user verification system, as described in [110].

5.2.1.7 ReMouse Dataset

Our novel mouse dynamics dataset (ReMouse), which we are introducing in this chapter and have already made available to the public on IEEE DataPort [123][54], has been collected by means of a web platform developed using the Django REST framework. To collect mouse data from genuine human participants, the platform was deployed on MTurk (for more details, see Section 5.3).

The main differences between our ReMouse dataset and the mouse dynamics datasets previously released by other researchers are as follows: (i) The ReMouse dataset contains the mouse dynamics information of 100 users of mixed nationality, residing in diverse geographical regions, and using different devices (hardware and software components). (ii) The dataset contains dozens of 'repeat sessions' per each user, where 'repeat sessions' are sessions during which the user is asked to complete the same logical task multiple times in a guided online environment (e.g., repetitively play an online game involving the same sequence of steps and intermediate objectives). Through analysis of such 'repeat sessions', it is possible to obtain a better insight into the actual impact of 'repetition' on the user's mouse behavior (e.g., mouse trajectory and speed). According to our knowledge, this is the first dataset of this kind offered to the public. (iii) Each session in the ReMouse dataset is depicted with more granular information relative to the sessions in other datasets. Namely, in addition to the timing and positioning information of the mouse cursor, our dataset also contains mouse movement speed/velocity, the applications' window size (the height and width), as well as the anonymized IP addresses of the participants as user IDs.

Table 10 compares the characteristics of the most commonly studied publicly available dataset with those of our novel ReMouse dataset.

[54] We have obtained official approval from the Office of Research Ethics (ORE) at our institution, York University, which ensures the appropriate and ethical use of human input data in our work.

Table 10. The characteristics of the most prevalent publicly available dataset, including our novel ReMouse dataset.

Name	*Paper Ref.*	*# of Involved User*	*Overall Data Collection Period*	*Period of Observing Each User's Activity*	*Recorded Mouse Action*	*Recorded Session Fields*	*Task Environment*	*Repeat Sessions'*
Balabit	[109]	10	N/A	N/A	Mouse Movement, Point Click, Drag and Drop	Timestamp, coordinates, pressed button, state of the mouse	Non-guided	No
Bogazici	[126]	24	1 month	2550 hours	Mouse Movement, Point Click, Drag and Drop	Timestamp, coordinates, button, state of the mouse, application window name	Non-guided	No
The Attentive Cursor	[128]	3K	N/A	2 hours	Mouse Movement, Point Click	Timestamp, coordinates, event name, XPath of the DOM element that relates to the event, the DOM element attributes (if any)	Guided	No
SapiMouse	[127]	120	N/A	4 minutes of each user's activity	Mouse Movement, Point Click, Drag and Drop	Timestamp, coordinates, button, state of the mouse	Guided	No
Chao Shen	[129]	28	2 months	30 sessions of 30 minutes	Mouse Movement, Point Click, Drag and Drop	Timestamp, action-type, application-type, screen-area, window-position	Non-guided	No
DFL	[110]	21	7 months	Daily users' mouse activities for 7 months	Mouse Movement, Point Click, Drag and Drop	Timestamp, coordinates, button, state of the mouse	Non-guided	No
ReMouse	[123]	100	2 days	10 minutes of each user's activity	Mouse Movement, Point Click, Drag and Drop	User ID, session ID, timestamp, coordinates, button, event type, state of the mouse, speed, screen size	Guided	Yes

5.3 ReMouse Dataset

5.3.1 Web Platform for Data Collection

Our interactive web platform, which was developed for the purpose of mouse dynamics data collection, is hosted on AWS (Windows Server IIS) and is accessible through the following URL: http://human-likebots.com. On the front/user-facing end, the platform simulates a simple 'Catch Me If You Can!' online game (refer to Figure 15). The game's webpage contains a JavaScript code which captures the actual mouse dynamics data (i.e., mouse move, load, click, scroll, … events) as

well as the associated metadata. Specifically, in the time interval during which the user stays on the website and plays the 'Catch Me If You Can!' game, the script preforms a discrete 'event polling' of various event listeners every 30 ms. In addition to recording the mouse-dynamics-related events, the script also captures the timestamps and x–y coordinates of the recorded events, mouse speed, session ID and screen size. The data collected by the script are first buffered and then sent to the back-end server every few seconds (we decided against shorter sampling and transmission intervals to avoid unnecessary data overhead). Using the Django Rest Framework [131], the server-side web application is able to receive and store the recorded event data in a log file (CSV format). The client- and server-side applications do not record any personal information about the users interacting with the human-likebots.com site.

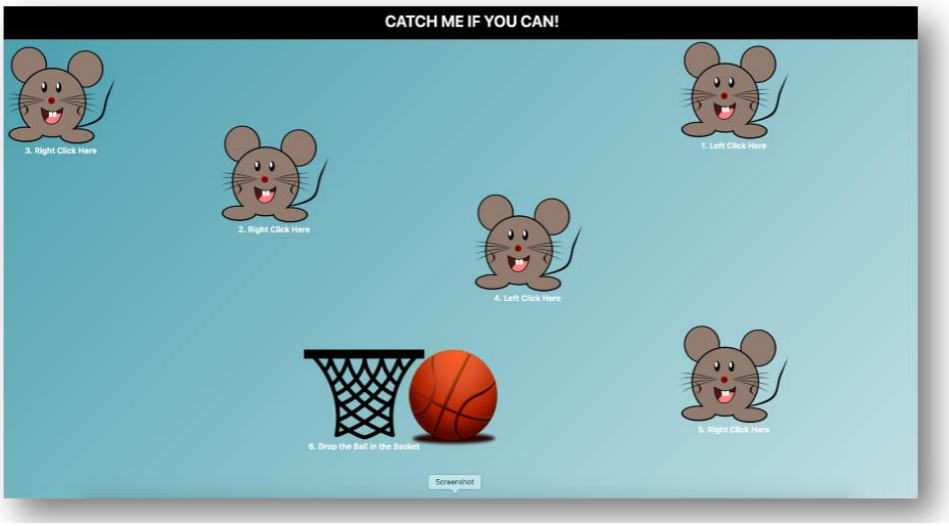


Figure 15.The website 'Catch Me if You Can!'.

5.3.2 ReMouse Dataset Acquisition

In order to collect real human-user data, our interactive human-likebots.com page was deployed on the Amazon MTurk platform. (MTurk is a crowdsourcing marketplace that allows researchers to hire anonymous virtual workers to complete human intelligence tasks for pay. Currently, MTurk offers access to over 500,000 virtual workers from 190 countries[55]). We

[55] Generally, MTurk workers tend to be relatively young, educated, and employed. It has been observed that there is a higher representation of males (57.8%) compared to the greater representation of females reported in existing literature [164]. Additionally, there are some MTurk workers who identify as having a disability. It's noteworthy that, based on the

specifically requested 100 MTurk users to visit and interact with our ‘Catch Me If You Can!’ site by playing multiple rounds of the game—for a total duration of 10 min. In each round of the game, the users were asked to follow six steps and perform three different actions, including left-click, right-click and drag-and-drop actions. We considered each round played by a particular user as a separate mouse movement session. Figure 16 shows the total number of sessions generated by each participating user, while Figure 17 shows the minimum, maximum and average session counts across all 100 involved users.

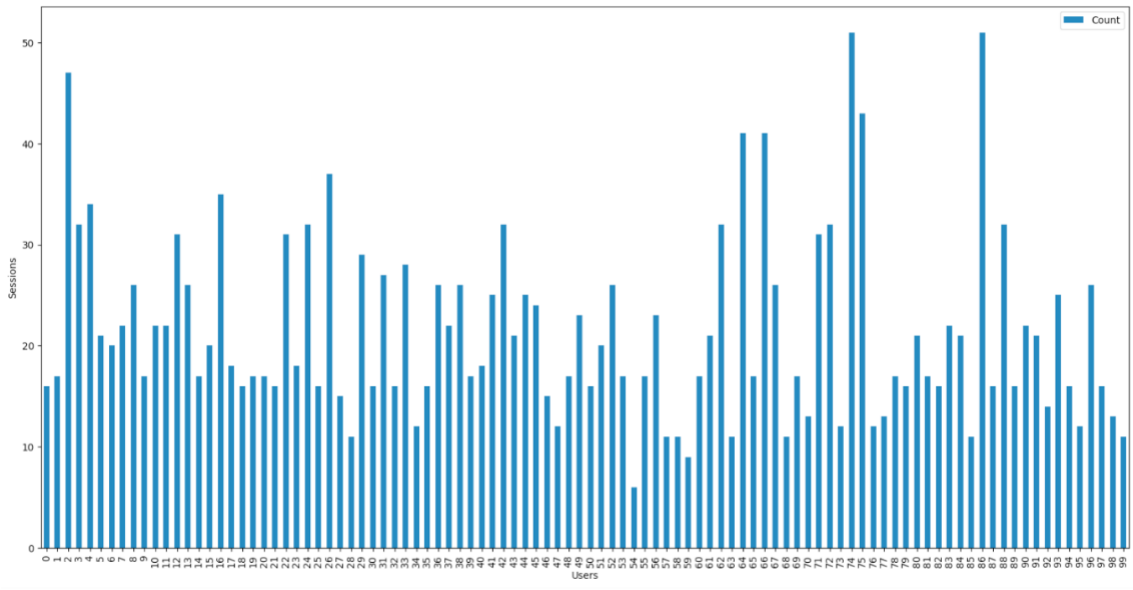


Figure 16.The number of sessions generated by each user.

results of our data collection, none of the identified demographic factors—age, education, employment status, gender distribution, or disability status—seemed to have an effect on our study. [https://dl.acm.org/doi/fullHtml/10.1145/3411764.3445291].

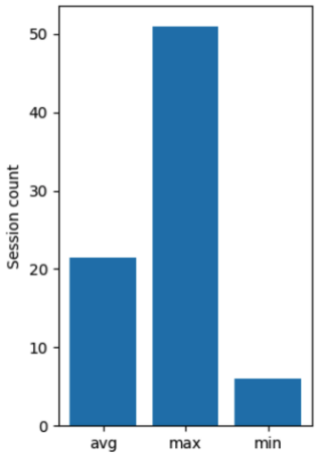


Figure 17. Session status.

5.4 ReMouse Dataset Analysis

5.4.1 Sessions Generated by The Same User

In the first stage of our ReMouse dataset study, we have focused on analyzing the sessions generated by each individual user in isolation from other users. For the purpose of this analysis, a mouse cursor trajectory of a particular session was modeled by means of two time-dependent variables: (1) 2D coordinates/position of the mouse cursor; (2) speed of mouse cursor[56]. As an illustration, Figure 18 displays the trajectories comprising only the mouse coordinates (i.e., positional information) of session number 3 for ReMouse users 90 to 98.

Our analysis of single-user sessions led to some interesting observations:

Observation 1.1: It is evident from the collected data that by repeating the same online task over time (i.e., repeating multiple rounds of our 'Catch Me If You Can!' game), each user generally becomes faster and able to complete every subsequent round of the game in a progressively shorter amount of time. These findings are illustrated in Figure 19, which displays the 'time taken' and the 'average mouse movement speed' for user 82 (which is randomly chosen among the 100

[56] Utilizing mouse speed instead of mouse timestamps when analyzing mouse movement data offers distinct advantages. Firstly, it simplifies the data analysis process, providing a consistent and straightforward measure of movement that facilitates easy comparisons and pattern identification. Secondly, mouse speed reduces sensitivity to user-specific variations, such as pauses or deviations, enabling a clearer focus on underlying movement patterns. Thirdly, by emphasizing movement intensity, it becomes a valuable tool for distinguishing between human users and automated bots, as bots often exhibit unnaturally uniform movement, which can be discerned through variations in speed. These benefits collectively enhance the efficiency, reliability, and accuracy of mouse movement data analysis, making it particularly relevant for applications such as security and user behavior analysis.

participants) across each of the 16 rounds/sessions of the game that this particular user has performed. The same observation is also evident from Figure 20, which shows the dynamic time warping (DTW) distances [132] between the trajectories of subsequent pairs of sessions generated by user 82 (e.g., trajectories of first and second session, second and third session, etc.). As can be seen in Figure 20, the DTW distances between the trajectories of subsequent sessions become closer and shorter as the user keeps repeating the same task.

Note that we opted for the use of the DTW distance metric in our analysis as it has allowed us to measure the distance between two sessions (two time-series) of different lengths and different time-wise alignments (DTW re-aligns two feature vector sequences by warping the time axis iteratively until an optimal match between the two sequences is found [132]). Figure 21 provides a closer look into the trajectories of two particular sessions (number 13 and 14) of user 82 and their respective DTW cumulative distance.

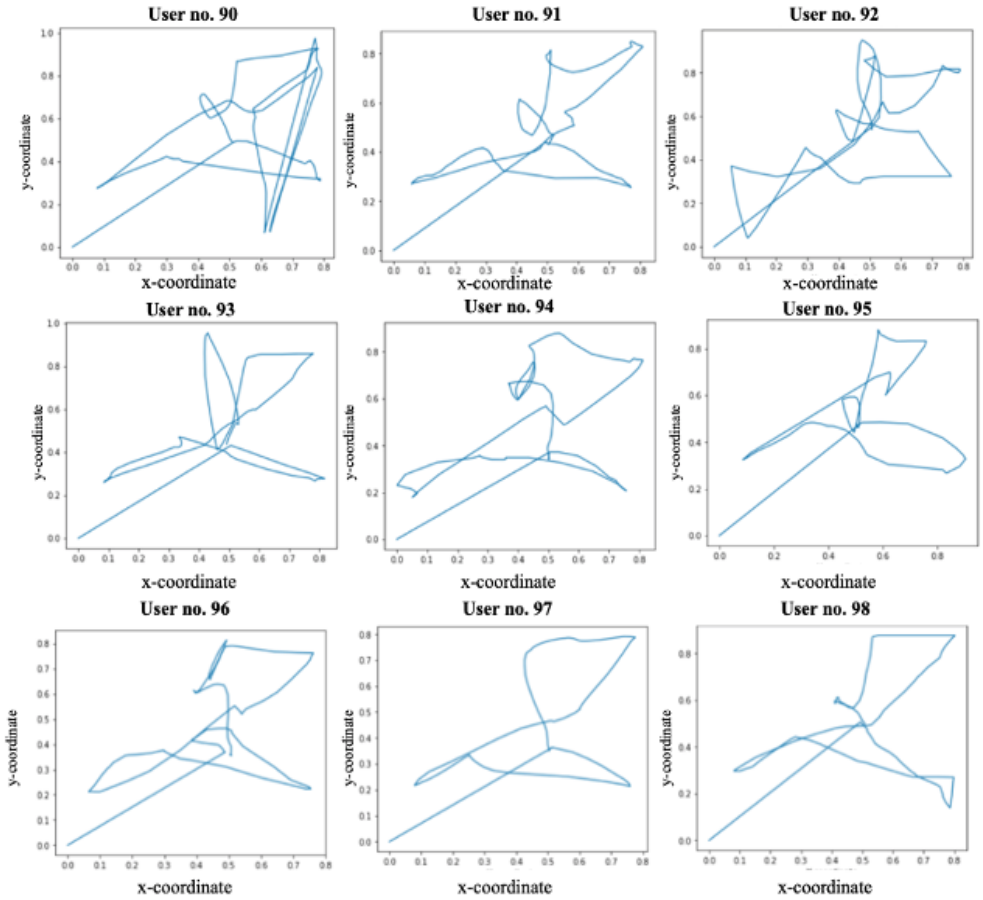


Figure 18. Visual representation of mouse cursor trajectory in the session with order number 3 for users 90 to 98.

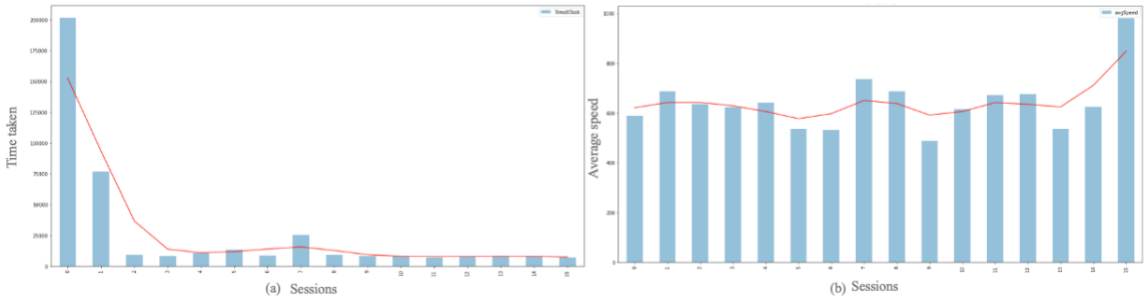


Figure 19. (a) Time taken to complete each of 16 conducted sessions for user number 82; (b) Average mouse movement speed for each of 16 conducted sessions.

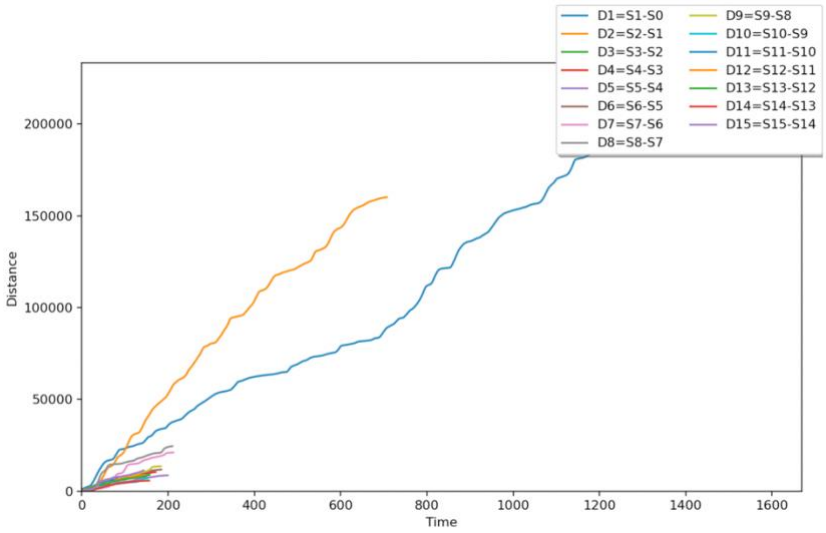


Figure 20. Cumulative difference/distance between subsequent pairs of sessions generated by user 82.

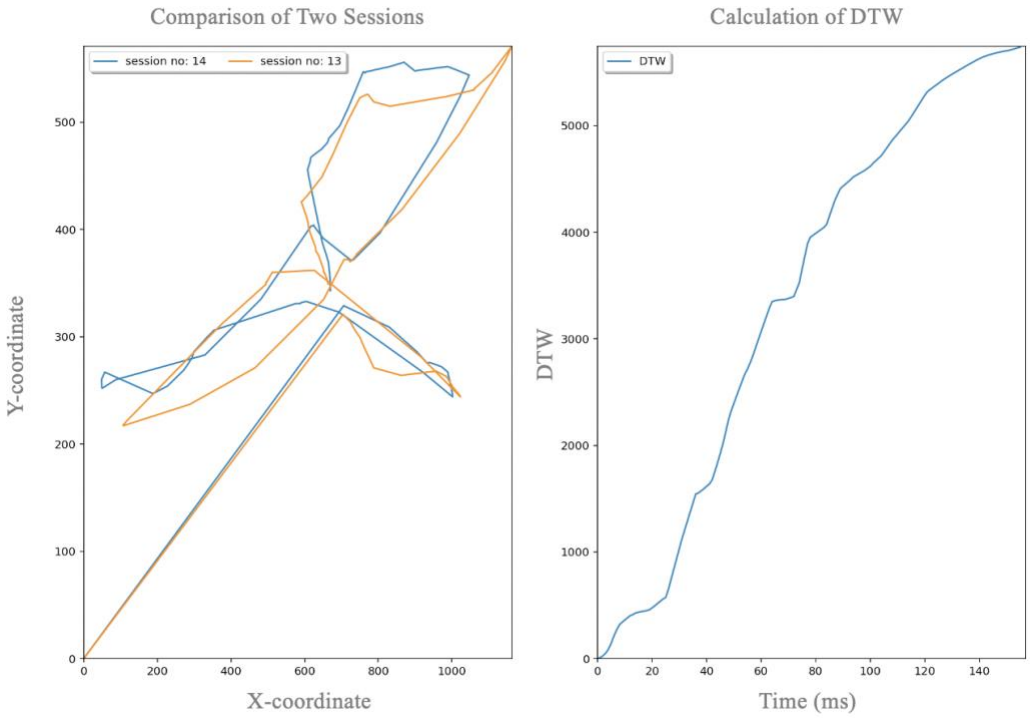


Figure 21. (a) Trajectories of sessions 13 and 14 of user 82; (b) Cumulative DTW distance between two sessions.

To confirm Observation 1.1, we have also deployed simple 'trend line analysis' [133] on the entire ReMouse dataset. A trend line is a bounding line that captures a trend and emerging patterns

in a given dataset. We have employed this analysis to discover the trend in 'time taken to complete a session' and 'average mouse speed' in relation to the session order number for each participating user. The average duration for completing a session was 2254.14 in initial sessions, 417.0 in mid-sessions, and 214 in concluding sessions. This trend indicates that, on average, participants spent less time on the task with each subsequent session or repetition. On the other hand, the average value of the slope in the 'speed of mouse movement' trend lines, when calculated across all users, was 6.5, for mid-sessions it was 10.0, and for concluding sessions, it reached 14.3. This observation further supports the notion that participants became progressively faster in completing similar online tasks with each successive session.

Observation 1.2: Even though the repeat sessions generated by each particular user became progressively 'closer' (as illustrated in Figure 20), no user was able to produce two entirely identical consecutive mouse trajectories when repeating the same online task. This observation is illustrated in Table 11, which shows the ids of the two closest consecutive sessions generated by each respective user in the ReMouse dataset when measured using the minimum normalized cumulative DTW distance. Moreover, since the overall cumulative DTW distances will be greater when the sessions are longer—cumulating over time—we normalized the DTW distance values by the time taken to complete each pair of sessions (i.e., the trajectory time-wise length). That way, the time component does not affect the results, and the minimum DTW distances show the actual trajectories' closeness. A closer inspection of the values in Table 11 reveals that user 74 produced the most similar consecutive trajectories in the ReMouse dataset (corresponding to sessions number 39 and 40), with a normalized cumulative DTW distance of 64.23521268 (note that two identical sessions would produce a DTW distance of 0). The graph shown in Figure 22 plots the minimum normalized cumulative DTW distance values from Table 11, confirming Observation 1.2. Figure 23 provides a closer look at the trajectories of sessions 39 and 40 of user 74, as well as their respective normalized cumulative DTW, for illustration purposes.

Observation 1.3: Through the analysis of ReMouse dataset, we further observed that in the initial sessions the users acted generally more confused, i.e., their cursors exhibited more 'erratic' behavior until the users finally figured out what exactly they were expected to do. However, even in these initial sessions, the mouse speed was not considerably slower than in the later session, which is indicated through a relatively small positive slope value obtained from the 'trend line analysis'.

Table 11. The most similar trajectories generated by each participating user in the ReMouse dataset with their respective DTW values—the minimum DTW normalized cumulative distance between the closest sessions.

Users	*Sessions*	*Min DTW Normalized Cumulative Distance*	*Users*	*Sessions*	*Min DTW Normalized Cumulative Distance*
0	7,8	591.6516	50	2,3	303.9826
1	5,6	295.2985	51	4,5	291.6989
2	35,36	147.0755	52	7,8	272.5094
3	13,14	192.1207	53	13,14	196.9675
4	9,10	180.0245	54	2,3	1490.494
5	4,5	398.1191	55	13,14	421.657
6	8,9	272.4871	56	11,12	276.5871
7	19,20	293.7516	57	8,9	1387.489
8	17,18	192.9701	58	8,9	634.1661
9	11,12	345.1108	59	6,7	777.4243
10	5,6	308.2797	60	6,7	174.8066
11	3,4	572.3161	61	17,18	232.3106
12	2,3	107.556	62	27,28	126.1892
13	21,22	262.7717	63	3,4	1112.61
14	4,5	297.0564	64	33,34	142.0399
15	2,3	287.2074	65	9,10	301.4555
16	9,10	116.766	66	33,34	199.8493
17	10,11	247.4575	67	14,15	137.9862
18	12,13	275.4263	68	3,4	1728.454
19	9,10	371.7259	69	4,5	427.3393
20	7,8	175.7365	70	9,10	1201.285
21	11,12	280.7912	71	17,18	126.8211
22	23,24	127.987	72	16,17	211.9789
23	7,8	343.7548	73	5,6	487.4164
24	28,29	198.9364	**74**	**39,40**	**64.23521**
25	12,13	358.7146	75	24,25	85.11796
26	29,30	204.9529	76	8,9	402.6993
27	11,12	241.8954	77	3,4	623.3006
28	7,8	462.876	78	10,11	412.5679
29	26,27	110.2986	79	11,12	355.0567
30	5,6	210.5634	80	18,19	488.2605
31	11,12	203.5428	81	7,8	315.7737
32	5,6	213.7062	82	13,14	383.0098
33	14,15	258.7817	83	9,10	262.1923
34	8,9	503.8331	84	6,7	275.4376
35	2,3	241.2987	85	8,9	2391.673
36	23,24	210.416	86	48,49	174.3101
37	10,11	305.7957	87	11,12	422.6979
38	23,24	112.3997	88	24,25	113.6169
39	4,5	191.0098	89	7,8	354.2762
40	7,8	429.8543	90	17,18	134.8357
41	17,18	143.9127	91	6,7	299.5449
42	21,22	318.2114	92	5,6	792.4915
43	18,19	226.5839	93	7,8	292.0623
44	4,5	446.748	94	8,9	282.6595
45	6,7	181.1306	95	9,10	432.2253
46	6,7	240.4841	96	23,24	210.416
47	5,6	630.878	97	13,14	261.8753
48	12,13	294.704	98	2,3	753.1881
49	2,3	315.2712	99	8,9	386.572

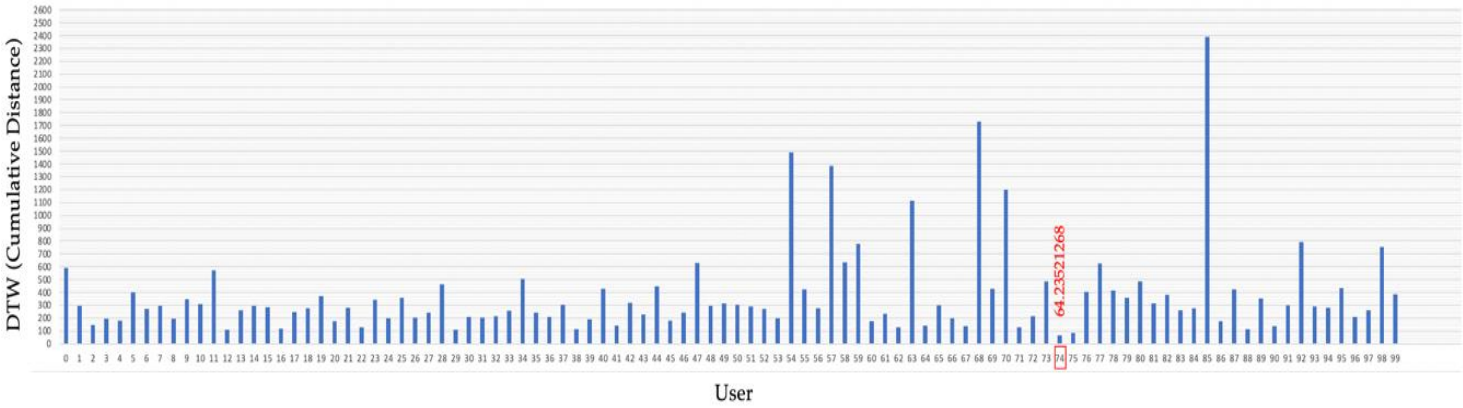


Figure 22. Minimum DTW normalized cumulative distances across sessions of each individual user.

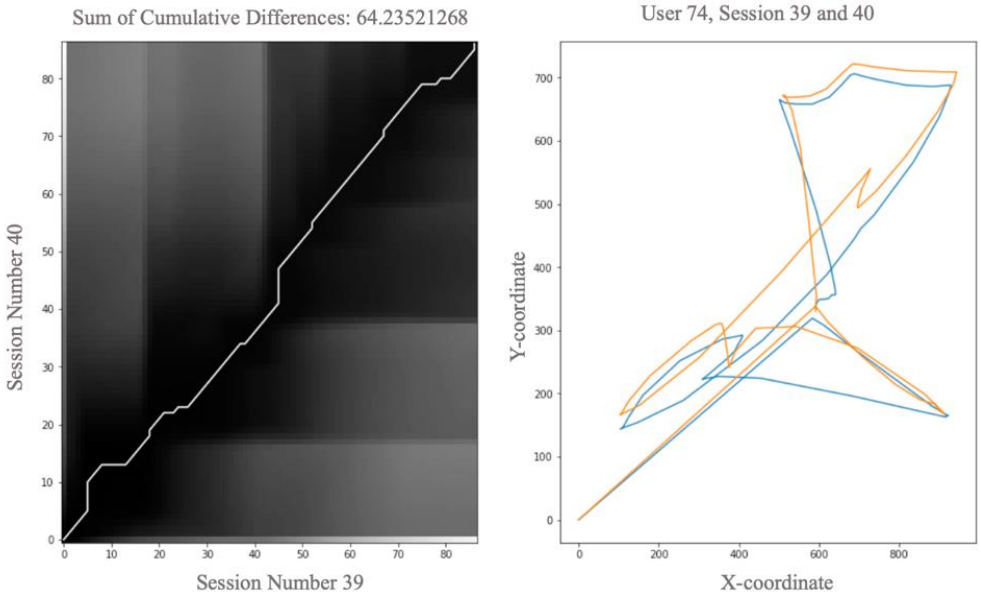


Figure 23. (a) Sum of cumulative DTW distance value in sessions generated by the same user, user 74; (b) Sessions 39 (blue) and 40 (orange) of user 74.

5.4.2 Sessions Generated by Different User

In the second stage of our ReMouse dataset study, the focus was on the pairwise analysis of sessions generated by different users. The findings of this analysis are summarized below:

Observation 2.1: Different users produced different-looking sessions when completing the same/similar online task.

The validity of this observation was confirmed by comparing all users' sessions in our dataset (i.e., by calculating the cross-user pairwise minimum DTW distance). Table 12 shows the minimum

normalized cumulative DTW distance value between two sessions of two distinct users out of all users' sessions. As shown, the most similar trajectories generated by two distinct users are sessions 6 and 29 of users 1 and 2, respectively. The actual DTW distance between these sessions is 21.94, which suggests that, although similar, these two sessions are not identical. This observation can be further generalized, implying that even though sessions generated by two distinct human users while completing the same/similar online task may exhibit a high degree of similarity, they are also likely to be sufficiently distinct from each other.

Observation 2.2: There are no two sessions created by two distinct users that are closer to each other than (any) two sessions created by the same user when completing the same/similar online task.

To confirm this observation, in addition to calculating the distance between sessions generated by different users, we also computed the minimum normalized cumulative DTW distance between ANY two (not just consecutive) sessions generated by the same user in the ReMouse dataset. Table 13 summarizes these results, and it shows that out of the entire ReMouse dataset, user 1 has generated two most similar trajectories (corresponding to sessions number 16 and 28) with a respective distance of 20.376812.

The observations of this section can be further generalized and put in the context of session-replay bots. Namely, the numerical results obtained through the analysis of ReMouse dataset imply that no two sessions (i.e., mouse trajectories) generated on a static website—regardless of whether they are generated by the same or two distinct users—can be identical. Based on this, we further hypothesize that only pre-programmed session-replay bots are theoretically able to produce identical browsing sessions (i.e., mouse trajectories). Or, put another way, any occurrence/observation of 'identical' or 'almost identical' browsing sessions (i.e., mouse trajectories) in a website should be taken with caution, potentially warranting further investigation for the presence of session-replay bots.

Table 12. Cross-user pairwise DTW normalized cumulative distance calculation result.

Min DTW	Users	Sessions
21.941833	1 and 2	6 and 29

Table 13. Pairwise DTW normalized cumulative distance calculation result—the same user.

Min DTW	Users	Sessions
20.376812	1 and 1	16 and 28

5.5 Feature Engineering—Preparing ReMouse Dataset for Machine-Learning-Based Analysis

In previous studies on mouse dynamics, researchers have commonly relied on heuristics-based (i.e., manually selected) mouse movement features, such as 2D cursor position, mouse speed, click frequency, etc. The results of our own ReMouse dataset analysis using manually selected features are presented in Section 5.4. However, some known challenges of manual features selection are: (1) manual feature selection requires in-depth expert knowledge of the specific dataset at hand and the ultimate application environment; (2) there is often a need to fine-tune the number and type of manually selected features for each dataset, which tends to be a time-consuming process; (3) the generalization value of the results obtained using manual feature selection is often questionable. One of the objectives of our work was to analyze the ReMouse dataset by means of advanced machine learning (ML) techniques. However, for the reasons outlined above, we were hoping to avoid basing our ML analysis on manually selected features. Additionally, due to the different durations of individual user sessions in the ReMouse dataset, we were facing very heterogeneous 'mouse location' and 'mouse speed' feature vector representations (i.e., the feature vectors representing different sessions were of variable/non-fixed length). Training an ML algorithm using such non-uniform set of feature vectors would have required additional expert-knowledge decision making and the manual re-engineering of input data.

As an alternative to manual feature selection and feature vector re-engineering, and inspired by works [73, 134], we pursued a novel approach to representing individual user sessions in the ReMouse dataset. Namely, in this part of our analysis, rather than manually extracting features to describe a user's unique mouse behavior characteristics, we mapped the mouse trajectories into images. In order to conduct automated feature extraction on image representations of user sessions from the ReMouse dataset, we further deployed a pre-trained deep learning model—VGG16 [135]. In particular, we used the VGG16 library implemented in Keras [136]. VGG16 is a convolutional neural network model well known for its ability to perform very-high-accuracy feature extraction

on image datasets [137]. The reason why we resorted to deploying a pre-trained VGG16 model is the fact that working with a 'from-scratch' convolutional neural network would require days of training and millions of images to achieve a high accuracy in real-world applications [138] (from the perspective of image processing, our ReMouse dataset is of relatively small size, containing the sessions of 'only' 100 users). For the purposes of our research, we acquired the generic pre-trained VGG16 model from [136] and retrained it on our own image representations of web sessions from the ReMouse dataset (the process of re-using the weights from a pre-trained model is called 'Transfer Learning' [139]). The original VGG16 model used in our work was trained on standard computer vision benchmark datasets, including ImageNet [140].

Using VGG16, we ended up with each image (i.e., user session) being represented as a vector with 1000 features [141]. To further reduce the number of features identified with VGG16, next, we used principal component analysis (PCA) [142]. PCA produced 100 eigenvectors over the VGG16 feature space. Nevertheless, as shown in Figure 24, not all of the 100 identified PCA eigenvectors were of the same significance, as 95% of data variance occurs over the first 57 eigenvectors. Thus, for the purpose of our ML-based analysis (as discussed in the next section) we opted to map our original ReMouse dataset into a set of feature vectors over the first 57 most significant PCA eigenvectors.

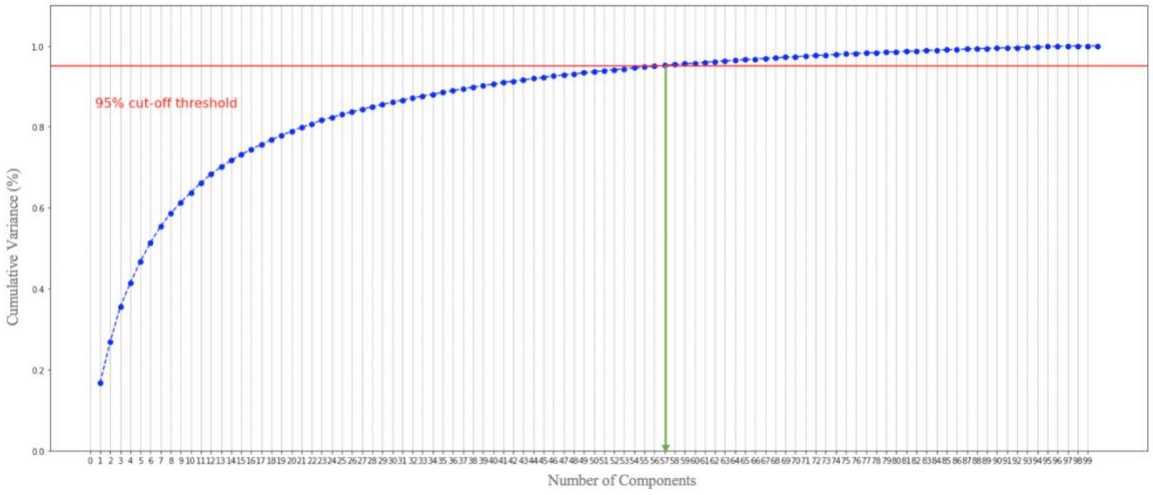


Figure 24.The number of components needed to explain the variance.

5.6 ML-Based Analysis of ReMouse Dataset in Image Representation: Focusing on Sessions Generated by Different Users

The objective of our ML-based analysis of the curated image-based ReMouse dataset (as explained in Section 5.5) was to investigate the (dis)similarities between comparable (same-order number) web sessions generated by different users. We specifically decided to look at the third session generated by each of the 100 participating ReMouse users (forming one data subset, which we will refer to as 'ReMouse Subset-3′ in the reminder of this thesis), as well as the fifth session generated by each of the 100 participating ReMouse users (forming the second data subset, which we will refer to as 'ReMouse Subset-5'). We opted to look at the third and fifth sessions due to our observation that for most ReMouse users some of the originally exhibited 'erratic' mouse behavior largely disappears after the first two rounds/repetitions (i.e., sessions) of the 'Catch Me If You Can!' game (see Section 5.3). In other words, the user behavior and mouse trajectory in the 3rd and 5th sessions are generally 'stable' and thus likely to produce more accurate results. To conduct the cross-user session (dis)similarity analysis, we specifically decided to deploy unsupervised ML learning, including the Self-Organizing Map (SOM) and several unsupervised clustering ML algorithms.

As explained in Chapter 3, the SOM algorithm is typically used to build a topology-preserving mapping of high-dimensional input data to 2D or 3D space, where the similarity of individual input points can be assessed in more intuitive (visual and non-visual) ways. Unsupervised clustering is known for its ability to decompose a dataset into subgroups based on their similarity so that data points in the same cluster are more closely related to each other than data points in different clusters [143].

According to our knowledge, this is the first research study that has looked into the use of unsupervised clustering on the image representation of user sessions for the purpose of cross-user session (dis)similarity analysis. Additionally, the only other work that has pursued image-based web-session representation and analysis [1] was specifically concerned with the problem of malicious web bot detection through session classification, and thus ultimately opted for the use of supervised deep learning—as opposed to tackling the question of session similarity, which has been the focus of our work and thus required the use of unsupervised techniques.

5.6.1 Data Analysis Using SOM Map

For the purposes of our research, we trained two 15-by-15-sized SOM maps (experimentally), one using the ReMouse Subset-3 and the other using ReMouse Subset-5. We used the SOM implementation from the Python SOMPY package [88], which has a structure similar to *somtoolbox* in MATLAB. In terms of functionalities, the package uses only batch training (which is faster than online training) and *sklearn* or random initialization.

The heatmaps generated on each of the two trained SOM maps are shown in Figures 25.a) and 25.b), respectively. An SOM heatmap is produced by displaying how many of the training inputs are associated with each node in the trained SOM map [144]. It is very evident from the two heatmaps that there are no actual (i.e., distinguishable) clusters in either ReMouse Subset-3 or ReMouse Subset-5—as most neurons are 'fired' by no/one single-input point, and only a handful of neurons are fired by two or more (distinct) input points. It should also be noted that the neurons with an input-data membership of two or more are largely distributed at the edges of the respective SOM maps, which suggests that the actual 'closeness' of the input points that fire these neurons may not be significant. Border neurons in an SOM map do not 'stretch out' during the training process as much as they should, and as a result they tend to 'attract' many potentially very different/distant points located on the 'outside' of the SOM border. This phenomenon in known in the literature as the 'SOM border effect' [145].

From a practical point of view, the fact that data points from ReMouse Subset-3 and ReMouse Subset-5 produce such disperse distribution (as shown in Figures 25.a) and 25.b)), is a clear indication that individual users—when performing the same general online task—are likely

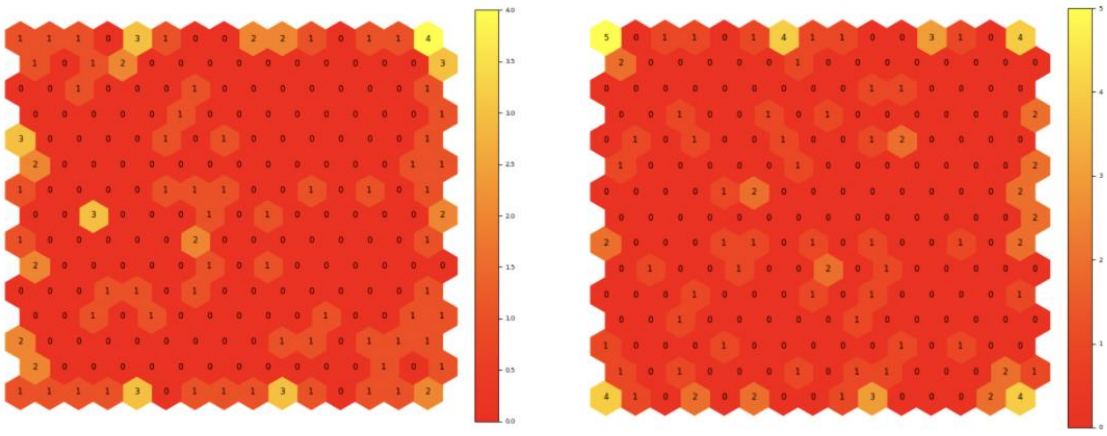


Figure 25. Users' data points map: (a) session number 3; (b) session number 5.

to end up producing substantially different/distinct mouse trajectories. When put in the context of session-replay bots, and as noted in Section 5.4, this further suggests that any session/trajectory that shows a significant similarity with an already-observed session/trajectory should be flagged as potentially ‘malicious’, since (according to our results) the likelihood that both of such sessions are genuinely human is rather small.

The following section will showcase the outcomes of implementing Spherical SOM on the ReMouse dataset to address further the issue of the 'border effect' identified in our dataset.

5.6.2 Spherical SOM to Tackle the Problem of Border Effect in 2D SOM

The common 2D self-organizing map (SOM) topologies, such as circle, square, and rectangle, suffer from "boundary effects," where neurons located on the borders are more heavily influenced by their neighbors and show less variation than their central counterparts [146]. To address this issue, various solutions have been proposed, including the heuristic weighting rule method [147] and local-linear smoothing [148]. Another approach, suggested in [148 - 151], involves using a Spherical SOM, which eliminates grid boundaries by applying the SOM on a sphere.

It has been shown that Spherical SOMs are more visually appealing and intuitive than other methods [152], as they generate maps that are similar to world maps, which people are more accustomed to, rather than maps based on a torus [153]. The idea behind the Spherical SOM is to use a spherical topology that does not have any actual boundaries. In a Spherical SOM, no neuron is located on an edge, so every neuron has a complete neighborhood of other neurons, and no neuron is subjected to higher concentration effects due to its location relative to other neurons.

To address the encountered border effect within the 2D square-shaped SOM maps of the ReMouse dataset, the adoption of the Spherical SOM was deemed necessary. Prior to implementing the Spherical SOM, a deliberate choice was made to employ t-SNE instead of PCA to facilitate dimensionality reduction of the features identified through VGG16. t-SNE, short for t-distributed Stochastic Neighbor Embedding [154], constitutes an unsupervised non-linear dimensionality reduction technique tailored for data exploration and high-dimensional data visualization. This method is particularly adept at delineating patterns within data that cannot be linearly separated. t-SNE not only provides the means to visualize complex datasets within two or three dimensions but also grants an intuitive comprehension of data arrangement within higher dimensions, unraveling

latent relationships and patterns. Distinct from PCA, t-SNE addresses dimensions characterized by non-linear associations by minimizing the divergence between two distributions: the first gauging pairwise similarities among input objects, and the second quantifying pairwise similarities among corresponding low-dimensional points within the embedding [155]. The preference for t-SNE over PCA in our analytical approach was founded on fundamental distinctions between the two techniques, driven by the imperative to surmount the "border effect" challenge inherent in SOM.

This choice was substantiated by two core reasons. Firstly, PCA, with its emphasis on maximizing variance and upholding large pairwise distances, contrasts t-SNE, which specifically preserves small pairwise distances and accentuates local similarities. These properties render t-SNE especially suited for capturing intricate local patterns and conserving the intrinsic structure of data, notably in instances encompassing non-linear manifold structures. (Note, given that we were ultimately focused on discovering the potential presence of clusters in ReMouse dataset, our resorting to t-SNE as the dimensionality reduction method of choice seemed well justified.) Secondly, by allowing us to specifically/explicitly reduce the number of dataset dimensions to 2 or 3, t-SNE also allowed us to visualize all of the 'raw' ReMouse data – even before applying SOM algorithm. The exploration of the 'raw' data is an alternative way to corroborate the primary findings of similarity and disparity within human mouse movement behavior on the web.

For our ReMouse dataset experiment, we have utilized the Scikit-Learn implementation of the t-SNE algorithm [156] and mapped our original dataset into three t-SNE components. The main parameters that can be tuned in t-SNE algorithm include perplexity, learning rate, number of iterations, initialization, and metric. Perplexity controls the balance between preserving the global structure of the data and revealing the local structure of the data, while the learning rate controls the step size at each iteration of the optimization process. The number of iterations determines how many times the algorithm updates the positions of the data points in the lower-dimensional space. The initialization method determines how the data points are initially placed in the lower-dimensional space, and the metric specifies the distance metric used to compute pairwise similarities between data points in the high-dimensional space. The optimal parameter values may depend on the specific dataset and the goals of the analysis. We have set the perplexity value to 50,

which is typically set between 5 and 50[57] [154], the Learning rate to 200, the number of iterations to 1200 with random initialization and Manhattan as the distance metric.

We have then used the extracted features to train 2D square-shaped SOM as well as Spherical SOM to investigate the similarities and dissimilarities between web sessions generated by different users in the ReMouse dataset. (Note that, in order to ensure a fair comparison, here we have repeated the experimentation/training of 2D square-shaped SOMs with ReMouse data that has undergone t-SNE based dimensionality reduction. The original 2D square-shaped SOMs from Section 5.6.1 were trained on ReMouse data that has undergone PCA based dimensionality reduction.)

It is important to note that we have also utilized another specification of t-SNE, specifically geared towards data exploration and visualization of high-dimensional data, to gain insight into how the ReMouse data is distributed in a high-dimensional space. The visualization in Figure 26 depicts (a) session number 3 and (b) session number 5 for all users in the ReMouse dataset using t-SNE. As shown, the results indicate that there are no discernible clusters among either session number 3 or session number 5 produced by the 100 users in the ReMouse dataset. This finding implies that while sessions generated by actual human users may be relatively similar to each other, no clear clustering exists across all the sessions in the ReMouse dataset.

[57] Several experiments have been done to reach the value of 50 for perplexity. Furthermore, as discussed in [https://distill.pub/2016/misread-tsne/], for t-SNE to operate properly, the perplexity should be smaller than the number of data points; so, considering the ratio of perplexity to the number of our data points, 50 was found to be the correct value.

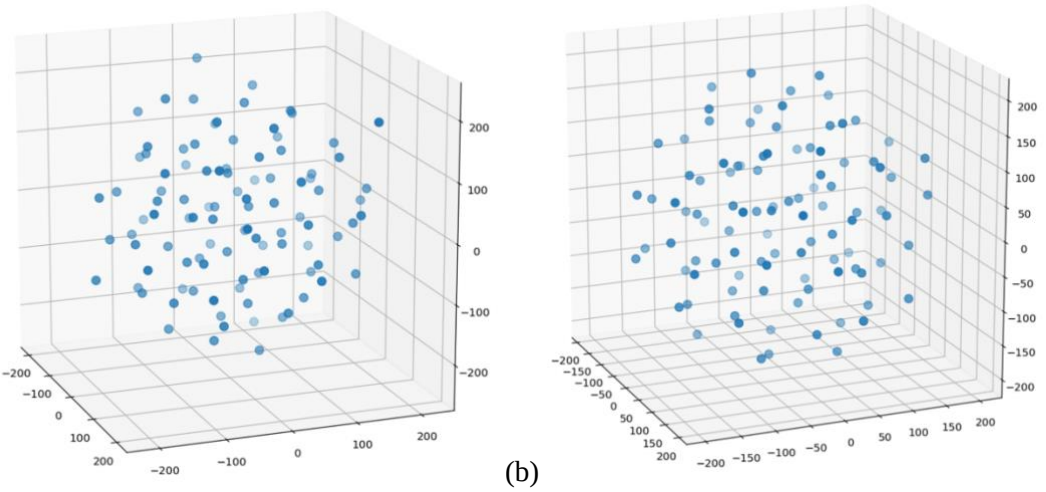


Figure 26. Visualization of 100 users' mouse movement trajectories using t-SNE, (a) session number 3 (b) and session number 5.

5.6.2.1 Data Analysis Using Spherical SOM

To conduct experiments with a spherical SOM, we have utilized *somsphere* [157], which is a Python implementation of SOM in spherical coordinates. It is important to note that in the case of a spherical SOM, the neuron positions in the topology are determined by longitude and co-latitude values, specifically $p_i = (\theta_i, \phi_i)$ where $\theta_i \in [0; 2\pi]$ and $\phi_i \in [0; \pi]$ [158]. The longitudes θ_i and co-latitudes ϕ_i are set by the Healpy package to create a fixed number of equal-surface cells over a sphere, determined by the *n_side* parameter [159]. The default learning rate values were selected to ensure that the spherical SOM weights were highly sensitive to the gradient during the initial learning steps, and less sensitive at later steps, up to the point of convergence. Several operational tests were performed, and it was found that the SOM converged on various randomly generated data points after 50 iterations.

Figures 27.a) and 27.b) show the results obtained using the Spherical SOM algorithm on ReMouse Subset-3 and ReMouse Subset-5. These figures provide clear evidence that the input data is distributed widely throughout the input space, as seen by the data points scattered around the sphere. With Spherical SOM, we have effectively resolved the "border effect" problem encountered with 2D SOM when working with our ReMouse dataset. This allows us to properly observe the behavior of ReMouse participants during different sessions, confirming our earlier hypothesis that session trajectories generated by different users while completing the same online task are sufficiently distinguishable from each other, as demonstrated in the figures.

Figures 28.a) and 28.b) illustrate a comparison of the visualization of the ReMouse dataset undergoing t-SNE based dimensionality reduction using 2D square-shaped SOM and Spherical SOM. The results indicate that Spherical SOM is effective in addressing the "border effect" problem, while 2D square-shaped SOM still suffers from this problem in spite of the deployment of a different dimensionality reduction scheme. These findings highlight the usability of Spherical SOM in analyzing complex data and provide new insights into the behavior of ReMouse participants during different sessions. Figure 29 depicts the flowchart for the analysis of the ReMouse dataset using 2D SOM and Spherical SOM.

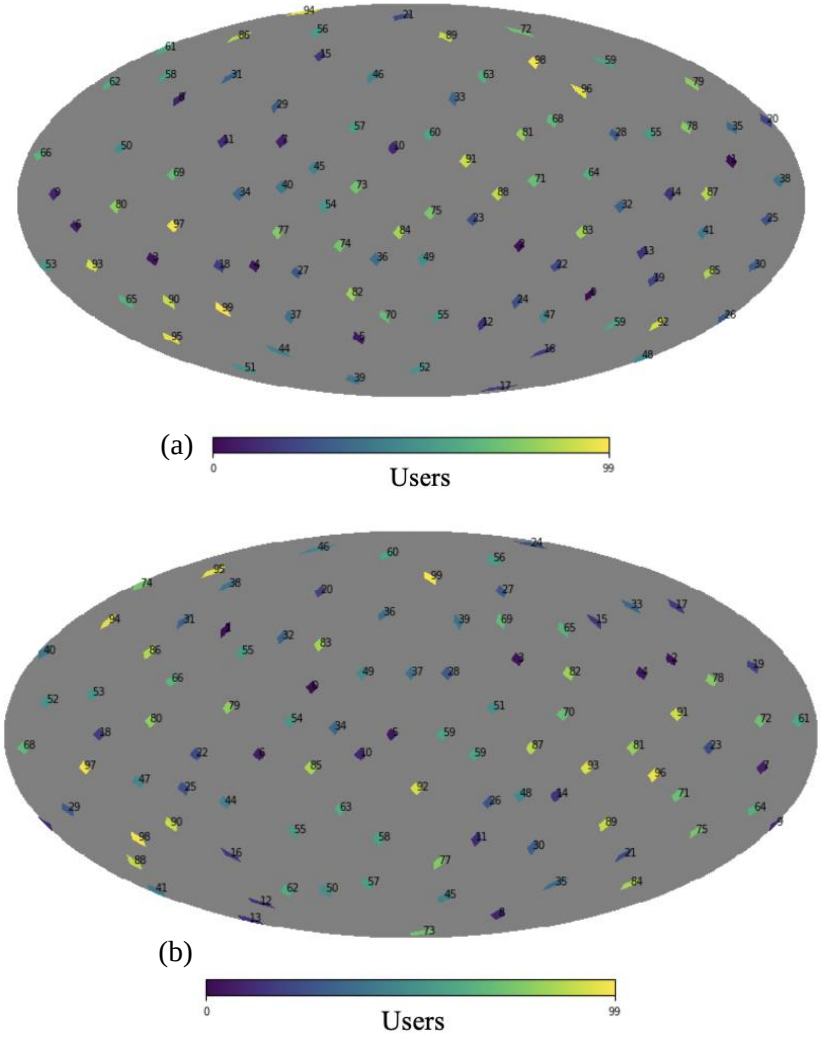


Figure 27. Users' data points sphere SOM map, session number 3 (a) and session number 5 (b).

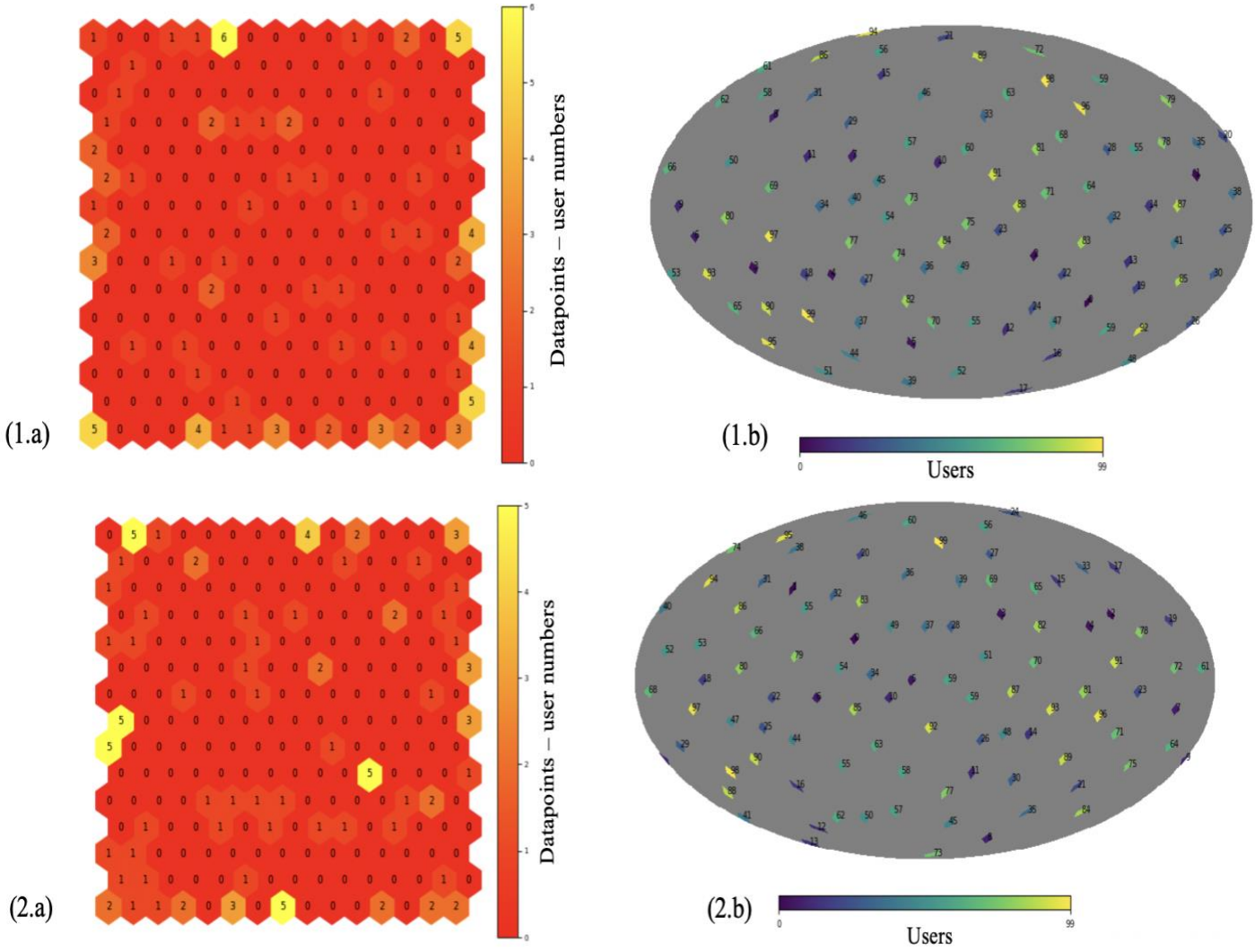


Figure 28. Users' data points map, session number 3, (1.a) SOM (1.b), Spherical SOM and users' data points map, session number 5, (2.a) SOM (2.b), Spherical SOM.

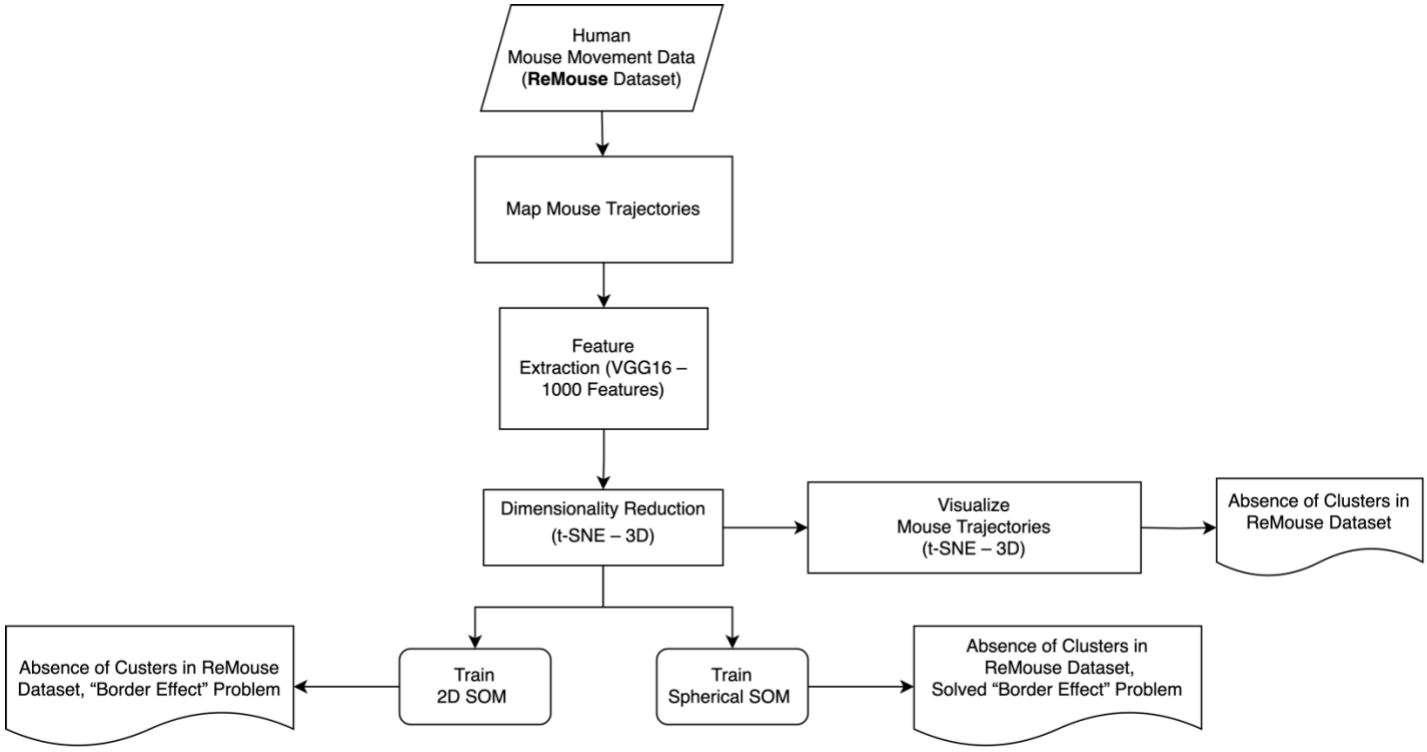


Figure 29. The flowchart of ReMouse dataset analysis using 2D SOM and Spherical SOM.

5.6.3 Data Analysis Using Unsupervised Clustering Techniques

In order to further validate our initial findings obtained by means of SOM heatmaps (and Spherical SOM), we have further performed an unsupervised clustering of ReMouse Subset-3 and ReMouse Subset-5 using the SOM clustering [88] (the python package provides an additional feature which enables automated identification of the main clusters within the formed map using K-means clustering algorithm), K-means clustering [160], and agglomerative clustering [161] algorithms.

An important result coming out of this stage of our research is obtaining the Silhouette and Davies–Bouldin scores, which were calculated by performing clustering on the two data subsets with a gradually increasing number of assumed clusters [162, 163]. The Silhouette score measures how similar an object is to its own cluster (cohesion) compared with other clusters (separation). A higher Silhouette value implies that points are well matched to their own cluster and poorly matched to neighboring clusters. The Davies–Bouldin score is the average similarity measure of each cluster

with its most similar cluster. Clusters that are farther apart and less dispersed will result in a higher Davies–Bouldin score.

Figures 30 and 31 depict the Silhouette and Davies–Bouldin score obtained using K-means clustering algorithms. Similar results have been obtained with the other two clustering algorithms. In the cases of all three algorithms, the highest values of the two scores are recorded for k = 2, suggesting that the optimal number of clusters is two. Figures 32, 33, and 34 provide 2D and 3D visualizations of the actual clustering results obtained on ReMouse Subset-3 and ReMouse Subset-5 using the three selected clustering algorithms and assuming k = 2. All three figures provide clear evidence that, even under the optimal number of clusters (k = 2) [58], the input data is pretty spread out throughout the input space, and many points that formally belonging to the same cluster are at a significant distance from each other. This further supports our earlier hypothesis that session trajectories generated by different users while completing the same online task are sufficiently distinguishable from each other.

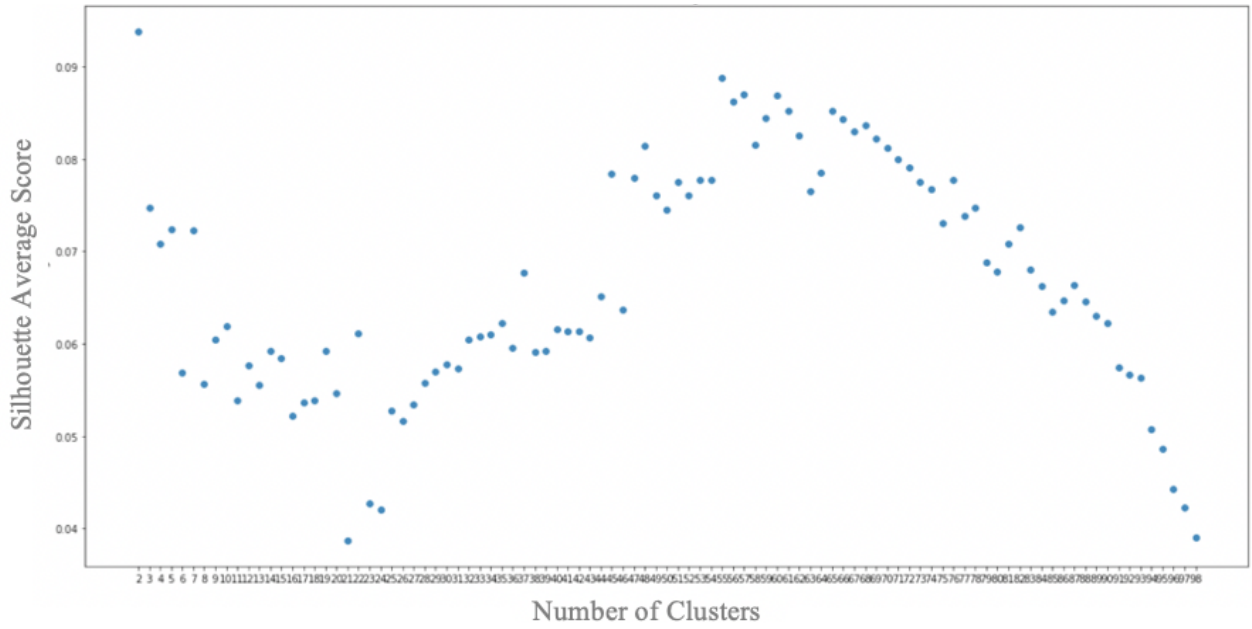


Figure 30. Silhouette average score[59].

[58] One could contend that determining an optimal value for K is challenging due to the inherent unclusterable nature of our dataset. This is attributed to the similarity in users' behaviors, which, despite appearing similar, are fundamentally different. Our empirical experiments further substantiate this observation, revealing that sessions generated by distinct users do not coalesce into discernible clusters. The input data is widely dispersed across the input space, with many points that would conventionally be assigned to the same cluster residing at significant distances from one another.

[59] The optimal clustering range, based on the results depicted in this figure, would be between 2 to 4 and 55 to 65 clusters.

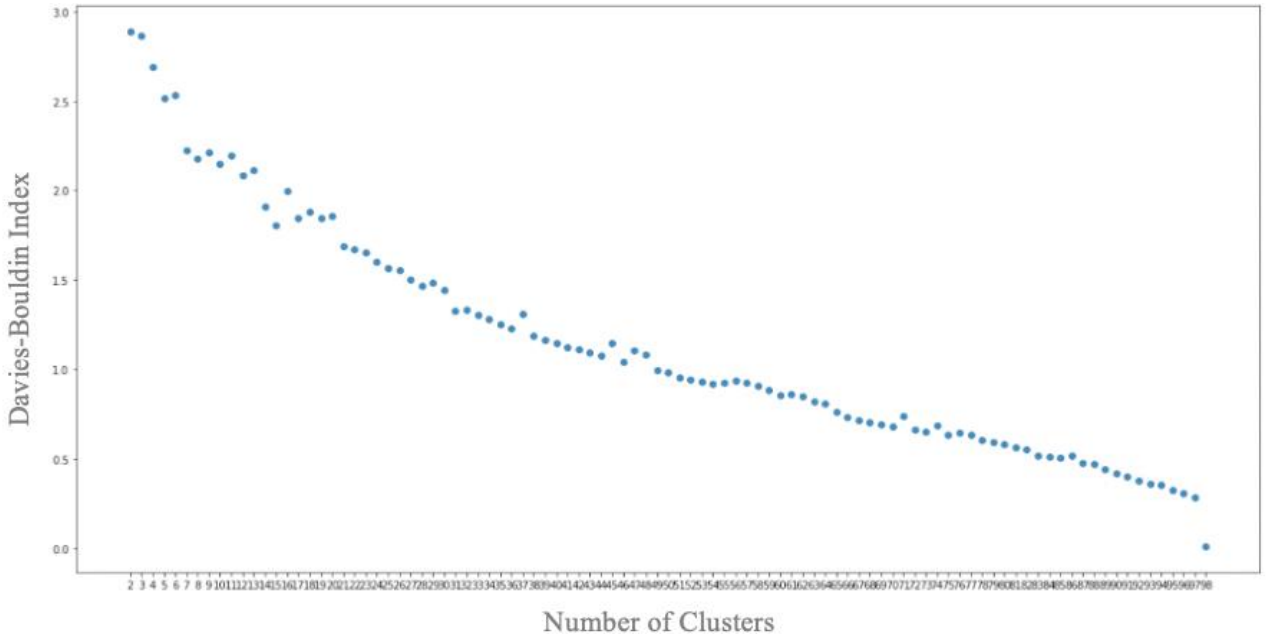


Figure 31. Davies–Bouldin index.[60]

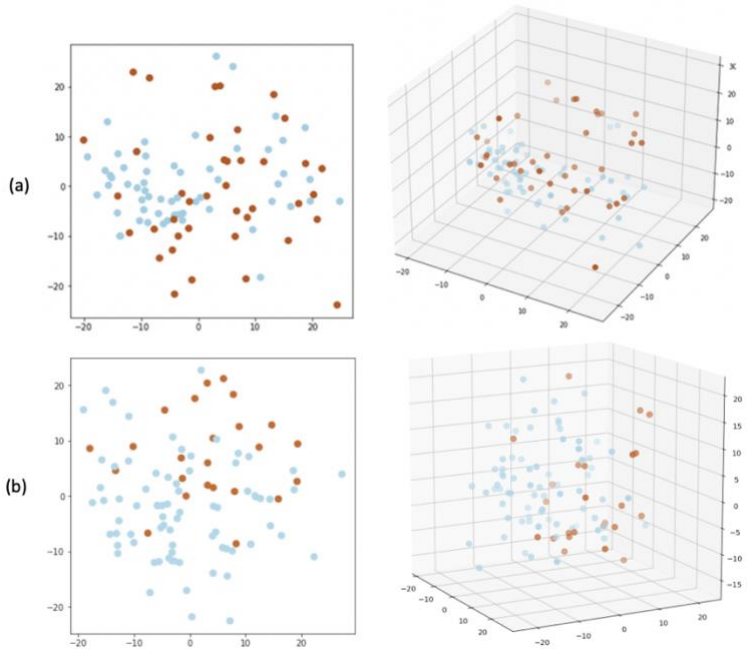


Figure 32. Unsupervised clustering visualization using SOM: (a) session number 3 and (b) session number 5 of all users.

[60] The optimal clustering range, as indicated by the results presented in this figure, falls between 2 to 4.

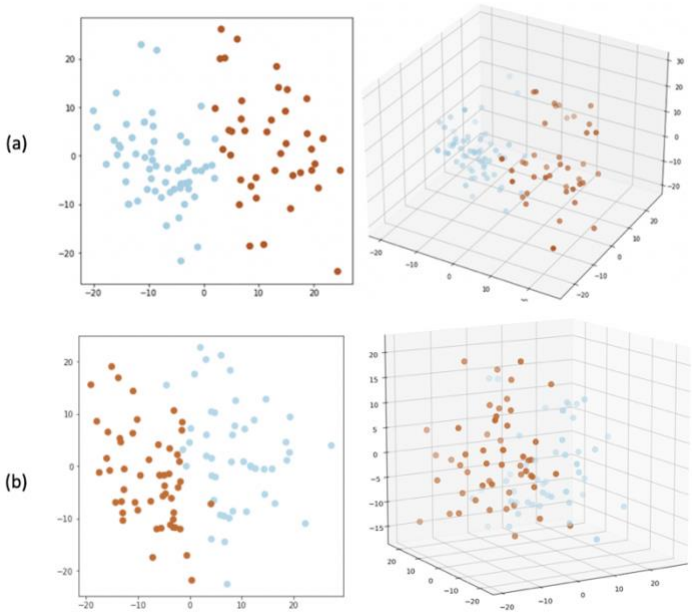


Figure 33. Unsupervised clustering visualization using K-means clustering algorithm, (a) session number 3 and (b) session number 5 of all users.

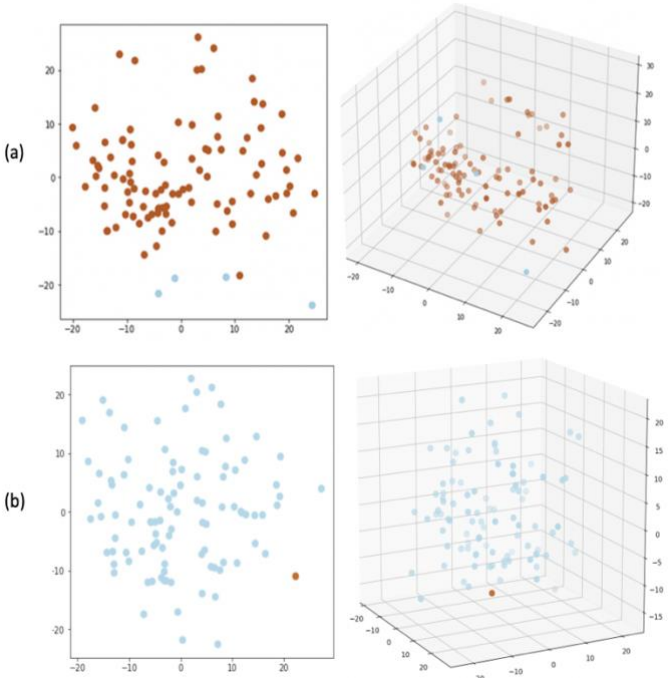


Figure 34. Unsupervised clustering visualization using agglomerative clustering algorithm, (a) session number 3 and (b) session number 5 of all users.

5.7 Conclusion

In this chapter, we have presented an in-depth analysis of our novel real-world mouse dynamics dataset, the ReMouse dataset. We first provided a summary of several publicly available mouse dynamics datasets. We then analyzed the ReMouse dataset using statistical and advanced ML-based methods, including deep and unsupervised neural learning.

In the first stage of the preliminary analysis using statistical methods, we focused on analyzing the sessions generated by each individual user in isolation from other users. Second, the focus was on the pairwise analysis of sessions generated by different users. Based on the preliminary analysis of our novel ReMouse dataset, we concluded that although sessions generated by genuine human users are relatively similar to each other, there always exist some minimum distinguishable differences between them. This further implied that sessions whose 'difference' from each other is below the determined threshold should potentially be flagged as 'replay' sessions generated by session-replay bots.

Considering the fact that the generalization value of the results obtained using manual feature selection is often questionable, we then investigated the (dis)similarities between comparable (same-order number) users' web sessions by utilizing image-based representation of ReMouse dataset and by means of advanced machine learning techniques. The results further supported our earlier hypothesis that session trajectories generated by different users while completing the same online task are sufficiently distinguishable from each other.

According to our knowledge, the ReMouse dataset is the first publicly available mouse dynamics dataset containing repeat sessions generated by the same human user(s). As such, this dataset can be a very valuable resource for research studies that aim to improve our understanding of (human) user behavior during repetitive interactions with the same website, with the ultimate goal of developing effective techniques for the detection of, and defense against, sessions-replay bots.

We believe that the ReMouse dataset contains enough statistical data to facilitate unbiased and high-quality research in the above-mentioned research areas. However, we also would like to point out a few possible, though minor, limitations of our dataset and work. One potential limitation of our dataset/work can be related to the platform we used to collect the data, MTurk. Although MTurk workers are generally pretty diverse when it comes to their place of residence or profession,

they tend to be less diverse in terms of their age, education, computer-use proficiency, etc. relative to the ‘general public’ [164].

In the forthcoming chapter, we will present ReBot (Replay Bot), our own software tool that emulates the behavior/functionality of session replay bots. In particular, we will first provide an extensive account of the methodology underpinning the development of the ReBot software. Subsequently, we will delve into the practical application of ReBot and outline the process of integrating the 'attack' data produced by ReBot into the ReMouse dataset.

Chapter 6

ReBot (Replay Bot): A Session Replay Bot Tool to Generate Human-like Mouse Trajectories

While the security community is generally familiar with the notion of 'session-replay bots', in reality (and outside of the dark web realm) it is hard if not impossible to gain access to an actual session-replay bot software. In the absence of software tools capable of emulating session-replay bots, it is very challenging to make a scientific analysis or conduct an actual research pertaining to this particular class of web bots. In this chapter we introduce Replay Bot (ReBot), which is a Selenium based software capable of recording and replaying human-generated mouse trajectories, and as such falls in the category of session-replay bots. ReBot has been exclusively developed by our research group to facilitate a credible study on session-replay bot detection and defences. Our experimental results demonstrate ReBot's excellent performance in generating replay sessions on real-world web sites, thus confirming its suitability and critical importance for state-of-the-art research in this particular field.

6.1 Introduction

The emergence of session-replay web bots has posed a significant challenge for the wider Internet community due to their highly inconspicuous nature. Namely, these bots are designed to closely mimic genuine human browsing behavior (i.e., mouse movements), and as such can launch a variety of hard-to-detect attacks. In addition, these bots are often distributed across numerous IP addresses, which further compounds the complexity of their detection process (refer to Section 2.3 for more details).

One particular application of session-replay bots[61] is observed in e-commerce domain, where they serve as advanced automated scrapers that first capture and then replay the activities performed

[61] These bots presenting security risks in several other areas such as:

by human visitors through their mouse and keyboard inputs. By doing so, session-replay bots allow hackers to conduct the so-called price scraping (i.e., illegal competitive price monitoring), which can seriously undermine the integrity and performance of the target/victim e-commerce platform(s). Hackers are acutely aware of the critical role of behavioral monitoring within the e-commerce sector, given its engagement in financial transactions. This understanding has prompted hackers to devise more intricate evasion techniques. Specifically, they have moved away from conventional scraper bots and are now employing session-replay bots that replicate human behavior. This strategic shift enables them to simulate legitimate user actions and potentially outmaneuver stringent bot detection mechanisms. (Refer to [165] for more information on automated scrapers.)

While the previous research on the operation and detection of simple web bots abounds, the specific topic of session-replay bots has been largely overlooked in the scientific literature. The likely reason for this situation is the fact that, according to our knowledge, there is no publicly or readily available software tool or code capable of emulating the operation of session-replay bots. (The access to such tools/code is likely confined to dark web and illegal marketplaces for hackers.)

In this chapter we introduce ReBot, our custom-made Selenium (browser) based software application written in Java and JavaScript that is capable of emulating the operation of session-replay bots. ReBot can be used in two main modes: Record and Replay. In the Record mode, ReBot tracks and stores all major mouse actions executed by the human user (i.e., bot operator) in a dedicated browser tab while visiting a 'target' web page. In the Replay mode, ReBot can repeat the previously executed/recorded mouse actions any arbitrary number of times ultimately generating that many 'new' visitations/sessions on the target page/server.

Credential Harvesting (capturing login details for unauthorized access), E-commerce Fraud (exploiting user interactions during online shopping, including payment information), Form Submission Attacks (automatically filling and submitting forms to exploit vulnerabilities or cause disruption), Exploiting Application Vulnerabilities (interacting with web applications to expose weaknesses), Account Takeover Attacks (manipulating sessions to gain control of user accounts), Privacy Violations (capturing sensitive user data, violating privacy policies and regulations), and Automated Fraudulent Activities (automating actions on websites for fraudulent purposes). Therefore, studying these bots is crucial for several reasons as follows:

•Security Awareness: Enhances awareness of potential threats and vulnerabilities.
•Risk Mitigation: Helps organizations develop effective security measures to mitigate risks.
•User Data Protection: Enables the implementation of robust measures to protect user data.
•Compliance with Regulations: Ensures compliance with privacy regulations and data protection laws.
•Proactive Defense Strategies: Facilitates the development of proactive defense strategies against evolving cyber threats.
•Preserving Online Integrity: Safeguards the integrity of online platforms by preventing fraudulent activities.
•Continuous Improvement: Allows organizations to adapt and improve security measures in response to emerging threats.

The structure of this chapter is organized as follows. Section 6.2 provides an overview of the existing literature related to the topic. In Section 6.3, we present a detailed explanation of the methodology employed in developing the ReBot system. In Section 6.4, we illustrate ReBot's real-world use and in Section 6.5 we describe the integration of 'attack' data generated using ReBot into the ReMouse dataset. Preliminary results of our analysis on session-replay web bot attacks are shared in Section 6.6, while Section 6.7 succinctly summarizes the chapter's key conclusions.

6.2 Related Work

When it comes to the previous research works on the subject of session-replay bots, we were able to identify only the following two studies:

1. In [97] the operation of the so-called Form-Spamming Playback Bot has been discussed. This malicious program is utilized for automatic account registration. It records the actions of human users as they interact with a webpage using a mouse and keyboard. Later, it plays back the recorded traces on the webpage to masquerade as the human user.
2. In [1] the so-called Replay Bot has been introduced. This bot utilizes the Global Mouse and Keyboard Library for Windows, which offers both record and replay capabilities. When a human user fills out a form, the bot records their actions and later impersonates the user by replaying the recorded traces on the same form submission page.

It should be noted that unfortunately the first model (i.e., the work of [97]) not only lacks any explanation about their bot's design and implementation, but also it does not provide access or reference to the bot's actual source code. The second study (i.e., the bot discussed in [1]) is based on a proprietary source code and dataset involving session-replay blog bots, which is a very narrow subcategory of web bots. Moreover, a common drawback of both studies is that they deploy a very simplistic assumption about the behavior of regular human visitors to the target site(s). Namely, they assume that genuine human users will never be in the position to generate similar or repeated sessions (or simply visit the target site multiple times), which in fact is a very common situation in news, social media, banking, or gaming websites.

6.3 Design and Operation of ReBot (Replay Bot)

Due to the lack of publicly or readily available tools capable of emulating the operation of session-replay bots, our research group has undertaken the task of developing such a tool using the latest knowledge and software available. Specifically, for the purposes of our research, we have developed Replay Bot (ReBot)[62], which is a Selenium Web-Driver, Java and Java Script based session-replay bot. The author of this thesis assumed diverse responsibilities encompassing various phases of the ReBot development and implementation process, as outlined below:

- Participating comprehensively in all stages of the ReBot's development, spanning the inception of requirements, architectural design, software construction, rigorous testing, and comprehensive documentation of the software (ReBot).
- Playing an active role in contributing to the coding aspect of the project, thus being an integral part of the software development process.
- Establishing a robust test environment by creating and configuring both client and server test environments, which played a pivotal role in ensuring the software's reliability and functionality.
- Undertaking the responsibility of collecting pertinent data and executing in-depth analyses of the obtained results. This entailed the development of a Python-based software analyzer tailored to evaluate the test outcomes.
- Leadership role in guiding the team, providing strategic direction throughout the software development journey, ensuring alignment with project goals and objectives.

The general main principle of ReBot's operation is illustrated in Figure 35. First, the attacker (ReBot operator) records his/her own original human session, including the respective mouse dynamic data, while visiting a target website. At the end of the session, the recorded details are stored in a file/script. Subsequently, at the time chosen by the bot's operator, ReBot is instructed to replay the recorded session by recreating (i.e., reading out) the steps from the previously stored file.

[62] https://amazing-aryabhata-661e97.netlify.app/docs/.
https://github.com/chenc118/eLoki2/releases/tag/0.6.0.

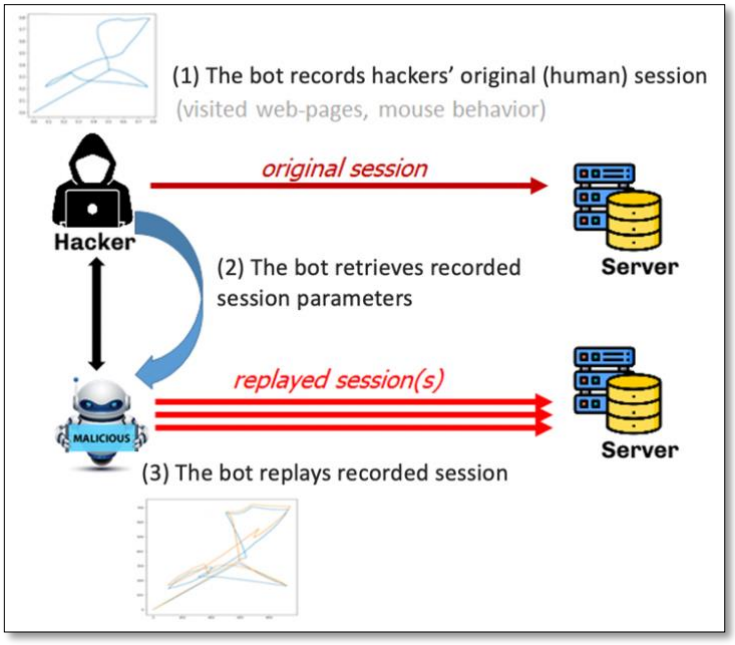


Figure 35. Session-replay bot replaying hacker's session.

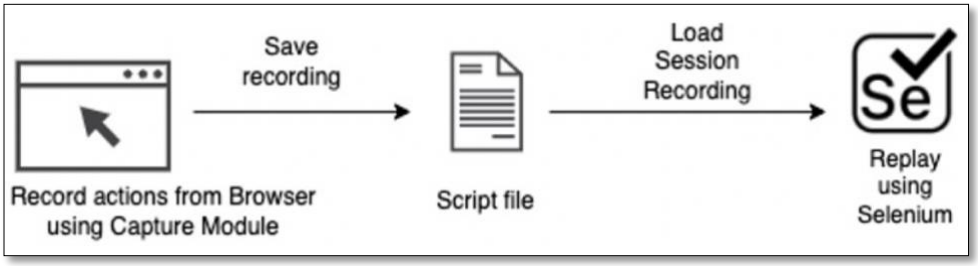


Figure 36. ReBot Flowchart.

Selenium Web-Driver is one of the key software packages used in the development of ReBot, which generally allows for the automation of network-based tasks and full in-browser page rendering. By means of Selenium Web-Driver, and in the initial recording phase, ReBot software gives the user a choice of browsers (Firefox or Chrome) through which the target site is to be accessed and the actual user's session (i.e., sequence of mouse actions) recorded. During ReBot's session replay phase, the same browser type (as the one chosen in the recording phase) will be evoked. Figure 36 shows the typical sequence of steps during ReBot's operation – from the original session recording to the ultimate session replaying. The specifics of ReBot's session recording and session replaying operation are described below.

A. ReBot's Record Module

This module is responsible for capturing the user's actions by listening to various mouse, key, and scroll events, and recording the time and order in which they occur.

Specifically, upon initial evoking of ReBot software in Record mode, an interactive browser window is launched allowing the user to enter the URL of the website that needs to be visited (see Figure 37).

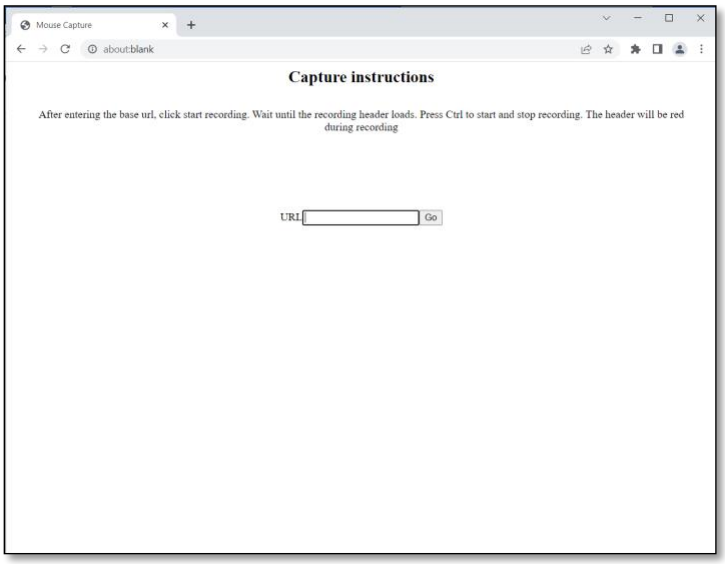


Figure 37. ReBot opens a new browser window in order to enter the URL of the website that needs to be captured.

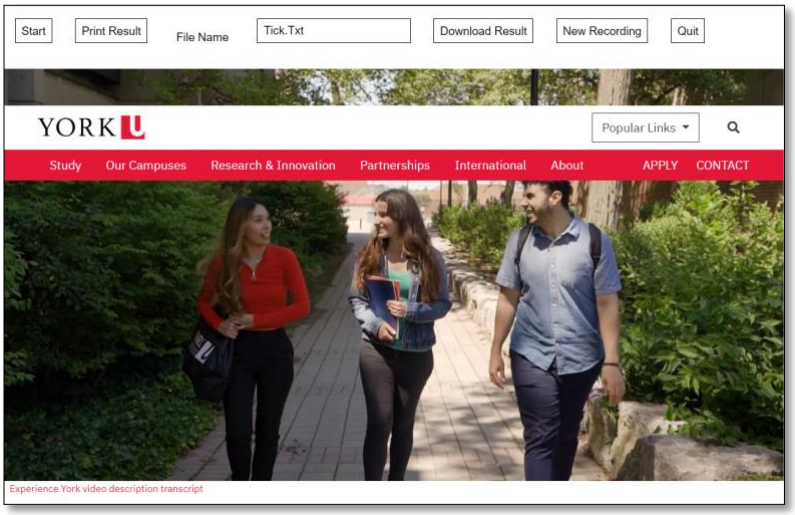


Figure 38. Successfully loaded page with ReBot Record Module.

Upon entering the URL of the target website, the loaded page is displayed along with ReBot's Record module header, which includes buttons for Start, Print Result, File Name, Download Result, New Recording, and Quit (as shown in Figure 38). To initiate the recording of a session, activation of the Record module header requires a click on the CTRL key, followed by clicking the Start button. Once the session starts, ReBot's internal logic captures the details of subsequent mouse movement/actions. To end the session (i.e., its recording), the user must click on the CTRL key once more to halt recording, followed by clicking the Download Result button. This action stores the recorded mouse movement and action details in the 'Tick.txt' file/script, which is subsequently used during ReBot's replay mode. Lastly, the session recording task is completed by clicking the Quit button.

- **Tick.txt Script Formatting**

Each line in the recorded script file, i.e., Tick.txt file, follows a general format of '@timestamp *actionName* [*parameters*].' (See Figure 39) The timestamp is in milliseconds and is typically obtained using a new date object in the browser. The following are some of the core actions that are recorded in the Tick.txt script:

- *mouseMoveScroll* [*x*] [*y*] [*sx*] [*sy*]: Captures current *x* and *y* coordinates of the user's mouse on the screen as well as the window's *sx* and *sy* scroll positions.
- *getPage* [*url*]: Logs the user's choice of navigating the browser to the specified URL.
- *click* [*x*] [*y*] [*cssSelector*]: Captures user clicks at the designated x and y coordinates. If a CSS selector is provided, it documents clicking attempts on the specified object.
- *right_click* [*x*] [*y*]: Documents instances where the user performs a right-click action at the given *x* and *y* coordinates.
- *resize* [*w*] [*h*]: Records user-initiated resizing of the browser window's visible area to the specified width (*w*) and height (*h*).
- k*eystroke* [*key*]: Captures key presses, indicating the specific key that the user interacted with.

```
@1665697166987 resize 1037 758
@1665697168647 getPage http://human-likebots.com/
@1665697168647 attachMouse
@1665697168790 mouseMoveScroll 465 559 0 0
@1665697168803 mouseMoveScroll 460 561 0 0
@1665697168820 mouseMoveScroll 453 567 0 0
@1665697168847 mouseMoveScroll 418 587 0 0
@1665697168857 mouseMoveScroll 387 599 0 0
@1665697168870 mouseMoveScroll 375 604 0 0
@1665697168887 mouseMoveScroll 352 614 0 0
@1665697168914 mouseMoveScroll 317 623 0 0
@1665697168920 mouseMoveScroll 304 625 0 0
@1665697168936 mouseMoveScroll 281 627 0 0
@1665697168953 mouseMoveScroll 277 627 0 0
@1665697168980 mouseMoveScroll 270 624 0 0
@1665697168993 mouseMoveScroll 263 618 0 0
@1665697169003 mouseMoveScroll 262 607 0 0
@1665697169020 mouseMoveScroll 264 602 0 0
@1665697169048 mouseMoveScroll 280 588 0 0
@1665697169053 mouseMoveScroll 340 547 0 0
@1665697169070 mouseMoveScroll 417 502 0 0
@1665697169087 mouseMoveScroll 462 478 0 0
@1665697169117 mouseMoveScroll 480 468 0 0
@1665697169120 mouseMoveScroll 521 441 0 0
@1665697169137 mouseMoveScroll 545 424 0 0
@1665697169153 mouseMoveScroll 549 420 0 0
@1665697169184 mouseMoveScroll 549 420 0 0
@1665697169194 mouseMoveScroll 549 418 0 0
@1665697169204 mouseMoveScroll 550 416 0 0
@1665697169220 mouseMoveScroll 551 414 0 0
@1665697169251 mouseMoveScroll 558 405 0 0
@1665697169262 mouseMoveScroll 561 401 0 0
```

Figure 39. A part of a recorded file Tick.txt.

A. ReBot's Replay Module

The purpose of this module is to execute the replay of user-recorded actions from the Tick.txt file. Specifically, upon invoking ReBot in Replay mode, the same browser type utilized during the recording phase is launched. To facilitate the replay process, ReBot's Replay module employs the Selenium framework, enabling the seamless reenactment of the previously recorded session within an actual browser environment. This functionality is integral to ReBot's ability to faithfully replicate the actions documented in the Tick.txt file, thereby accurately reconstructing the original user's interactions. An exceptional feature of ReBot is its capacity to iteratively reproduce previously executed mouse actions any desired number of times. Importantly, the Replay Module opens an actual browser window, affording users the opportunity to visually witness the authentic 'replay' of the previously recorded session.

6.4 ReBot - Performance Evaluation

In this section, we present the results of experiments that were conducted to evaluate the performance of ReBot software on our test website human-likebots.com. For the purposes of this analysis, we have developed four separate Python-based scripts to: (1) Conduct a comparative analysis of mouse movement coordinates between the original human sessions and their

corresponding replay sessions generated by ReBot, with an emphasis on visual inspection, (2) Quantify the Dynamic Time Warping (DTW) distance between the trajectories of the original human sessions and the respective replay trajectories produced by ReBot, (3) Perform a comparative assessment of mouse speed between the original human sessions and the corresponding replay sessions executed by ReBot, and (4) Undertake a thorough comparison of click events/actions in the original human sessions and their respective replay sessions facilitated by ReBot.

(1) *Comparison of mouse movement coordinates in original human vs. respective replay sessions produced by ReBot – visual inspection*

We conducted an assessment of ReBot's capacity to record and replicate mouse trajectory coordinates from an original human session, considering variations in both low and high mouse speeds. Our extensive analysis conclusively demonstrated that ReBot adeptly reproduces mouse-movement coordinates with visual fidelity across both speed conditions. A sample of the obtained visual results is presented in Figure 40. While this sample serves as an illustrative case, we have rigorously duplicated this comparative analysis across a multitude of trajectories encompassing diverse users. Remarkably, the outcomes remained consistent across hundreds of trajectories.

In the subsequent phase of our investigation, we employ the Dynamic Time Warping (DTW) distance metric to quantify the (dis)similarity between trajectories generated by actual human users and ReBot-generated trajectories. This endeavor serves to further corroborate our findings observed in this initial stage of evaluations.

(2) DTW distance between trajectories of original human vs. respective replay trajectories produced by ReBot

In order to systematically evaluate ReBot's performance, we conducted a comprehensive analytical study. This study involved calculating the Dynamic Time Warping (DTW) distance for various pairs of trajectories – where each pair consisted of one original trajectory created by a different human user and one respective trajectory created/replayed by ReBot, under scenarios of both low and high speeds. The main objective was to quantitatively assess the dissimilarity between the trajectories generated by human users and the corresponding trajectories reproduced by ReBot.

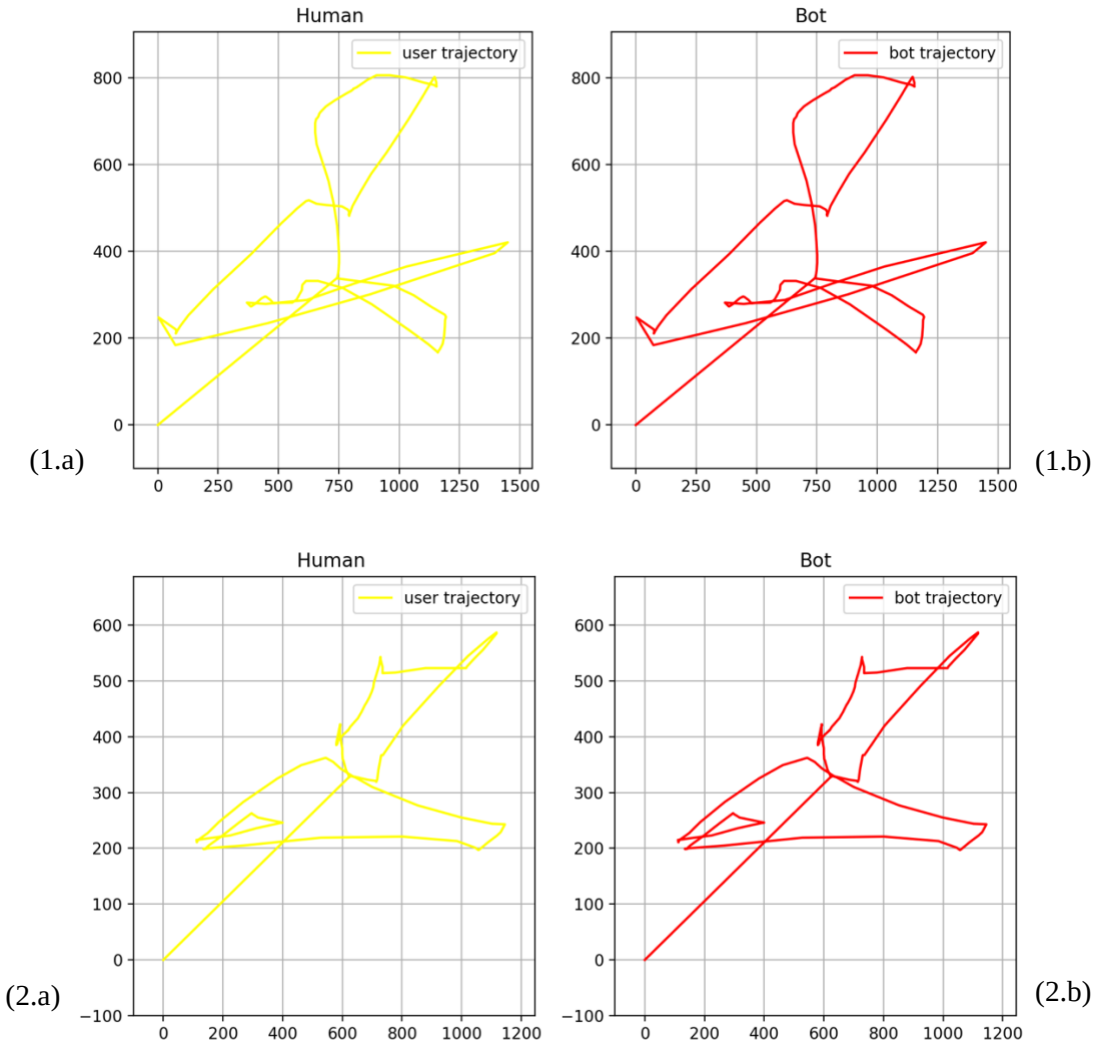


Figure 40. 1.a) Human trajectory in a slow-case scenario, 1.b) ReBot (replayed) trajectory in a slow-case scenario, 2.a) Human trajectory in a fast-case scenario and 2.b) ReBot (replayed) trajectory in a fast-case scenario.

Illustrated in Figure 41 is a representative example that demonstrates the appearance of one pair of mouse trajectories – the original and the replayed, and the respective value of DTW distance between them. This comparison serves to verify ReBot's capability to accurately capture and reproduce the originally initiated human sessions. Notably, across all instances, the computed DTW distance between the original and the replayed trajectory consistently yielded a value of zero. This observation signifies a remarkable accuracy rate of 100%, affirming ReBot's proficiency in faithfully reproducing genuine human user mouse trajectories, specifically in terms of coordinates.

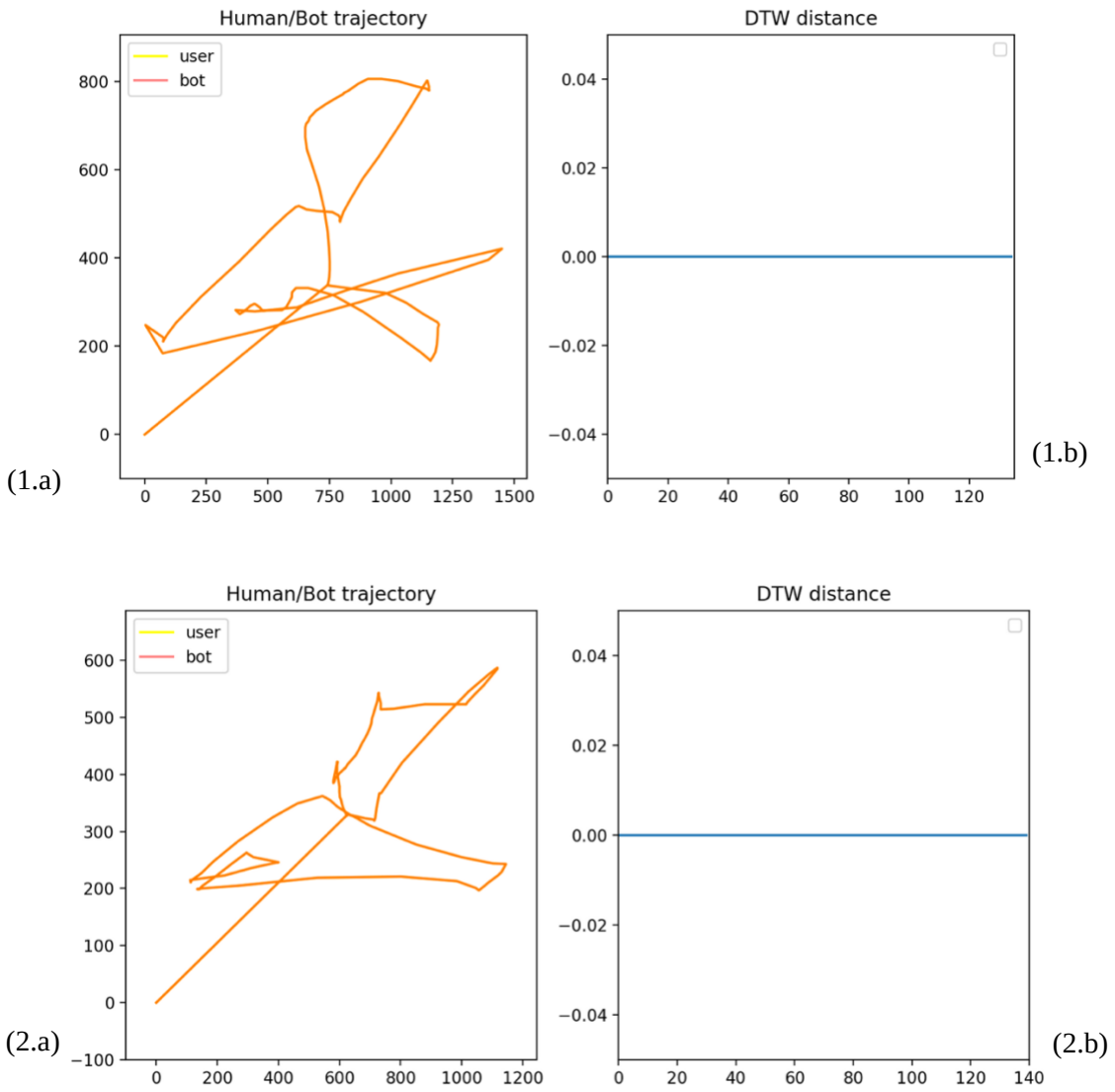


Figure 41. 1.a) Human and ReBot trajectories, in a slow-case scenario, 1.b) DTW distance between the two trajectories, 2.a) Human and ReBot trajectories, in a fast-case scenario, and 2.b) DTW distance between the two trajectories.

(3) Comparison of mouse speed in original human vs. respective replay sessions produced by ReBot

To assess the mouse movement speed in original human sessions compared to the corresponding replay sessions produced by ReBot, we conducted an extensive evaluation using trajectories generated by diverse users who employed ReBot in scenarios involving both low and

high mouse speeds. According to our empirical findings, ReBot exhibited an accuracy of approximately 92.1% in capturing mouse-movement speed within low-speed mouse trajectory scenarios, and an approximate accuracy of 98% in high-speed mouse trajectory scenarios.

In our pursuit of understanding the factors contributing to the accuracy variation, we conducted a comprehensive analysis of mouse movement measurements on both the client and the server sides. It's noteworthy that, on the client side, we implemented a polling interval of 5 ms (representing the frequency of the system's mouse position checks), while on the server side, a polling interval of 2 seconds was employed for client data updates, involving the transmission of buffered events to a remote server[63]. This configuration was carefully devised to strike a balance between the quantity of collected data and the precision of recorded events[64].

With this setup, when users executed slower mouse movements, the server captured a greater number of events, resulting in a larger overall volume of recorded data. However, this abundance of data posed challenges during the subsequent data recreation process carried out by ReBot. The increased number of events within a given timeframe introduced the potential for certain events to be inadvertently overlooked or dropped. Conversely, during instances of high-speed movements, the server recorded a relatively smaller volume of data. This streamlined data collection proved beneficial for ReBot's capability to more accurately replicate authentic human behavior, particularly in terms of mouse movement speed. This optimized data gathering facilitated ReBot's ability to mimic genuine interactions with greater fidelity.

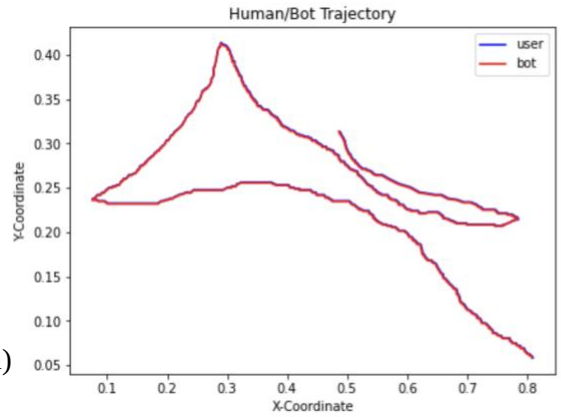
Figures 42.a) and 43.a) provide illustrative examples showcasing the trajectories of one human and one respective ReBot session under slow- and fast-case scenario. Figures 42.b) and 43.b) present the speed profiles of the respective human and the ReBot sessions/trajectories from Figures 42.a) and 43.a). Finally, Figures 42.c) and 43.c) highlight the average mouse speeds of the respective human and the ReBot sessions/trajectories from Figures 42.a) and 43.a).

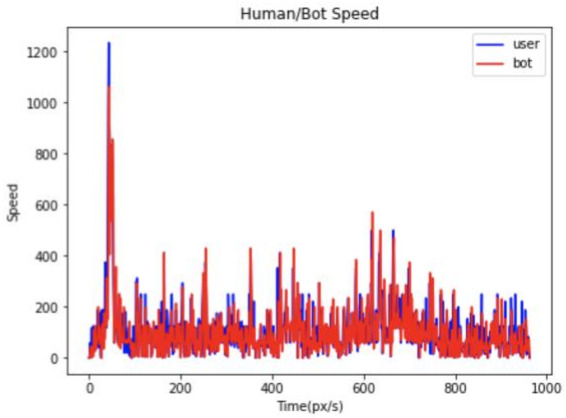
---

[63] In measuring mouse movement on the client side, the polling interval determines how often the system checks the mouse position. A lower interval, such as 1000 Hz (1 ms) or 500 Hz (2 ms), results in smoother and more accurate tracking, while a higher interval might lead to slight delays or reduced accuracy. When transmitting client data to a server, the polling interval refers to how often the client sends updates. A shorter interval provides quicker updates but can increase network traffic and server load. Conversely, a longer interval conserves resources but causes delayed updates. For optimal mouse movement tracking, a polling interval of 500 Hz (2 ms) to 1000 Hz (1 ms) is recommended. In client-server data transmission, the best interval depends on real-time needs and server capacity. Applications needing near real-time updates might use intervals of a few hundred milliseconds (e.g., 200 ms) to a few seconds (e.g., 2-5 seconds). The ideal polling interval varies based on factors like application nature, user experience goals, hardware capabilities, network conditions, and server load. [166].

[64] This setup was informed by a comprehensive review of previous research in the field [1, 74, 75, 98, 114, 117, 118, 166].

(a)





(b)

(c)

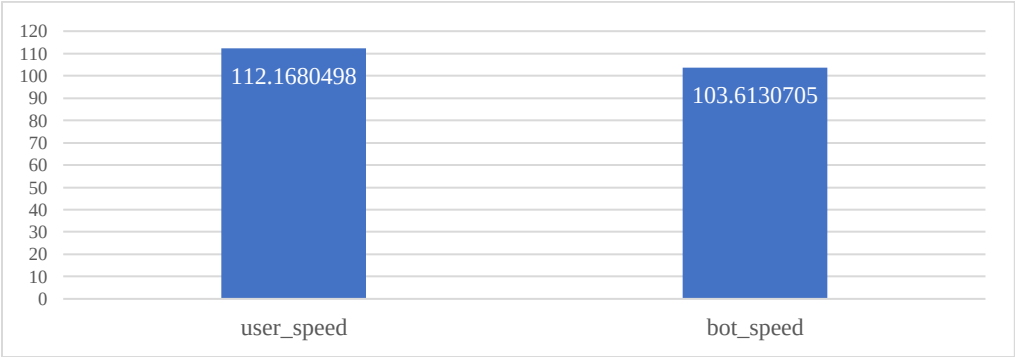


Figure 42. a) trajectories of the human and ReBot's replay sessions, b) mouse speed of the human and ReBot sessions as a function of time in the slow-case scenario and c) numerical values of average mouse speed of the human and ReBot sessions in a slow-case scenario.

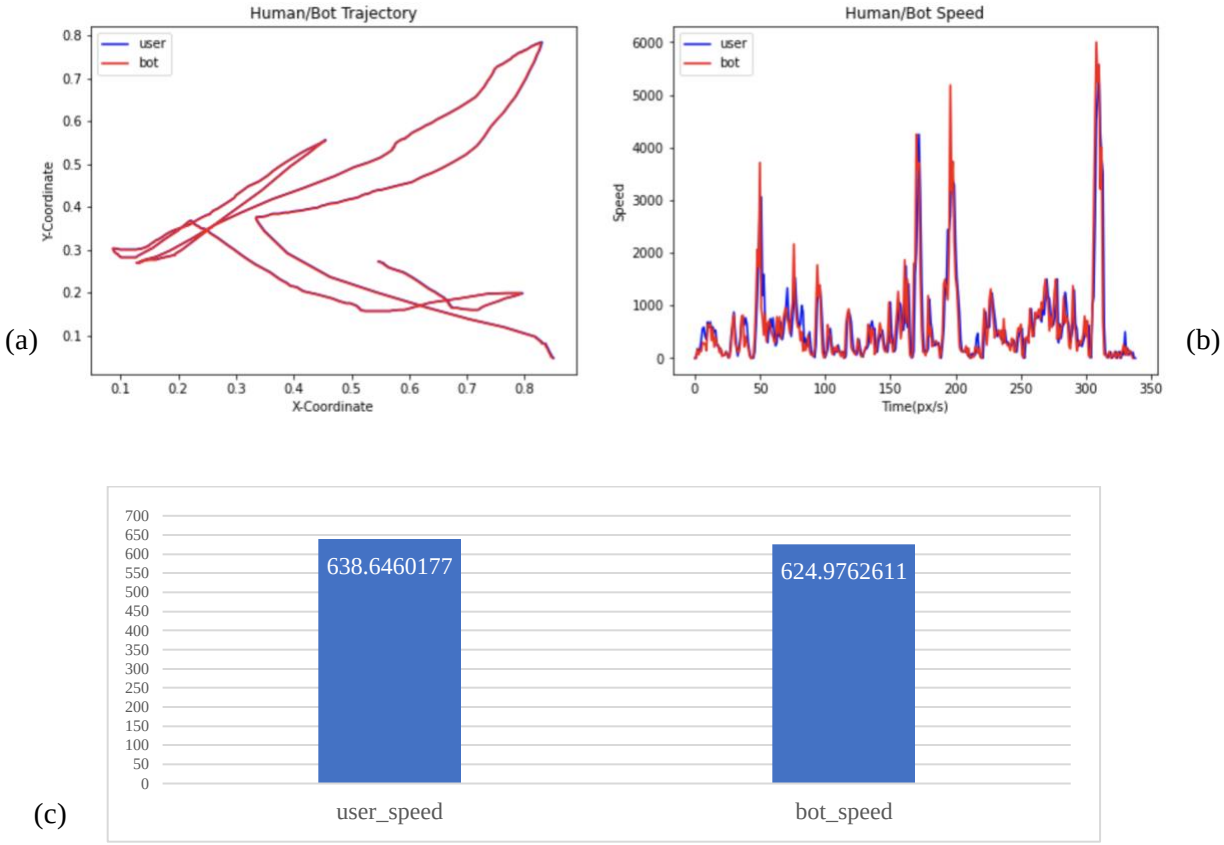


Figure 43. a) trajectories of the human and ReBot's replay sessions, b) mouse speed of the human and ReBot sessions in the fast-case scenario and c) numerical values of average mouse speed of the human and ReBot sessions in the fast-case scenario.

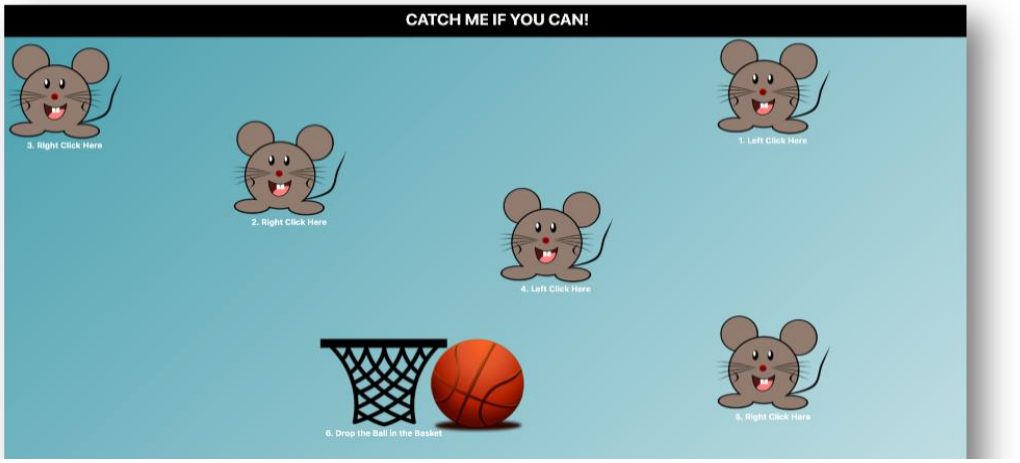
(4) Comparison of click events/actions in original human vs. respective replay sessions produced by ReBot

During our experimentation, we have discovered that ReBot's ability to capture and replay human mouse click events was adequate on majority of everyday websites (e.g., news agency, banking, social media, etc.), where on average only one or a few clicks are executed by the user in each 1 minute interval. However, ReBot's click record/replay performance was not as satisfactory in scenarios (i.e., websites) where the user was required to execute a large number of clicks in a relatively short interval of time – which was the case for our human-likebots.com website. It turned out that on our human-likebots.com site, and in both low- and high-speed mouse movement

scenario, ReBot was only able to capture and replay 2/3 of the clicks[65]. In order to deal with this performance limitation of our ReBot software, we have opted to make some minor modifications to human-likebots.com website (which has been built and utilized for capturing of real-world human-user and ReBot data, as explained in Section 5.3). In the next section, we describe these specific minor modifications made on our data-collection platform/website.

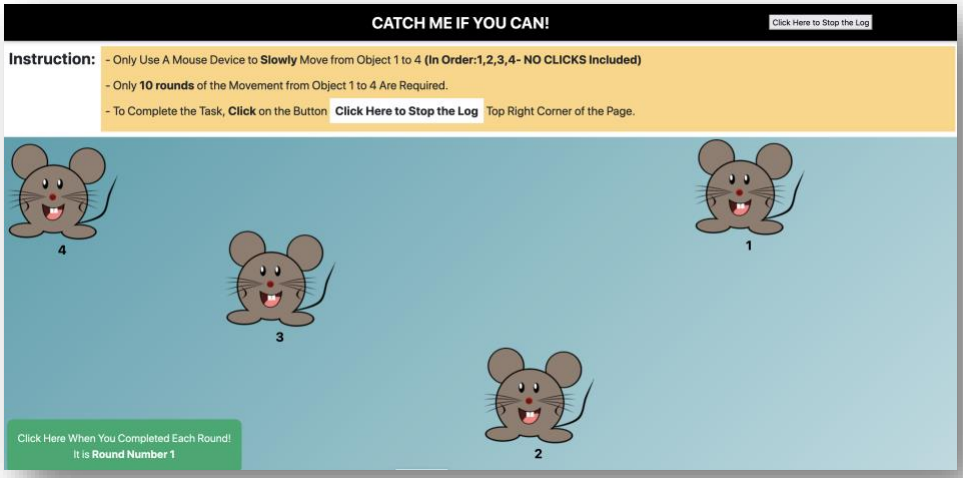
6.5 Data-Collection Website Modification and Repeated ReBot Evaluation

To ensure accurate capture and replay of click events by ReBot software on our human-likebots.com data collection platform, we have removed several of the mouse a right-click and drag-and-drop actions. Specifically, instead of being required to 'click on' each of the main enumerated objects/images on the website, now the user just needs to 'hover over' these objects until the end of the game/session is reached, at which point the user should make a click action if choosing to start another round of the game. Figures 44 depict the appearance of the original and the modified versions of human-likebots.com data-collection website/platform.



(a)

[65] Selenium WebDriver might encounter difficulties when recording and replaying distinct click actions, such as right-clicks or mouseDown/mouseUp events. These challenges stem from the intricate nature of browser interactions and the manner in which WebDriver interacts with them. Notably, certain interactions like triggering context menus via a right-click might not be accurately reproduced during replay. Similarly, the handling of mouseDown and mouseUp events by WebDriver might not faithfully replicate the intended behavior, resulting in disparities between recorded and replayed actions. For deeper insights into the reasons underlying the occasional failure of Selenium clicks, you can delve into the following reference: [167].



(b)

Figure 44. a) original version of "human-likebots.com", b) the modified version of "human-likebots.com".

To evaluate the ReBot's ability to accurately record and replay human mouse behavior on the **modified** version of human-likebots.com, we replicated the procedure detailed in Section 6.4. Specifically, we utilized the four distinct Python-based scripts that were developed, enabling a comparative analysis of mouse movement coordinates, mouse speed, and click events between the original human sessions and the corresponding replay sessions generated by ReBot. The evaluation results confirmed an enhanced performance of ReBot in terms of capturing and replaying of click events. Namely, overall, ReBot was able to achieve a remarkable 100% accuracy rate in terms of both trajectory and click event replays. Nevertheless, it's noteworthy that ReBot's proficiency in replicating the velocity of human mouse movements remained consistent, retaining an accuracy rate of 98%[66].

6.6 ReMouse2 Dataset Acquisition and Preliminary Analysis

This section outlines our second data collection process, which has been refined to ensure the precise capture and replay of click events by the ReBot software on the modified versions of the human-likebots.com data-collection website/platform.

[66] Appendix B contains illustrative examples of our comparative analysis.

6.6.1 ReMouse2 Dataset Acquisition

Similar to our previous procedure for collecting real human-user data on the original human-likebots.com page using the Amazon MTurk platform (as described in Section 5.3.2), we once again enlisted the help of 100 MTurk users by requesting them to visit and interact with our modified 'Catch Me If You Can!' game. The users were asked to play multiple rounds[67] of the game, for a total of 10 minutes. In each round of the game, the users were instructed to move the mouse cursor from object 1 to object 4 and then click on the green button located in the down-left corner of the webpage. (Note, this sequence of mouse events, where the cursor passes over multiple different areas of a webpage and is followed by a mouse click, is typical of human browsing behavior on majority of everyday websites.) We considered each round of 'Catch Me If You Can' game played by a particular user as a separate mouse movement (i.e., browsing) session.

In addition to collecting real human-user data on our modified human-likebots.com website, our next goal was to also expand/integrate this new dataset with malicious ReBot' s replay sessions. The integration of ReBot data into the new ReMouse dataset (we have named this new dataset ReMouse2), has been of critical importance for the proceeding step of our research on detection and defences against session-replay bots. The specific procedure of integrating ReBot sessions into ReMouse2 dataset is outlined in Figure 45 and described below.

Our human-likebots.com site was open for access by real human users through MTurk platform from January 3, 2023, to January 5, 2023. During this period, we played the role of a presumptive attacker by periodically[68] visiting and interacting with the modified human-likebots.com site (for a total of 10 visitations), and following the same instructions provided to the MTurk users. Each of these 10 sessions were recorded by ReBot and then replayed 3 times, thus emulating session-replay bot attack(s) - as described in Section 6.3. Figure 45 visually represents the process used to create the ReMouse2 dataset. As a result of this process, the ReMouse2 dataset comprises sessions generated by 100 authentic human users, 10 attacker sessions, and 30 ReBot sessions.

[67] 10 rounds.
[68] Roughly every 2 hours.

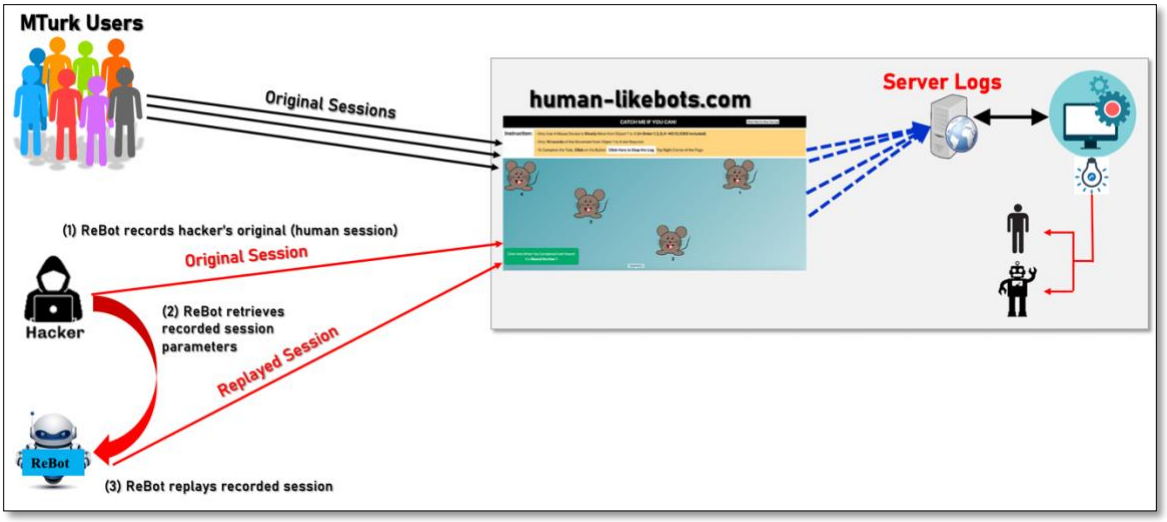


Figure 45. The process of collecting ReMouse2 dataset.

6.6.2 Preliminary Analysis of Human Sessions Only in ReMouse2 Dataset Using SOM Algorithm

In this subsection, we validate our previous finding from Sections 5.6.1 and 5.6.2[69], this time on the ReMouse2 dataset (**excluding ReBot sessions),** to once again demonstrate that two different human users cannot generate the same or similar-looking sessions when performing the same or similar online task. As before, we have used SOM and Spherical SOM for this purpose. The process begins with translating mouse trajectories into images, followed by utilizing VGG16 to automatically extract image representations' features from user sessions within the ReMouse2 dataset. Subsequently, we train two SOM maps, each with a size of 15-by-15. One map is trained with ReMouse2 Subset-3 (session number 3), while the other utilizes ReMouse2 Subset-5 (session number 3). The experimental setup for Spherical SOM remains consistent with the one discussed in Section 5.6.2.1.

As illustrated in Figure 46 - which specifically relates to sessions numbers 3 and 5 of all human users in the ReMouse2 dataset - ReMouse2 similar to original ReMouse dataset contains no pronounced clusters (i.e., all human user sessions in this dataset are sufficiently distinguishable). Also, similar to what was observed in Section 5.6.1, 2D SOM trained on ReMouse2 (the image

[69] In those sections, we employed the ML-based analysis of the curated image-based ReMouse dataset to investigate the (dis)similarities between comparable (same-order number) web sessions generated by different users.

representation of user sessions) is again affected by the "border effect" issue, while Spherical SOM does not suffer from this problem. Overall, these results confirm our earlier conclusion from Section 5.6 and derived from the original ReMouse dataset.

In the upcoming chapter, we will present a visual exploration of the ReMouse2 dataset, specifically examining the differences between human-generated vs. session-replay bot sessions generated by ReBot.

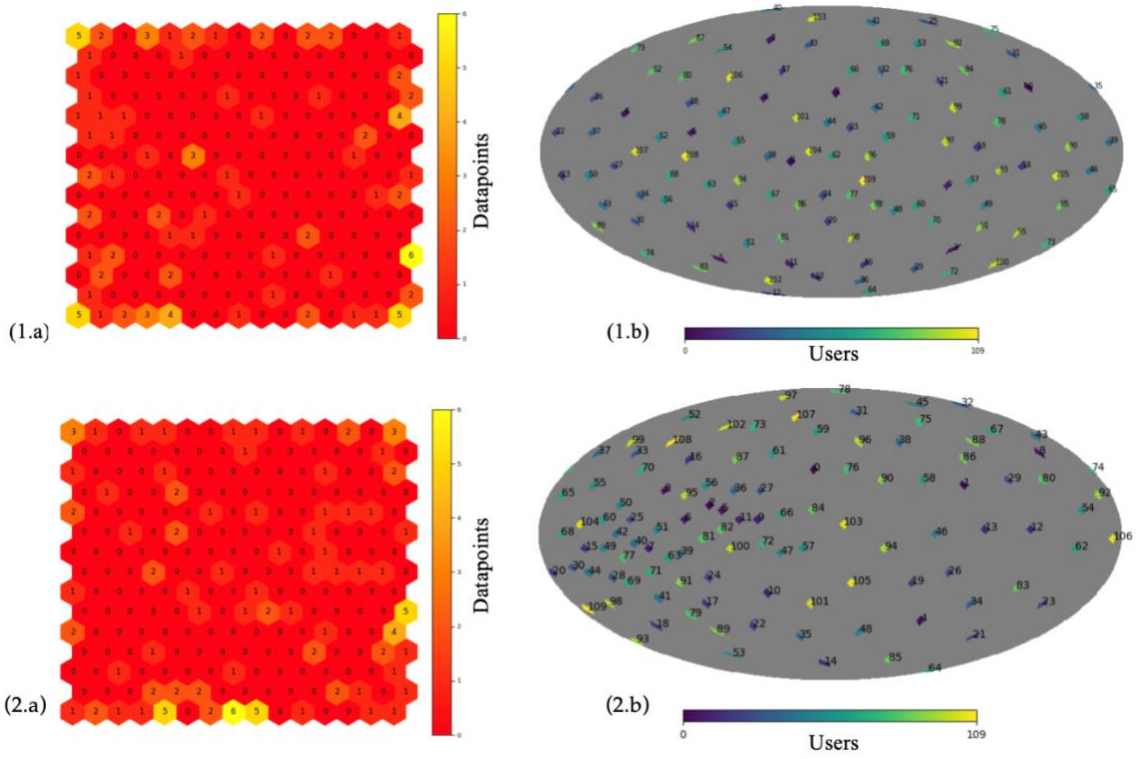


Figure 46. ReMouse2 dataset - users' datapoints map, session number 3, (1.a) SOM (1.b), Spherical SOM and users' data points map, session number 5, (2.a) SOM (2.b), Spherical SOM.

6.7 Conclusion

In this chapter, we introduced Replay Bot (ReBot), our own Selenium-based session-replay bot capable of recording and replaying human-generated mouse trajectories. We provided a comprehensive explanation of the methodology employed in the development of the ReBot software. Subsequently, we showcased our experimental results, underscoring ReBot's remarkable

capability to faithfully generating/replay the original human sessions on real-world websites. This demonstration solidifies ReBot's significance and suitability for our research on session-replay bot detection and defenses, as further elaborated in the subsequent chapters.

The remaining sections of the chapter delved into our efforts to adapt our experimental environment, particularly our human-likebots.com website, to achieve a high level of compatibility with ReBot software and some of its unavoidable performance limitations. We also detailed the process of assembling the ReMouse2 dataset, which was followed by the validation of our previous findings from the ReMouse dataset. This validation reaffirmed our earlier observation that disparate human users cannot generate indistinguishable or similar-looking sessions while executing analogous online tasks.

Moving forward, the subsequent chapter will present a visual exploration of the ReMouse2 dataset. This exploration will specifically focus on discerning distinctions between human-generated sessions vs. replay sessions generated by ReBot. Additionally, the chapter will propose an enhanced model for detecting the presence and activities of session-replay bots, such as ReBot.

Chapter 7

Detection of Session-replay Bot Attack(s) – Identified Pitfalls & Newly Found Solutions

In this chapter, we introduce the ReBotDetector, a specialized bot-detection system capable of identifying malicious replay sessions generated by ReBot software within the ReMouse2 dataset. ReBotDetector leverages an LSTM feature extractor to capture dynamic mouse data features and applies the Cosine similarity technique to recognize session(s) that exhibit a high degree of similarity with an earlier observed (i.e., the original human) session.

7.1 Introduction

The most simplistic approach to detecting ReBot sessions from a dataset such as ReMouse2 would entail an exhaustive pairwise examination of session data in order to identify instances that are identical or near-identical. While this technique may be useful when applied to a smaller dataset, it is generally ineffective and may lead to suboptimal results for progressively larger datasets. Namely, considering that a real-world bot attack may span over an extended period of days or months, it is reasonable to expect that the volume of collected data during such an attack becomes substantial, and any attempt to analyze the given data(set) in a pairwise manner would inevitably lead to a significant memory and processing overhead. (Please keep in mind that the recording of the mouse trajectory of a single user session – which consists of the timestamp followed by (e.g.) the respective mouse coordinates and speed, and are typically made in millisecond intervals – could potentially have thousands of timestamped entries.)

To mitigate these challenges, our approach seeks to overcome the resource-intensive nature of direct pairwise comparisons by implementing data compression. The objective of this compression is to represent the extensive data in a more compact form, thereby transitioning the detection process into a lower-dimensional space that demands fewer resources. This maneuver

inherently introduces a trade-off between accuracy and resource optimization[70]. Namely, by compressing the data, some information is inevitably sacrificed, with the remaining data being condensed and stored.

In view of the complexities outlined above, we have embarked on the path of leveraging 'data compression' techniques to streamline the processing and storage complexities pertaining to the detection of session-replay bots. This entails reducing the dimensionality of the session data, making it more amenable to efficient processing while striving to retain essential characteristics that facilitate effective bot detection.

This chapter specifically introduces our innovative approach, ReBotDetector, designed for detecting session-replay bots in ReMouse2 dataset. ReBotDetector is an integrated model explicitly crafted to identify session-replay bots that 'faithfully' replay the mouse trajectories of genuine user sessions. ReBotDetector utilizes an LSTM feature extractor to capture the most significant features from the mouse-dynamic data and employs the Cosine similarity technique to identify sessions closely resembling genuine user sessions.

Additionally, our detection model includes a complementary component: a t-SNE visualization tool which effectively demonstrates the capability of our model to successfully identify session-replay bots. Namely, this tool very evidently showcases that sessions generated by session-replay bots exhibit a substantial 'visual' overlap with the original sessions which they are derived from, thereby proving a high degree of similarity between them.

The chapter is organized as follows: In Section 7.2, we present our initial attempt to detect session-replay bots in the ReMouse2 dataset through an analysis of their image-based representations (similar to our analysis of the original ReMouse dataset presented in Chapter 5 and analysis of human-only sessions of ReMouse2 dataset presented in Chapter 6). However, after recognizing some suboptimal aspects of image-based detection process, Section 7.3 delves into our motivation to utilize a time-series representation of the ReMouse2 dataset for the purpose of detecting ReBot sessions. Furthermore, we discuss the design and operational details of our refined ReBotDetector, and we present some of our most significant experimental results obtained by deploying this detector on the ReMouse2 dataset. The primary conclusions drawn from this research are summarized in Section 7.4.

[70] However, it has been emphasized [181] that the next generation of bot detection techniques should aim to strike a balance between accuracy and data/algorithmic efficiency.

7.2 Image Representation & t-SNE Based Analysis of ReMouse2 Dataset for Visual Exploration of Session-replay Bots– Identified Pitfalls

In this section, we embark on an in-depth analysis of the key characteristics of ReBot sessions within ReMouse2 dataset, building upon our method previously employed on the original ReMouse dataset. (Recall, our original ReMouse dataset comprised of genuine human-sessions only.) In particular, our aim here is to leverage the technique from Chapter 5 (Figure 29) which comprised of the following steps: a) the conversion of mouse movements/trajectories into images, b) the use of VGG16 to extract the main features from images, and c) the use of t-SNE for the purposes of dimensionality reduction and visualization[71]. The approach outlined in Figure 47 extends the methodology initially applied to the ReMouse dataset (as described in Section 5.6.2.1 and Figure 29), so that it can now be applied to ReMouse2 dataset.

To distinguish between bot and legitimate human sessions in the ReMouse2 dataset visually, we applied annotations based on the ground truth. Sessions performed by MTurk virtual human users and hackers were considered legitimate since real humans generated them, while sessions replayed by the ReBot were labeled as malicious.

Recall from Section 5.5 that the primary purpose of the initial stages of the process outlined in Figure 47 (converting individual sessions/trajectories into images) was to automate feature extraction and avoid manual intervention (i.e., avoid the hand-picking of most significant features). The images are then processed by a pre-trained deep learning model (VGG16 [135]) to automatically extract their main representation features. By performing this process on the ReMouse2 dataset, we have again obtained 1000 primary extracted features.

To further reduce the number of features and consequently decrease the overall dataset processing time, we have applied the t-SNE technique (see Section 5.5). Specifically, we have used the Scikit-Learn implementation of the t-SNE algorithm [156] to map the entire ReMouse2 dataset

[71] VGG16 is a deep convolutional neural network (CNN) primarily created for image classification. It effectively extracts high-level features from images, but these features are generally high-dimensional and intended for classification purposes. These features can be numerous, ranging from thousands to tens of thousands, depending on the specific layer selected. Directly reducing these high-dimensional features to just 2 or 3 dimensions may result in a substantial loss of information, rendering them unsuitable for many tasks. On the other hand, dimensionality reduction techniques like t-SNE are explicitly designed to reduce data dimensionality while retaining as much meaningful information as possible. To address this, the common approach involves a two-step process. First, a deep learning model like VGG16 is employed to extract high-level features. Then, dimensionality reduction techniques are applied to these features. This sequential approach allows us to preserve the richness of the features while effectively reducing dimensionality.

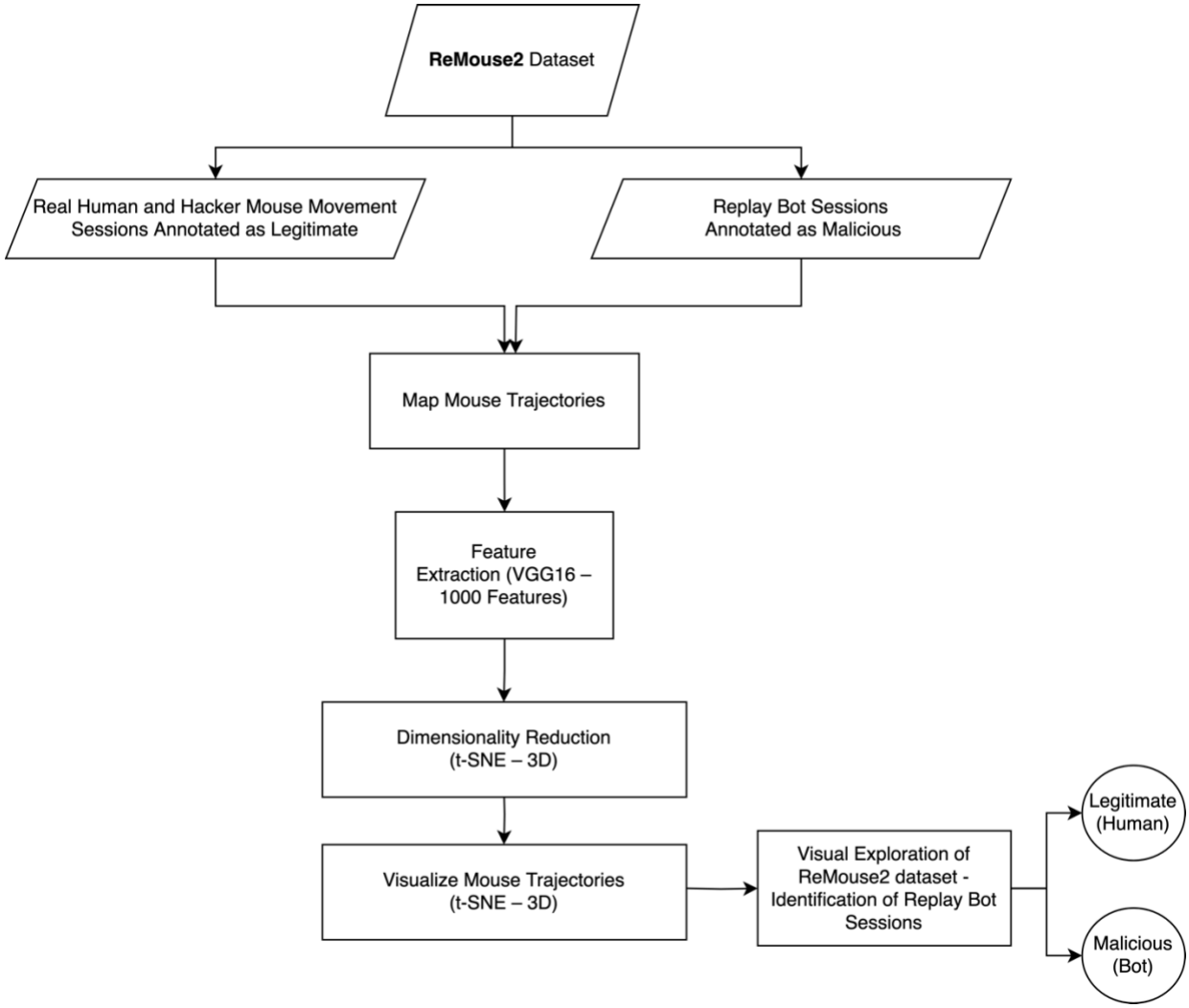


Figure 47. Image-based feature extraction & t-SNE for session-replay bot detection flowchart[72].

into three t-SNE components. We configured the perplexity value to 50, the learning rate to 200, the number of iterations to 1200, and the distance metric as "Manhattan." This approach not only

[72] It is crucial to emphasize that our study specifically addresses a scenario in which the adversary deploys a session-replay bot to record and replay their own browsing session within a website. In this context, sessions identified as identical are both labeled as 'malicious.' However, in a more aggressive approach, a hacker could inject a session-replay bot into a victim user's browser (computer). If the bot records a real human session and replays it, any resulting identical sessions, where the replayed (i.e., the second) session follows the first, would be labeled as malicious. This specific scenario falls outside the scope of our study.

reduced the number of features but also enabled us to visualize the distribution of the ReMouse2 dataset in a lower-dimensional space.

Figure 48 illustrates the outcomes of our visual exploration of the ReMouse2 dataset, following the process detailed in Figure 47. Regrettably, the results reveal potential drawbacks in applying this process to the ReMouse2 dataset for session-replay bot identification. Specifically, as depicted in Figure 48, out of the 30 malicious (ReBot generated) replay sessions, only 4 exhibited complete 'overlap' with their corresponding original human-generated sessions (i.e., the initial ReBot session recorded by the hacker). Ideally, when using the procedure outlined in Figure 47 for purpose of session-replay bot visual exploration, we should observe full overlap between all 30 ReBot sessions and their respective (original) hacker-generated sessions.

It is worth pointing out that the approach from Figure 47 yielded successful results in Chapter 5 primarily because of the fact that each datapoint/session in the ReMouse dataset originated from a distinct human user, and (as proven in Chapter 5) was sufficiently distinguishable from all other datapoints/sessions. However, the nature of ReMouse2 dataset is fundamentally different because the mouse movement trajectories replayed by the ReBot are actually indistinguishable (i.e., almost identical) to those initially generated by their human operator.

Given the above identified limitations of the bot detection process from Figure 48, we have been prompted to look for alternative bot-detection methodologies. In the following sections, we describe the motivation, main characteristics, as well as the results obtained with our new and improved approach for detection of session-replay bots in ReMouse2 dataset.

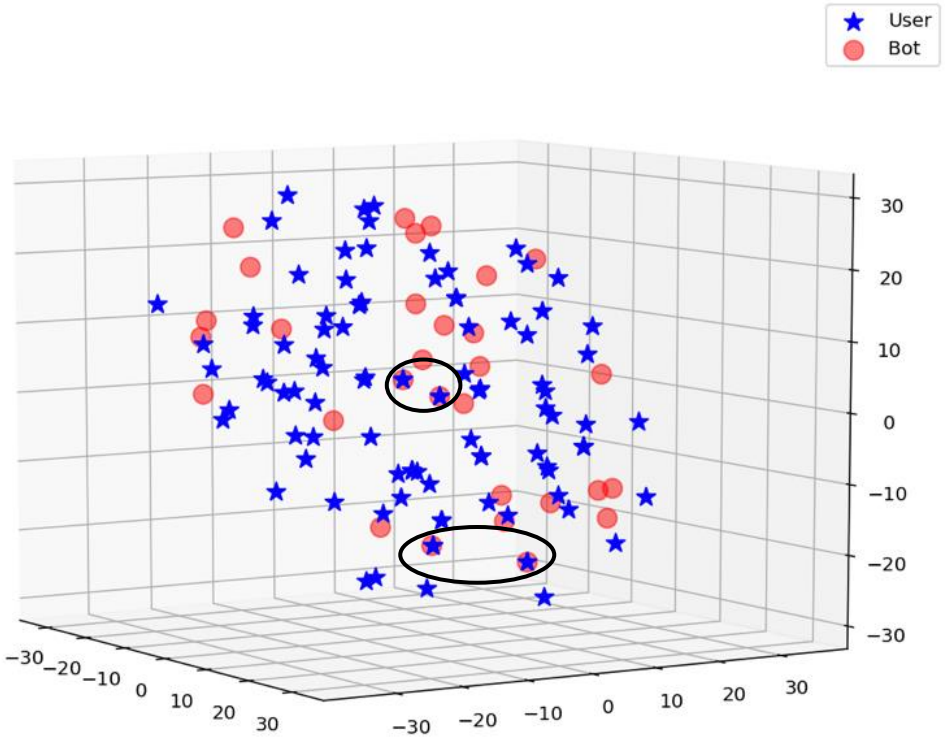


Figure 48. Visual exploration of ReMouse2 dataset for detection of session-replay bot attack – identified pitfalls.

7.3 ReMouse2 Dataset Analysis using Time-Series Based Mouse Movement Representations

7.3.1 Motivation for Deploying Time-Series Based Mouse Movement Representations

In general, two main approaches are traditionally used to model mouse trajectories generated by human users:

1) mapping mouse trajectories to corresponding images in order to facilitate more effective feature extraction, or
2) simply treating the trajectories as time-series.

Each approach exhibits its own strengths and limitations, and the choice between them depends on the research question that is being tackled as well as the characteristics of the dataset under investigation.

Image-based trajectory modeling offers several advantages:

- It provides visual 2D representations of input data and obtained final results that are easy to validate by the human researchers/investigators (e.g., for the presence of anomalies) [75].
- It facilitates automated feature extraction, as previously explained in this thesis.
- It can be used in conjunction with pre-trained deep learning models, which can save time and resources for the specific purpose of feature engineering.

On the other hand, the main limitations of image-based trajectory representations include:

- They are susceptible to noise and minor variations, as even some small and possibly insignificant changes in a mouse trajectory can significantly impact the appearance of the resulting/corresponding image, and thus lead to suboptimal results.
- They may require substantial pre-processing and normalization to ensure consistency across samples[73].
- They may not capture (or it may be very challenging to integrate) certain aspects of mouse behavior, such as velocity or acceleration, which can be relevant for specific types of analysis.

On the other hand, representing mouse movement data in the form of time-series offers several unique advantages, which include:

- Higher accuracy analysis. Namely, mouse movement data can be collected (and represented) with a very high temporal resolution, typically in the range of milliseconds, which is critical for achieving high accuracy in the data processing and in the final results. (Unfortunately, in the case of image representations of mouse trajectories, choosing the right resolution levels so as to achieve a desired accuracy is exceptionally difficult if not impossible.)
- Time-series representation of mouse movement data can easily encompass not only trajectory-related information (e.g., the mouse cursor coordinates), but also mouse velocity and acceleration. This can further improve the accuracy of the data analytics process and the final results.

[73] Utilizing image-based trajectory representations requires significant pre-processing and normalization efforts to ensure consistency across different dataset samples. Inconsistent or irregular data can distort image representations, affecting the accuracy and reliability of subsequent analyses. Pre-processing addresses noise and irregularities in raw trajectory data, while normalization standardizes data scales to facilitate meaningful data analysis. Neglecting these steps could lead to inaccurate and skewed representations, impacting the quality of research findings.

- Time-series representation of mouse movement data allows that some additional useful features be calculated - such as autocorrelation, seasonality, and trends – which can offer further deeper insights into the underlying patterns of the data.

However, there are also limitations associated with using time-series to model mouse movement data:

- In some cases, the patterns of mouse movements may be complex and challenging to interpret for the human researcher/investigator, especially in the absence of visual representation. The lack of interpretability can potentially hinder a deeper understanding of the final results and/or underlying cognitive processes in human users.
- The time-series representation of mouse movements is the clear preference when we have prior knowledge that user interactions involve only a few mouse movements. Conversely, if we anticipate that users will spend an extended duration on a page, possibly due to challenging queries or the specific nature of the search task, then opting for an image-based representation becomes the more suitable choice [169].

In the initial step of our analysis, we faced the challenge of dealing with highly diverse and complex mouse trajectory data in the ReMouse dataset. Notably, the actual count of time-stamped entries representing each user's session/trajectory varied significantly not just between different users but also between different sessions of the same user. In order to be able to process and compare such diverse sessions, the sessions clearly need to be 'normalized' (i.e., represented by the same fixed and much smaller number of features) prior to any further processing. However, extracting meaningful features directly from these trajectories was difficult, which led us to explore the use of images as a more structured and (visually) comprehensible representation of the mouse trajectory data.

Although our approach demonstrated its effectiveness in discerning differences in mouse movement behaviors among humans, as exemplified in Chapter 5, the use of image-based analysis to identify session-replay bot attacks in the ReMouse2 dataset has exhibited its own set of limitations. As previously pointed out, these limitations stem from the fact that the mouse movement trajectories generated by replay bots are very close (if not fully identical) to those initially produced by their human operators. Recently, our findings have also been confirmed by the research of Niu et al. [121], which underscores the intricate challenge of identifying session-

replay bot attacks. This challenge has motivated us to further investigate the use of time-series representation for the purposes of ReMouse2 dataset analysis.

7.3.2 Design and Operation of ReBotDetector (Session-replay Bot Detector)

To analyze the ReMouse2 dataset using mouse movement time-series representations, we have employed the approach outlined in Figure 49. We have named this integrated model ReBotDetector (Session-replay Bot Detector), and its primary purpose is to accurately identify session-replay bots that faithfully replicate genuine user sessions/trajectories. In this model, we utilize the LSTM (Long Short-Term Memory) algorithm [170] as an automated time-series feature extractor, and the t-SNE algorithm for data visualization. A similarity-based approach, specifically Cosine Similarity, is then used to effectively detect session-replay bots closely mimicking genuine user sessions/trajectories.

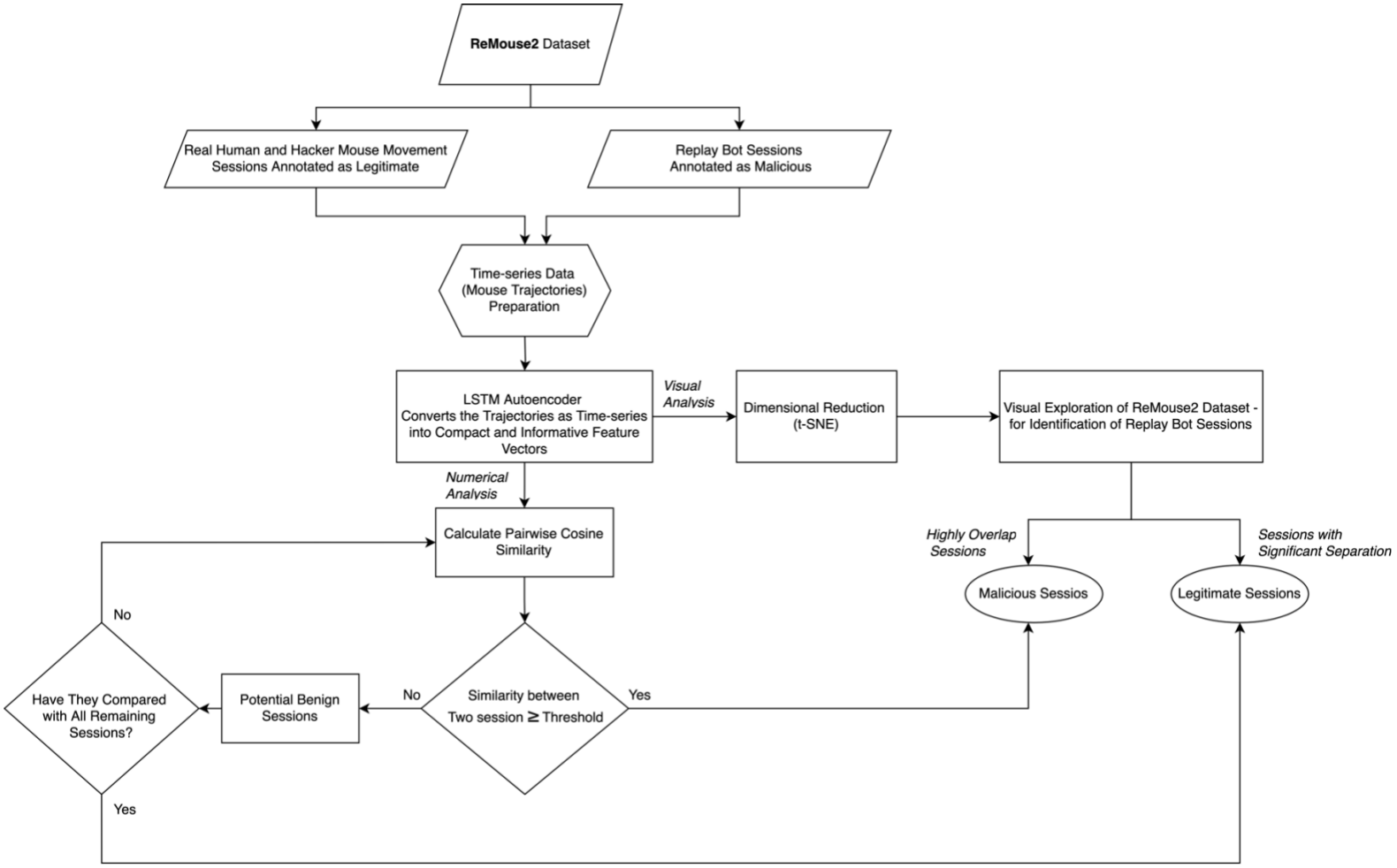


Figure 49. LSTM-based session-replay bot detection model - ReBotDetector.

7.3.2.1 Unsupervised Feature Extraction with LSTM Autoencoder

Long Short-Term Memory (LSTM) networks, a specialized type of recurrent neural network (RNN), excel in handling sequential data [189]. This property makes LSTMs particularly suitable for tasks like time-series analysis and classification, where data points follow a natural temporal order.

Unlike traditional feedforward neural networks, which struggle to capture temporal relationships, LSTMs are designed to effectively model and understand sequential dependencies. Often referred to as "smart" RNNs, LSTMs feature a unique architecture with a memory cell at their core.

The memory cell is governed by three primary gates:

1. Input Gate: This gate controls the flow of new data into the memory cell. It decides what information to store based on the current input and the previous state.
2. Forget Gate: The forget gate determines what information should be removed from the memory cell. It aids in discarding unnecessary or outdated data.
3. Output Gate: The output gate governs the information read from the memory cell, which forms the network's output. It computes the cell's current state and determines what to pass on as the final output [189].

By incorporating these gates, LSTMs can manage and update their internal state as new data points arrive, making them powerful tools for tasks involving sequential data.

Training an LSTM for a specific task encompasses a structured set of steps. Beginning with data preparation, the initial focus is on tasks like data cleansing, normalization, and scaling to homogenize the scale of input features. An indispensable component involves the transformation of the time series data into sequenced segments with predefined time steps.

Subsequently, the critical facet pertains to architecting the model. This entails the deliberate design of the LSTM model, with choices ranging from a singular LSTM layer to stacked LSTM layers. Furthermore, the process necessitates the determination of critical hyperparameters, notably the quantity of LSTM cells and the learning rate.

To facilitate productive training, the selection of an appropriate loss function tailored to the specific task is paramount. This may encompass Mean Squared Error for regression tasks or Cross-Entropy for classification objectives. Additionally, an optimizer, such as Adam or RMSprop, is judiciously chosen to regulate the model's weight updates during the training procedure [190].

During the training phase, the model assimilates knowledge by exposure to sequences of data, loss computation, and the subsequent adjustment of weights and biases through optimization algorithms. Finally, post-training, an evaluation is conducted on a distinct dataset to gauge the model's efficacy and suitability for the targeted task.

The LSTM Autoencoder is a type of unsupervised neural network where both the encoder and decoder are composed of Long Short-Term Memory (LSTM) networks. Serving as a powerful tool in learning encoding-decoding schemes from data, the autoencoder comprises an input layer, an output layer, an encoder neural network, a decoder neural network, and a latent space. During operation, the encoder compresses data into the latent space, and the decoder reconstructs the

encoded representation into the output layer. The autoencoder's key objective is not mere replication; by constraining the latent space to have a smaller dimension than the input, it forces the learning of the most salient features of the training data. This reduction in data dimensions retains the essential information of the data structure [191]. This approach is particularly useful for capturing meaningful patterns and features in sequences, such as time-series data, making it a powerful tool in data analysis and representation. In light of these precise specifications, we have opted to employ the LSTM autoencoder architecture in the design of our ReBotDetector.

Figure 50 depicts the architectural framework of the LSTM Autoencoder model utilized in ReBotDetector. The illustration outlines the methodology employed for processing and reconstructing sequence data using an LSTM Autoencoder.

The preprocessing of the ReMouse 2 dataset involves several key steps to prepare it for LSTM processing. Firstly, the "Data Cleaning" phase is implemented to enhance the dataset's reliability by addressing inconsistencies, rectifying missing values, and reducing noise in the raw data. Additionally, categorical data is transformed into numerical format to ensure compatibility with the model.

Following this, the "Normalization" step focuses on scaling the mouse cursor coordinates to the screen's width and height. This standardization process brings the coordinates within a consistent range, typically between 0 and 1, facilitating uniformity and compatibility for subsequent analysis.

The next step, "Data Segmentation," involves dividing mouse movement data into smaller sub-segments. The lengths of these sub-segments may vary based on user behavior, necessitating uniformity for effective model training. To achieve consistent session lengths, the methodology includes "Padding," whereby sessions are padded with placeholder values. This critical step ensures that all sequences share the same length, making them suitable for subsequent time-series analysis. The data is then transformed into a structured format compatible with LSTM models, specifically into input sequences characterized by three dimensions: samples, temporal times, and feature attributes[74].

[74] The LSTM encoder is responsible for the intake of these input sequences and their transformation into a condensed representation within a lower-dimensional latent space. This operation is achieved through the LSTM's ability to discern and encode temporal patterns and dependencies present in the data, yielding an output encapsulated by the shape parameters (batch_size, latent_dim), (see Figure 50). The next critical juncture is the bottleneck, where the data is in its most compressed form, otherwise referred to as the latent space or encoded representation. This condensed representation is effectively a distilled essence of the input data. A “Repeat Vector Layer” is employed to adjust the latent dimension

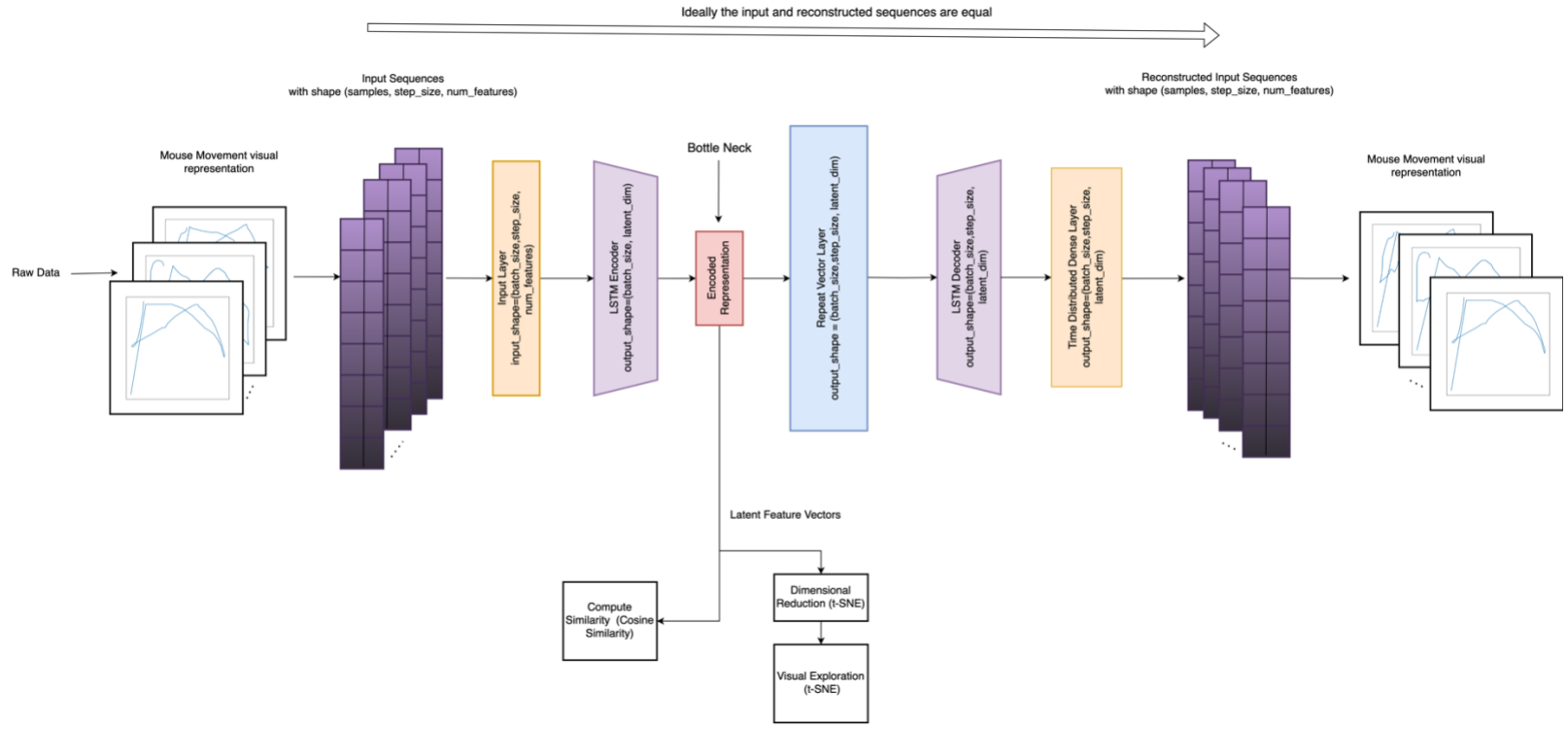


Figure 50. Architecture of the LSTM Autoencoder utilized in ReBotDetector.

The ReMouse2 dataset comprises a total of 110 users and 30 instances of RanReBots, each contributing/generating multiple sessions (1469 sessions in total). Within each session, 7 distinct features were recorded. These features include timestamps, mouse event types, mouse states, button states, mouse screen positions, as well as derived features such as distance and angle. It's worth

output from the LSTM encoder to the appropriate shape for reconstruction. This adjustment entails replicating the latent_dim vectors in accordance with the step_size, thereby resulting in a new shape of (batch_size, step_size, latent_dim). Following this, the LSTM decoder takes over, receiving the output from the Repeat Vector Layer. The Decoder's function is to reverse the encoding process, essentially re-expanding the encoded sequences into their original dimensions. Despite the repetitive nature of the step_size dimension from the Repeat Vector Layer, the decoder is tasked with learning to accurately reconstruct the input data, targeting an output shape mirroring that of the input (batch_size, step_size, latent_dim). The final layer in the sequence is the “Time Distributed Dense Layer”. Its role is to re-establish the num_features dimension from the latent_dim, culminating in the restoration of the input sequence to its original shape (samples, step_size, num_features). This reconstructed output should ideally match the initial input sequence, signifying a successful encoding and decoding process. Latent feature vectors extracted from this model serve as a basis for additional analyses, such as computing sequence similarity via Cosine similarity metrics or employing dimensionality reduction techniques like t-SNE for a more intuitive visual exploration of the data's intrinsic patterns.

noting that the last two features, distance and angle, are computed from the directly recorded user features.

To ensure uniformity in session lengths, padding[75] was applied to these sequences. The LSTM-encoder[76] was trained using a batch size of 128 and underwent 27 training epochs. The Adam optimizer was employed with a learning rate of 0.001, and the LSTM's hidden state had 64 dimensions.

As for the parameters deployed for the purposes of data visualization using t-SNE algorithm, we utilized the implementation provided by Scikit-Learn [156] and set the perplexity value to 50, learning rate to 200, the number of iterations to 1200, and the distance metric to "Manhattan."

To detect session-replay bots in the ReMouse2 dataset, we employed the Cosine similarity[77] technique, a valuable tool for identifying mouse movement samples with high similarity, particularly in systems vulnerable to replay attacks [121]. Cosine similarity calculates the similarity between two feature vectors[78], by measuring the cosine of the angle between them in a multi-dimensional space. It quantifies the alignment between these vectors, with a cosine similarity of 1 indicating identical feature vectors pointing in the same direction, and 0 representing orthogonal vectors with no similarity.

We set a threshold value of 100% to determine the necessary degree of similarity for classifying a session as a replayed one. Sessions with cosine similarity scores of this threshold are categorized as bots, while those falling below it are regarded as genuine human activity.

[75] In particular, we employed forward padding to fill missing values by duplicating the last observed data point and extending it to the right, ensuring continuity in the time series, https://www.alibabacloud.com/help/en/sls/user-guide/time-series-padding-function.

[76] The Encoder-Decoder LSTM can be implemented directly in the Keras deep learning library. https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM.

[77] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html#examples-using-sklearn-metrics-pairwise-cosine-similarity.

[78] We performed direct calculations of the Cosine similarity measure on feature vectors, each with a dimension of 64. As the computation of cosine similarity is a relatively simple mathematical operation, this approach was quick and efficient in terms of time.

7.3.3 Experimental results – ReBotDetector Performance

In this section, the final results obtained using ReBotDetector procedure on ReMouse2 dataset are presented. Figure 51[79] displays the distributions of the ReMouse2 dataset after ReBotDetector procedure, including t-SNE based visualisation, was applied.

The figure clearly shows that the replayed sessions generated by ReBot now have high overlap with the original session they were generated from. This further confirms the effectiveness of our novel approach in recognizing the basic version of session-replay web bot attack, and the usability of LSTM encoder as the feature extraction method in characterizing human mouse trajectories.

To further identify these replayed sessions generated by ReBot software, we employed Cosine similarity. This metric calculated the similarity between the features extracted from human data, which included sessions from MTurk users and the hacker, as well as the replayed sessions generated by ReBot from the hacker's sessions.

The obtained results are displayed in Figure 52, clearly indicating a 100% similarity match between the hacker's original sessions and the replicated sessions generated by ReBot. (In the calculation output shown in Figure 52, ReBot-R1-1 annotates the first replayed session produced by ReBot while recreating the initial session generated by the hacker which itself is labeled as hacker-R1.) As discussed in Section 6.6.1, each of the 10 sessions originally generated by the hacker was recorded by ReBot and subsequently replayed three times. Remarkably, both visually and using the Cosine similarity metric, all replayed sessions exhibited a flawless 100% match with the original human sessions from which they were derived, underscoring the accuracy of our detection method.

[79] For the purpose of visualization, we have constrained the number of data points displayed to highlight overlaps in cases where the data is easily discernible.

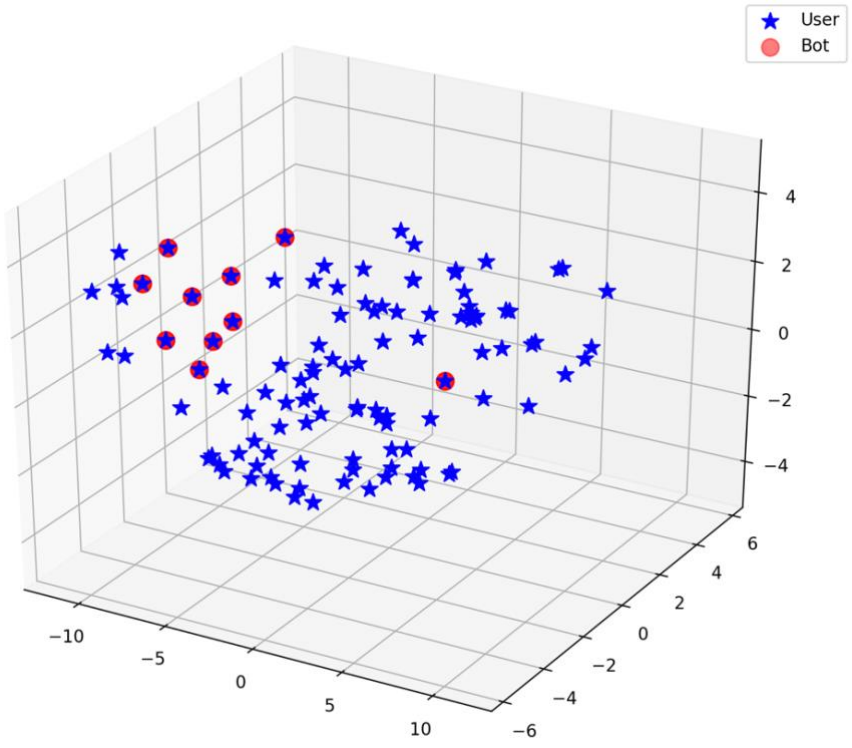


Figure 51. Visual exploration of ReMouse2 dataset using LSTM-based feature extraction & t-SNE.

```
User hacker-R1.csv matches with bot ReBot-R1-1.csv with cosine similarity: 1.0000001192092896
User hacker-R1.csv matches with bot ReBot-R1-2.csv with cosine similarity: 1.0000001192092896
User hacker-R1.csv matches with bot ReBot-R1-3.csv with cosine similarity: 1.0000001192092896
User hacker-R10.csv matches with bot ReBot-R10-1.csv with cosine similarity: 1.0
User hacker-R10.csv matches with bot ReBot-R10-2.csv with cosine similarity: 1.0
User hacker-R10.csv matches with bot ReBot-R10-3.csv with cosine similarity: 1.0
User hacker-R2.csv matches with bot ReBot-R2-1.csv with cosine similarity: 1.0000001192092896
User hacker-R2.csv matches with bot ReBot-R2-2.csv with cosine similarity: 1.0000001192092896
User hacker-R2.csv matches with bot ReBot-R2-3.csv with cosine similarity: 1.0000001192092896
User hacker-R3.csv matches with bot ReBot-R3-1.csv with cosine similarity: 1.0000001192092896
User hacker-R3.csv matches with bot ReBot-R3-2.csv with cosine similarity: 1.0000001192092896
User hacker-R3.csv matches with bot ReBot-R3-3.csv with cosine similarity: 1.0000001192092896
User hacker-R5.csv matches with bot ReBot-R5-1.csv with cosine similarity: 1.0000001192092896
User hacker-R5.csv matches with bot ReBot-R5-2.csv with cosine similarity: 1.0000001192092896
User hacker-R5.csv matches with bot ReBot-R5-3.csv with cosine similarity: 1.0000001192092896
User hacker-R6.csv matches with bot ReBot-R6-1.csv with cosine similarity: 1.000000238418579
User hacker-R6.csv matches with bot ReBot-R6-2.csv with cosine similarity: 1.000000238418579
User hacker-R6.csv matches with bot ReBot-R6-3.csv with cosine similarity: 1.000000238418579
User hacker-R7.csv matches with bot ReBot-R7-1.csv with cosine similarity: 1.0000001192092896
User hacker-R7.csv matches with bot ReBot-R7-2.csv with cosine similarity: 1.0000001192092896
User hacker-R7.csv matches with bot ReBot-R7-3.csv with cosine similarity: 1.0000001192092896
User hacker-R9.csv matches with bot ReBot-R9-1.csv with cosine similarity: 1.0000001192092896
User hacker-R9.csv matches with bot ReBot-R9-2.csv with cosine similarity: 1.0000001192092896
User hacker-R9.csv matches with bot ReBot-R9-3.csv with cosine similarity: 1.0000001192092896
```

Figure 52. Sample results of Cosine similarity calculation across all the sessions in ReMouse2 dataset to detect session-replay bots generated by ReBot.

7.4 Conclusion

In this chapter, we introduced the ReBotDetector, our proposed model designed to detect malicious replay sessions generated by ReBot software. We discussed the challenges associated with using image representations of mouse movement/trajectories in the context of identifying ReBot sessions, and we explained the necessary adaptations made in our detector to overcome these challenges.

In the following chapter, by taking the perspective of an advanced hacker interested in avoiding detection by ReBotDetector, our focus will shift to exploring the implementation of "trajectory randomization" technique in session-replay web bots. This technique aims to intelligently alter the trajectories of replayed sessions produced by ReBot so as to better emulate the authenticity and inherent randomness in human-user behavior, and thus effectively evade detection.

Chapter 8

Advanced Session-replay Bots: Design and Implementation

It is reasonable to expect that with the rapid advancements in artificial intelligence (AI), hackers will be increasingly leveraging these technologies to create even more sophisticated variants of session-replay bots. Even though we currently have no evidence that such AI-enabled bots exist in the wild, this chapter investigates a possible approach to employing "randomization" to intelligently modify the replayed session trajectories of the ReBot, thereby making them appear more authentic to human-user sessions. In this context, we introduce RanReBot, an innovative solution designed to improve the realism of these sessions while maintaining their fundamental structure. We provide empirical evidence of RanReBot effectiveness in evading the detection model discussed in the previous chapter (ReBotDetector), which was originally designed to and has been proven effective in identifying basic ReBot sessions.

8.1 Introduction

Randomization is a critical technique that introduces uncertainty and unpredictability into a system or process, and can be deployed for purposes of both - security defence as well as offence. The specific use of randomization for the purposes of security defence generally falls under the umbrella of so-called Moving Target Defence (MTD), which is a thriving subdomain of cyber security that has gain considerable attention over the past several years[80][171, 172]. On the other

[80] In the context of Moving Target Defense (MTD), randomization is a crucial strategy employed to enhance the security and resilience of a system. Some specific uses of randomization in MTD include *Network Address Randomization* (changing IP addresses and network identifiers to confound attackers aiming to pinpoint specific targets), *Code and Software Randomization* (shuffling memory layout, instruction order, and function locations to thwart exploitation of known vulnerabilities), *Software Diversity* (deploying varied software versions with slightly modified code to prevent attackers from exploiting common weaknesses), *Dynamic Resource Allocation* (distributing computing resources dynamically to obstruct attackers from predicting resource availability), *Randomized Access Control* (introducing randomness into access control policies to impede attackers' predictions of authorized access), and *Data Randomization* (shuffling or encrypting data before storage, transmission, or processing to hinder meaningful information retrieval by unauthorized parties).

hand, the use of randomization by cyber security offenders/hackers is not as evolved and documented, though some notable examples include: 1) deployment of randomized spoofed IP addresses during a DDoS attack, 2) selection of random bit/character sequences during a brute-force attack.

Within the specific context of session-replay web bot attacks (and from the perspective of a potential hacker), controlled randomization could play a significant role in modifying the trajectory of each replayed bot session, so as to ultimately increase the overall attack potency. Namely, as demonstrated in the preceding chapters, through the use of advanced data-processing techniques the detection of replayed bot sessions that are entirely identical to the original human session is possible. Yet, by leveraging trajectory randomization, the operator of a session-replay bot could attempt to 'modify' each replayed session making it appear as an authentic (i.e., non-suspicious) user interaction. Clearly, if actually deployed by hackers in the real world, this practice could pose a considerable challenge for security systems tasked with identifying and thwarting such attacks.

While we are not aware of any actual real-world existence of advanced session-replay bots that deploy trajectory randomization, we believe the emergence of this type of bots is inevitable if not imminent – especially as we witness the rapid progress in the development and utilization of AI, which has already shown to be the driving force behind the explosive advances in the domain of ChatBot technology. The work presented in this chapter aims to bring the attention of the research and cyber-security community to what is likely to be the next evolution in session-replay bots development, which is the use of stealthy randomization capable of producing attacks far more challenging to detect and defend against. In particular, our research provides an in-depth look at the specific challenges and possible solutions faced by those who embark on developing (i.e., hackers) as well as those who embark on defending against advanced randomization-based session-replay bots (i.e., security operators).

Now, when it comes to the development of advanced randomization-based session-replay bots by hackers, one of the key challenges that they need to tackle is determining the optimal type as well as the degree of trajectory randomization. For example, when it comes to the type of randomization, one that is accompanied with simple 'point based linear interpolation' may result in very unnatural (fake) looking trajectories with abnormal polygon-like or sharp-edge shaped appearance [173]. As for the degree of randomization, an excessive amount of randomization may

result in overly ‘unusual’ trajectories (relative to those of other web visitors/users), and as such could raise suspicion and trigger the scrutiny of sophisticated intrusion detection systems. On the other hand, inadequate levels of randomization run the risk of leaving discernible patterns in the bot's trajectory (relative to the trajectory of the original hacker’s session), which could also be easily detected and flagged as suspicious. Clearly, to circumvent detection effectively, it is crucial to find the optimal type and level of randomization which would ensure that the bot sessions/trajectories are sufficiently authentic (i.e., unique) but not too extreme in their appearance.

In this chapter, we introduce our innovative approach for intentionally altering the trajectories of ReBot sessions. This purposeful modification is designed to enhance the authenticity of these sessions, closely mimicking human-user interactions, albeit not being exact replicas of the original sessions generated by ReBot. We then provide empirical evidence showcasing the effectiveness of our method in evading the detection model discussed in the preceding chapter, which was capable of identifying ReBot sessions.

The rest of this chapter is structured as follows: Section 8.2 provides insights into our initial motivation for contemplating an advanced (i.e., randomized) version of session-replay web bot attack. Section 8.3 explores the concept and implementation details of our novel advanced/randomized session-replay bot software, RanReBot. Section 8.4 presents the performance evaluation of the attack. Section 8.5 discusses the ReMouse2.1 dataset, a new version synthesized from ReMouse2 using RanReBot. In Section 8.6, we demonstrate how the RanReBot attack successfully evaded the ReBotDetector system, which had previously detected the ReBot attack with precision. Finally, Section 8.7 concludes our findings.

8.2 Randomized ReBot (RanReBot): Motivation

As explained in the previous section, spatial trajectory randomization is a promising strategy that could ensure greater stealthiness of attacks conducted by means of session-replay bots. Spatial randomization entails the selection of a random location within a specified range around each selected point of the original mouse trajectory. Subsequently, these newly determined locations are utilized to interpolate the remaining points, ultimately forming a new trajectory. This process is visually depicted in Figure 53. Of course, this process has to take into account factors such as the webpage's size and the distance between its key elements. However, as also indicated in the previous section, in order to ensure the success of this strategy, the session-replay bot operator

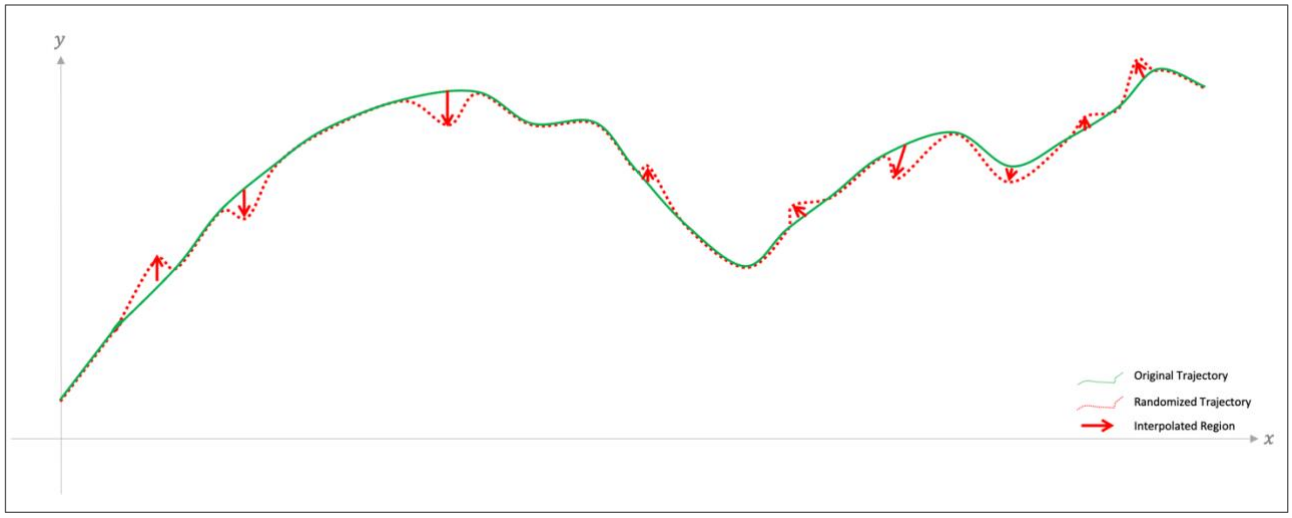


Figure 53. Spatial randomization of a mouse trajectory.

needs to determine not only the right degree but also the right type of trajectory randomization – which can be a very challenging task. (Overly extensive or overly erratic randomization of a trajectory would both be very obvious tell-tail signs that the trajectory is not generated by a genuine human user, and thus could be relatively easy to detect by a bot detection system.)

One of the goals of this thesis research was to conduct an in-depth analysis of possible trajectory randomization strategies deployed by a session-replay bot operator, so as to gain a better understanding of their actual evasiveness potential (i.e., their ability to avoid detection). Please note that the ultimate goal of this research is not only to advance the state of web bot development, but more importantly to advance the state of web bot detection. Namely, a better understanding of different randomization strategies from the perspective of a potential attacker/hacker is the first step towards developing effective defense systems capable of detecting and withstanding advanced session-replay bot attacks.

By building on the functionality of our original session-replay bot (ReBot) that was introduced and used for the purposes of research described in Chapter 6, in this chapter we introduce a specialized variation known as the Randomized Session-replay web Bot (RanReBot). RanReBot employs a point-wise randomization approach enhanced by advanced mathematical techniques, notably Bezier Curves [174]. The objective was to generate trajectories that appear more authentic, characterized by smoothness, and consequently become more challenging to detect (for more see Section 7.3).

Bezier curves have garnered significant attention in numerous research endeavors aimed at generating authentic mouse trajectories [75, 121, 122, 175, 176]. This popularity stems from their remarkable precision in controlling trajectory attributes, enabling the introduction of subtle variations in mouse movements, including adjustments in speed, acceleration, and direction. These fine-tuned modifications facilitate the creation of mouse paths that closely mimic those of genuine users, posing a considerable challenge for security systems to discern between legitimate and malicious activities.

Therefore, to enhance session-replay bot attacks and elude detection systems capable of identifying identical or replayed human sessions (such as ReBot sessions), we have leveraged Bezier curves. This intelligent approach has allowed us to modify the structure of ReBot (replay) sessions, making them distinct from the original sessions they were generated from while maintaining their fundamental structure.

8.3 Randomized ReBot (RanReBot): Concept & Implementation

As it is previously discussed In Section 6.3, ReBot works in two main modes: Record and Replay. In the Record mode, ReBot tracks and stores all major mouse actions executed in a dedicated browser tab. In the Replay mode, ReBot can repeat the previously executed/recorded mouse actions any arbitrary number of times. ReBot records mouse actions such as movement and click actions and saves them in a Tick.txt file during the record module's execution. When the replay module is executed, it repeats the actions from the Tick.txt file.

To transform and randomize the human trajectories recorded by ReBot from the Tick.txt file, we have devised and crafted a distinctive algorithm that leverages Bezier curves. This algorithm enables the creation of modified trajectories featuring controlled randomness while adhering to the recorded mouse movement data points. This transformation process unfolds in two distinct phases: i) *Point Selection,* where specific data points are strategically chosen on the original trajectory at fixed intervals, and ii) and *New Segment Generation*, which generates the coordinates for the points of the new segment using Bezier curves. This involves determining three control points (start, control, and end) for the implementation of Bezier curves (refer to Figure 54 for visualization). These control points are then utilized to derive a Quadratic Bezier Curve[81], which is subsequently

[81] A Bezier curve is a parametric curve employed in computer graphics and related domains. In our study, we've utilized the second version of Bezier curves, out of the three primary types distinguished by the number of control points. These

used to replace the respective segment (i.e., segment between the given control points) from the original trajectory. By doing so, controlled randomness is introduced into the newly derived trajectory (relative to the original trajectory) [177]. Ultimately, this approach allows that any desired number of new sufficiently 'human-looking' trajectories be derived, with each of them looking authentic (i.e., different from any other trajectory) while maintaining a sufficient similarity to the original trajectory.

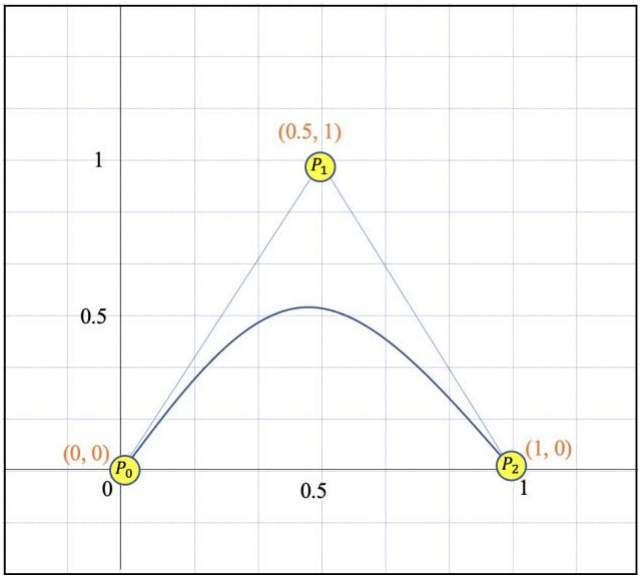


Figure 54. Quadratic Bezier Curve - defined by three control points ($\boldsymbol{P_0}$, $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$) with equation $\boldsymbol{B(t) = (1-t)^2 P_0 + 2(1-t)t\, P_1 + t^2 P_2}$. Curve coordinates depend on parameter $\boldsymbol{t}$ that changes within [0,1] range. Equations $\boldsymbol{x = (1-t)^2 x_0 + 2(1-t)t\, x_1 + t^2 x_2}$, and $\boldsymbol{y = (1-t)^2 y_0 + 2(1-t)t\, y_1 + t^2\, y_2}$ yield $(\boldsymbol{x, y})$ coordinates of the derived Bezier curve. An example with control points $(\boldsymbol{0, 0})$, $(\boldsymbol{0.5, 1})$ and $(\boldsymbol{1, 0})$ produces $(\boldsymbol{x, y})$ values according to equations $\boldsymbol{x = (1-t)t + t^2}$, and $\boldsymbol{y = -2t^2 + 2t}$ [174].

types are as follows: 1) Linear Bezier Curve: This simplest form of Bezier curve employs two control points: a start and an end point. It creates a straight line connecting the two points smoothly. 2) Quadratic Bezier Curve: The quadratic Bezier curve, which is our chosen version, involves three control points: a start, a control, and an end point. This curve introduces a higher level of complexity, generating a smooth trajectory that starts at the initial point, follows the influence of the control point, and eventually reaches the endpoint. 3) Cubic Bezier Curve: The most versatile of the three, the cubic Bezier curve employs four control points: a start, two controls, and an end point. This type of curve offers enhanced flexibility and control, leading to more intricate and varied curves in designs.

8.3.1 Randomized ReBot Algorithm Using Bezier Curves (RanReBot)

In this subsection, we present a comprehensive overview of our randomized ReBot algorithm, known as RanReBot, which leverages Bezier curves. This algorithm represents the cutting-edge approach in the realm of advanced session-replay bot attacks.

The RanReBot algorithm aims to enrich mouse trajectories with controlled randomness, utilizing recorded mouse movement data from the Tick.txt file. (Recall, the trajectory in Tick.txt is originally generated by a human user who was able to fully comprehend both the visual as well as the contextual information in the visited page.)

The RanReBot operation proceeds with the following steps:

• Data Loading: The algorithm initiates by loading of the original mouse trajectory data from the Tick.txt file.

• Mouse Movement Data Exclusion: A pivotal consideration in the development of the randomized algorithm is the precise nature of the mouse movement event data. These events are recorded based on their respective timestamps and encompass both mouse movements and mouse click events. Please note that we made a deliberate decision to exclude the mouse click events from the Tick.txt file. This exclusion was undertaken with a specific research focus in mind, aiming to concentrate solely on the analysis and randomization of mouse movement data. By excluding click events, the algorithm can concentrate on modifying and enhancing the continuity of the mouse movement trajectory, thereby refining the study's overall objectives and outcomes.

• Randomization Parameter Selection: This control parameter plays an important role in determining the extent of randomization applied in our experiments, as it allows us to fine-tune the degree of randomization as well as to limit its application to specific segments of the original trajectory. Based on our comprehensive experimentation, it has been established that a randomization level of 30% is empirically sufficient to achieve the desired balance between introducing controlled randomness and preserving the essential characteristics of the trajectory data. (Though, our algorithm allows that any other percentage of randomization be deployed in practice.)

• Point Selection: In this step, the algorithm deliberately identifies strategic points within the original trajectory. These strategic points are selected at fixed intervals (precisely every 10 data points in the Tick.txt file), and will serve as pivotal positions for introducing controlled randomness (see Figure 55).

• Randomization Process: The core of the algorithm lies in the randomization process applied to the strategic points from the previous step. Specifically, the algorithm assigns/finds a starting point (the x-coordinate of the current strategic point) and an ending point (the x-coordinate of the strategic point positioned 10 units ahead). It then determines a midpoint between these two points. To introduce controlled randomness, a random offset value within the range of 0 to 100 is generated and added to the midpoint, resulting in a control point. These three points (start, control, and end) are then used to define a quadratic Bezier curve.

• Trajectory Update: The Bezier curve function is employed to generate x-coordinates for new points located between the start and end points. These newly generated x-coordinates are utilized to update the original trajectory, effectively introducing controlled randomness while preserving the trajectory's overall structure (see Figure 56).

• Iterative Process: The algorithm repeats this randomization process, selecting a new strategic point from the list and applying the same procedure. This repetition continues until the desired percentage of randomization is achieved and evenly distributed across the entire trajectory.

• Output: The result is a modified trajectory, realistically randomized yet distinct enough to avoid detection as an exact human replay.

The pseudocode for the RanReBot algorithm utilizing the Bezier Curve function is presented in Figure 57.

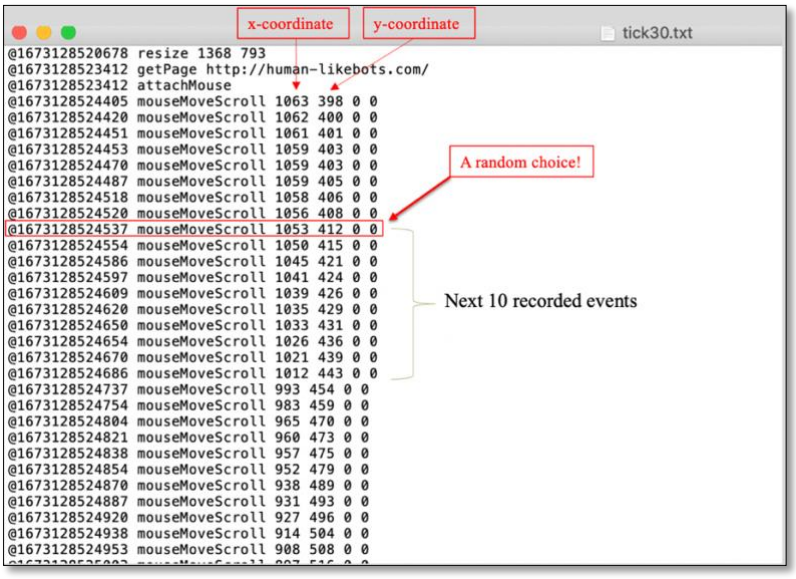


Figure 55. Tick.txt file - ReBot captured events from human mouse movement including timestamps, event type and coordinates.

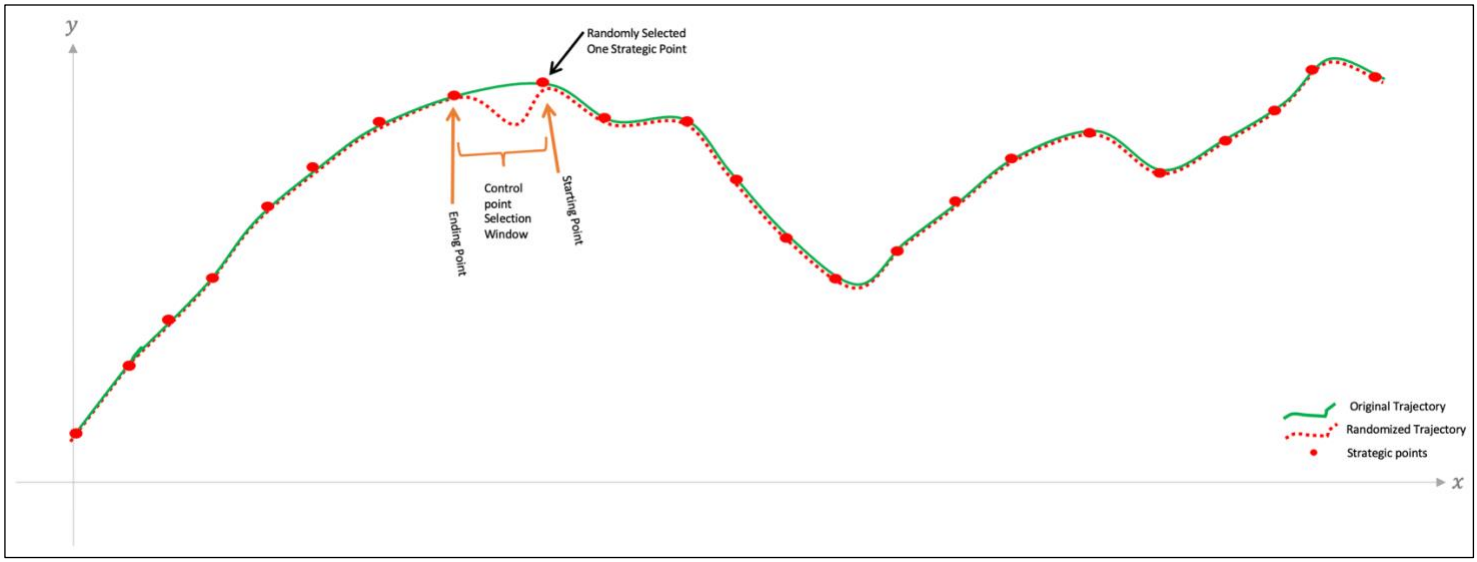


Figure 56. RanReBot algorithm utilizing Bezier curves for mouse trajectory randomization.

With regards to step 'Randomization Process' of the above RanReBot operation procedure, one specific question may arise: Why did we opt to exclusively manipulate the x-coordinates of strategic points in RanReBot trajectories within our randomized algorithm using the Bezier curve, while leaving the y-coordinates intact? This strategic decision was arrived at after our meticulous evaluation, taking into consideration the potential consequences of altering points along both the x and y axis. As our evaluation has shown, alterations along both axes might lead to excessive interpolation, resulting in trajectories that could become overly erratic and potentially diverge from representing human-like behavior. By confining randomization solely to the x-coordinates, we maintain a delicate balance between introducing controlled randomness while preserving the essential characteristics of the original trajectory.

Algorithm 1: Randomized Session-replay Bot Using Bezier Curves (RanReBot)

```
# Load original trajectory from 'Tick.txt'
original_trajectory = load_trajectory('Tick.txt')

# Define the percentage of original trajectory to be randomized
randomization_percentage = 30

# Calculate the number of randomization iterations required
total_points = len(original_trajectory)
randomization_points = int((randomization_percentage / 100) * total_points)

# Initialize an empty list for strategic points
strategic_points = []

# Identify strategic points within the trajectory
for point_index in range(0, total_points, total_points // randomization_points):
    strategic_points.append(original_trajectory[point_index])

# Randomization process
for strategic_point in strategic_points:
    start_point = x_coordinate(strategic_point)
    end_point = x_coordinate(strategic_point + 9)
    midpoint = (end_point - start_point) / 2 + start_point
    random_offset = generate_random_offset(0, 100)
    control_point = midpoint + random_offset
    new_points = BezierCurve(start_point, control_point, end_point)
    update_trajectory(original_trajectory, start_point, end_point, new_points)

# Repeat randomization until the desired percentage is achieved

# Output modified trajectory
modified_trajectory = original_trajectory
```

Figure 57. The pseudocode of RanReBot algorithm using Bezier curve function.

This approach (i.e., our reasoning and experimental observations) is visually represented in Figures 58 and 59.

In Figures 58.a.1) and 58.b.1), we can observe that manipulating solely the x-coordinates introduces a notable degree of randomness into the trajectory. Additional alterations along the y-axis have the potential to result in trajectories significantly distinct from the original (Figures 58.a.2) and 58.b.2).

In Figures 59.a.1) and 59.b.1), we can see the appearance of an original trajectory generated by a human (the attacker) and one of its respective replay trajectories generated by RanReBot. It's evident from these figures that RanReBot's replay trajectory is quite similar to the original human trajectory while also being sufficiently distinguishable. On the other hand, in Figures 59.a.2) and 59.b.2), we witness the effects of increased randomization achieved through alterations in both x- and y-coordinates. In these cases, the RanReBot trajectory becomes highly erratic and deviate from representing human-like behavior.

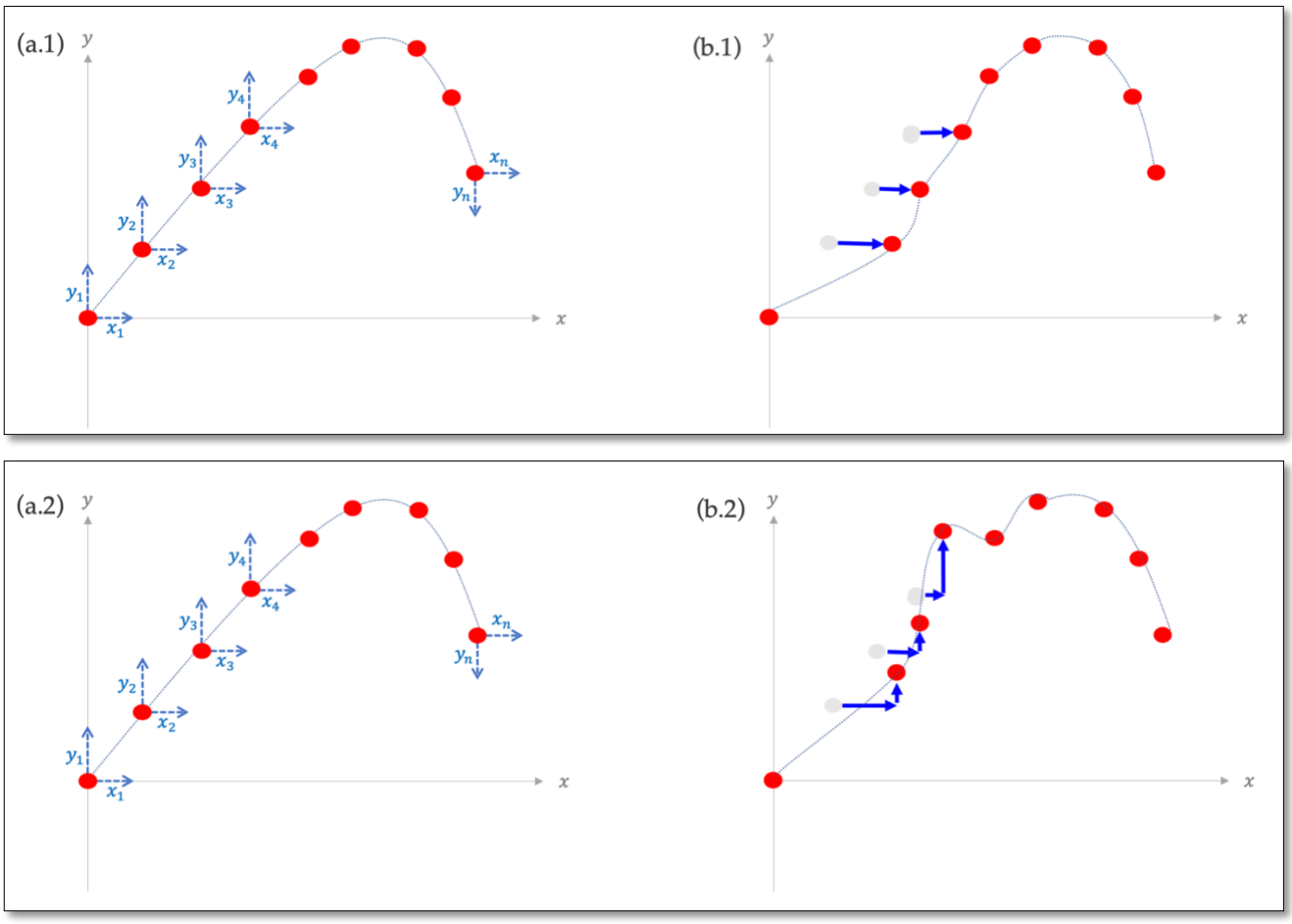


Figure 58. a.1) Original human trajectory, b.1) one respective RanReBot trajectory obtained through exclusive x-coordinate manipulation, a.2) Original human trajectory, b.2) one respective RanReBot trajectory with both x- and y-coordinate manipulation.

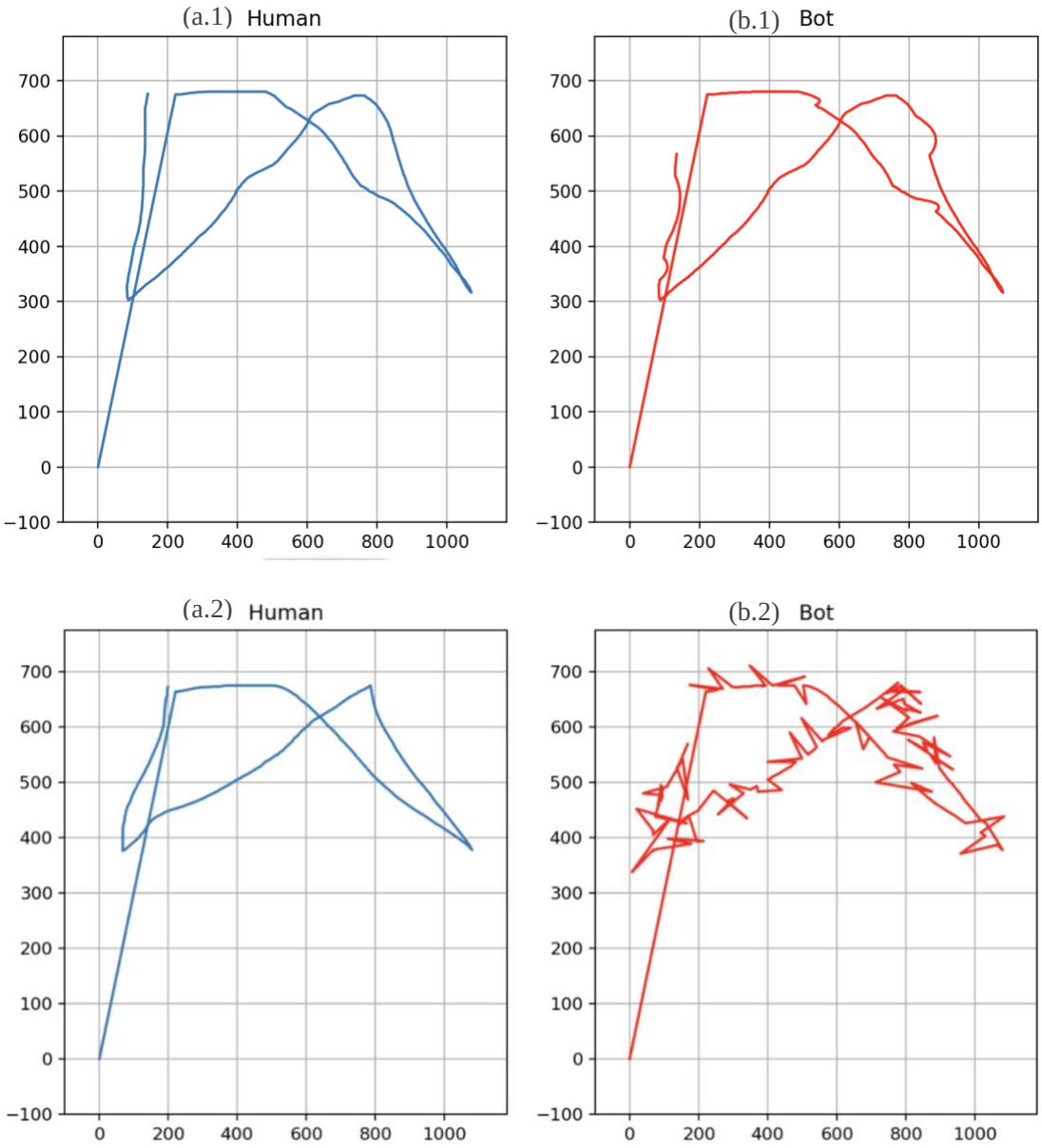


Figure 59. a.1) Original human trajectory, b.1) one respective RanReBot trajectory obtained through exclusive x-coordinate manipulation, a.2) Original human trajectory, b.2) one respective RanReBot trajectory with both x- and y-coordinate manipulation[82].

[82] This figure illustrates a 20% elevation in randomization, providing a clear depiction of our decision to specifically randomize the x-coordinate of a mouse trajectory for 30% of its entire path.

8.4 Randomized ReBot (RanReBot): Performance Evaluation

The RanReBot algorithm is integrated into the framework of the ReBot Replay module. As elaborated in Section 6.3, the primary objective of the ReBot Replay module is to replicate mouse movement actions previously recorded from users and stored within the Tick.txt file. When ReBot is initiated in Replay mode, the RanReBot algorithm assumes a critical role, 'positioning' itself between the initially/originally logged actions stored in the Tick.txt file and their subsequent reenactment during the replay process.

To evaluate the effectiveness of the RanReBot in replaying real human sessions with randomization, we have again conducted experiments on our webpage human-likebots.com following the same procedure as described in Section 6.4.

For instance, in Figure 60.b), we depict the actual DTW distances between the two trajectories previously shown in Figure 59, as discussed in the previous section. These distances are distinctly non-zero. In Figure 60.a), we provided a clear illustration of an original trajectory created by a human (i.e., the attacker) and one of its corresponding replay trajectories generated by RanReBot. This figure highlights the remarkable similarity between the RanReBot-generated trajectory and the original human trajectory, while still featuring noticeable distinctions.

(a) Human & RanReBot (b) DTW Cost Metric Distance Calculation & RanReBot

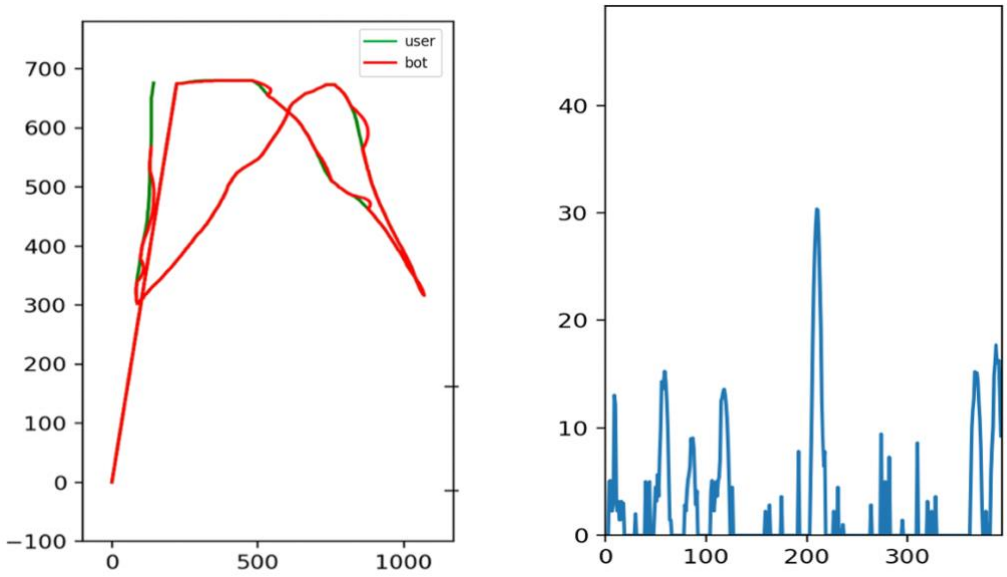


Figure 60. a) Human and RanReBot Trajectories, b) DTW cost metric distance calculation.

8.5 Utilization of RanReBot to Synthesize ReMouse2 Dataset

As explained in the introduction of this chapter, the creation of RanReBot session-replay tool was just an intermediate step towards our ultimate goal, which is the development of a robust advanced session-replay bot detection and defense system. The first concrete effort towards the accomplishment of this ultimate goal was to evaluate the performance of our previously developed bot detection system (ReDetector), which was presented in Chapter 7.

To meet this specific requirement, we generated a new version of the ReMouse2 dataset called ReMouse2.1 dataset, which is a dataset synthesized from ReMouse2 using RanReBot. In essence, we assumed the role of a potential attacker, intermittently visiting and interacting with the website human-likebots.com on a total of 10 occasions. We followed the same instructions that were given to the MTurk users when collecting the ReMouse2 dataset. Each of these 10 sessions was recorded by RanReBot, and subsequently replayed three times. As a result, ReMouse2.1 includes authentic human sessions from 100 MTurk users, 10 hacker sessions, and 30 replayed sessions generated using RanReBot. (For detailed information on how these replayed sessions are derived from the hacker sessions, please refer to Section 6.3.)

8.6 Exploration of ReMouse2.1 Using ReBoDetector

To assess the performance of our introduced session-replay bot attack, RanReBot, we employed our previously developed detection model, ReBotDetector (described in Section 7.3.2). This model is specifically designed to detect original ReBot sessions – replay sessions with no randomization. As explained in Section 7.3.2.1, ReBotDetector utilizes an LSTM feature extractor to capture the most significant features from the mouse-dynamic data and employs the Cosine similarity technique to identify sessions closely resembling genuine user sessions. Additionally, our detection model incorporates t-SNE visualization to facilitate visual identification of sessions produced by replay bots.

We trained ReBotDetector using the ReMouse2.1 dataset, which encompassed data from a total of 140 users (100 humans, 10 hackers and 30 RanReBot instances). Given that each human user produced multiple sessions, the overall number of sessions in the dataset was 1469, with each session characterized by 7 distinct features (refer to Section 7.3.2.1 for more details).

To ensure uniformity in session lengths, we applied padding to these sequences (also explained in Section 7.3.2.1). The LSTM-encoder was trained with a batch size of 128 and underwent 27 training epochs. We employed the Adam optimizer with a learning rate of 0.001, and the LSTM's hidden state had 64 dimensions. For the t-SNE algorithm, we set the perplexity value to 50, the learning rate to 200, the number of iterations to 1200, and used the Manhattan distance metric.

Figure 61 illustrates the distributions of sessions/points within the ReMouse2.1 dataset. The figure distinctly demonstrates that the replayed sessions generated by the RanReBot no longer exhibit 'overlap' with any of the original sessions they were derived from. That is, in terms of similarity assessment, our results show that NO two session instances exhibited 100% similarity match in the entire ReMouse2.1 dataset. From the defenders perspective, this unfortunately implies that the RanReBot attack successfully evaded the ReBotDetector system, while this same system had previously detected the ReBot attack with 100% precision.

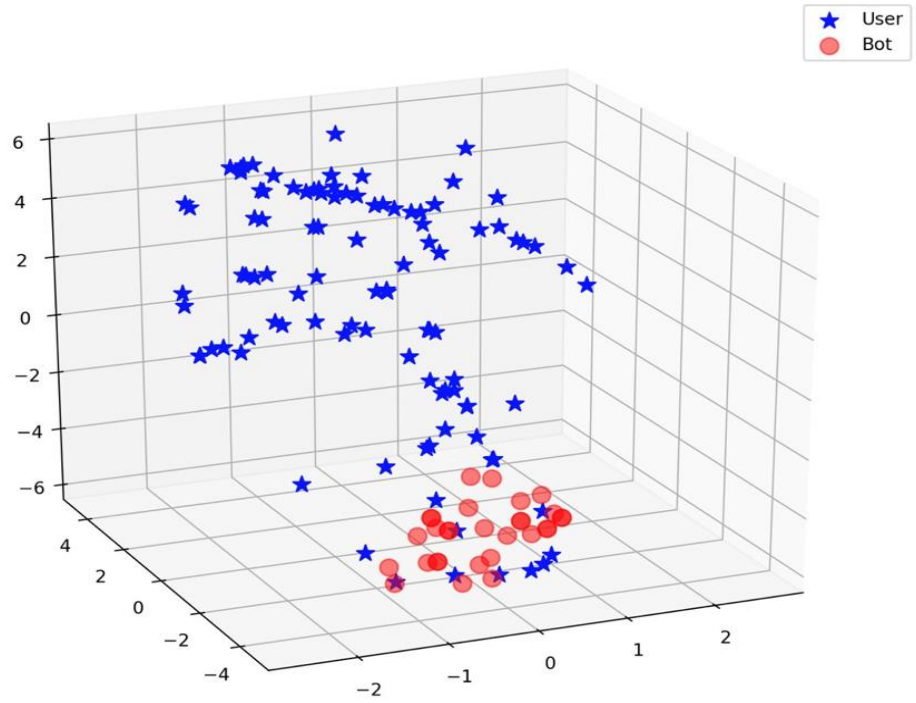


Figure 61. Visual exploration of ReMouse2.1 dataset – LSTM-based feature extraction & t-SNE.

In the next chapter, we will shift our focus from the attacker's back to the defender's perspective and introduce our further refinements of ReBotDetector system so as to make it capable of identifying RanReBot sessions.

8.7 Conclusion

In this chapter, we introduced RanReBot, a randomized session-replay bot capable of recording human browsing activities on the web and replaying them while incorporating controlled randomization. Specifically, RanReBot employs Bezier curves to generate trajectories that are similar to the ones originally created/recorded by RanReBot's operator but at the same time are sufficiently unique (i.e., authentic) and therefore more challenging to detect. We then presented empirical evidence showcasing the effectiveness of RanReBot in evading detection by ReBotDetector (detection model described in Chapter 7), which previously was capable of identifying (simpler/non-randomized) ReBot sessions.

Chapter 9

Advanced Session-replay Bots: Detection

In this chapter, we introduce our innovative approach for detecting advanced session-replay bots that deploy trajectory randomization (RanReBots), which we have previously developed and described in Chapter 8. Our new model, named "RanReBotDetector", is built upon the basic ideas/structure of ReBotDetector (from Chapter 7), but this time deploying the so-called Stacked LSTM architecture. The performance evaluation of RanReBotDetector reveals that our new model achieves an impressive 93% RanReBot detection accuracy within the ReMouse2.1 dataset. As another important contribution of this chapter, we subjected our RanReBotDetector to additional evaluations using a fresh set of synthetic replayed-session data generated using TimeGAN, a Time-series Generative Adversarial Network. The results affirm the robustness of our detection model in effectively detecting advanced session-replay web bots.

9.1 Introduction

In the preceding chapter, we delved into the complex realm of session-replay bots, examining their ability to impersonate human browsing activities on the web. Subsequently, we presented our novel state-of-the-art session-replay bot (RanReBot) that deploys sophisticated trajectory randomization. Our experimental results have shown that this technological advancement in the design of session-replay bot presents a formidable challenge for detection models, as RanReBot (and other trajectory-randomization bots potentially in existence) possess the capacity to elude identification even by advanced ML-based systems.

While tangible evidence of the actual existence of advanced session-replay bot (such as RanReBot) in the real world may currently be limited, our previous discussions have shed light on the continually evolving landscape of AI-powered bot attacks. As we navigate deeper into the sphere of artificial intelligence, it becomes increasingly evident that the emergence of sophisticated bots similar to RanReBot is not a matter of if, but rather when. Clearly, to uphold robust security in the digital ecosystem, it is imperative to anticipate and prepare for these imminent threats.

By assuming a proactive stance and investing in the development of sophisticated detection models, the objective of our research is to establish a resilient security infrastructure capable of withstanding the challenges posed by advanced state-of-the-art session-replay bot attacks (such as RanReBot described in Chapter 8). This foresight ensures that we are not caught off guard when the new threats actually materialize, empowering us to respond effectively and safeguard our digital assets, users, and privacy.

However, it should be pretty obvious that the detection of advanced session-replay web bots such as RanReBot presents a formidable challenge which necessitates innovative new solutions. As already seen in Chapter 7, Long Short-Term Memory (LSTM) models present one possible approach/solution to the given problem due to their excellent performance in modeling and analysis of sequential data. In this chapter, we will immerse ourselves in the implementation of LSTM models tailored for the specific task of session-replay bot detection in the presence of trajectory randomization, harnessing the capabilities of the TensorFlow deep learning library[83].

The upcoming sections of this chapter are structured as follows: Section 9.2 introduces our sequence classification model, RanReBotDetector, which relies on stacked LSTM architecture for detecting randomized session-replay bot attacks. Section 9.3 covers in-depth implementation details of the RanReBotDetector model and presents the results of a comprehensive evaluation of this model's performance. In Section 9.4, we utilize a generative model (TimeGAN) to create 'synthetic' replay sessions from the original human-generated sessions. This synthetic dataset is then used to conduct an additional rigorous assessment of our proposed detection model. Finally, Section 9.5 summarizes the key findings of Chapter 9.

9.2 Framework of Advanced Session-replay Web Bot Detection

Given the compelling evidence presented in Chapter 8, which demonstrated RanReBot's ability to bypass our initial detector model (ReBotDetector) by introducing randomness into replayed human sessions, it has become essential to address the challenge of advanced randomized session-replay bot attacks.

As discussed in Section 7.3.2.1, LSTM models have gained recognition for their effectiveness in modeling and analyzing sequential data [116, 121, 178, 179], and as such appear a

[83] https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM.

highly suitable approach/tool for dealing with the problem of advanced bot detection using sequential trajectory data. In this section, we discuss the modifications made to our ReBotDetector (from section 7.3.2), resulting in the integration of a stacked LSTM architecture for sequence classification. Diverging from the application of an LSTM autoencoder as a feature extractor in the previous ReBotDetector model, this chapter embraces a stacked LSTM architecture that permits direct classification. This transition eliminates the necessity of relying on Cosine similarity for the detection of advanced replay bot sessions. Our primary objective is to augment the precision of the detector in identifying RanReBot sessions within the ReMouse2.1 dataset.

9.2.1 Sequence Classification with LSTM

In order to distinguish between genuine human sessions and RanReBot sessions within the ReMouse2.1 dataset, we have employed a 'stacked LSTM architecture' depicted in Figure 62. This architecture incorporates multiple LSTM layers stacked on top of each other, allowing the model to uncover hierarchical patterns within sequential data[84]. Additionally, the model is comprised of a single hidden layer followed by a final output layer responsible for making critical classification decisions.

The model undergoes end-to-end training, optimizing all of its components, including the stacked LSTM layers and the output layer, simultaneously. The objective of the training process is to minimize a loss function that quantifies the disparities between the model's predictions and the actual labels (user or bot). Lower loss values indicate improved model performance, making it more proficient at distinguishing between the two categories.

[84] For additional details on stacked LSTM networks, please see Appendix D.

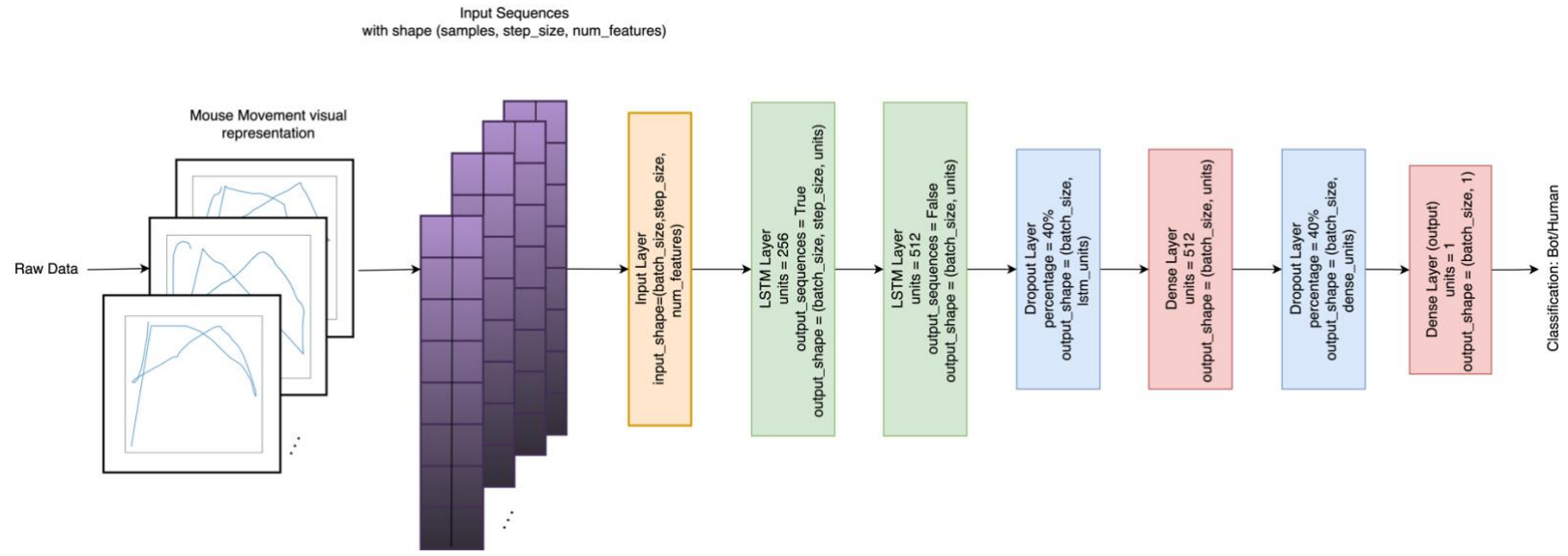


Figure 62. Architecture of the stacked LSTM utilized in RanReBotDetector.

9.2.2 Details of RanReBotDetector Design and Operation

To construct the stacked LSTM-based model for the detection of mouse movement trajectories generated by RanReBot, we have made modifications to the original ReBotDetector procedure originally illustrated in Figure 49 – as shown in Figure 63. This model, referred to as RanReBotDetector (Randomized Session-replay Bot Detector) is designed specifically for the detection of randomized session-replay bot attacks. The framework comprises four essential components: input data, data preprocessing, time-series representation, and the construction of the deep model.

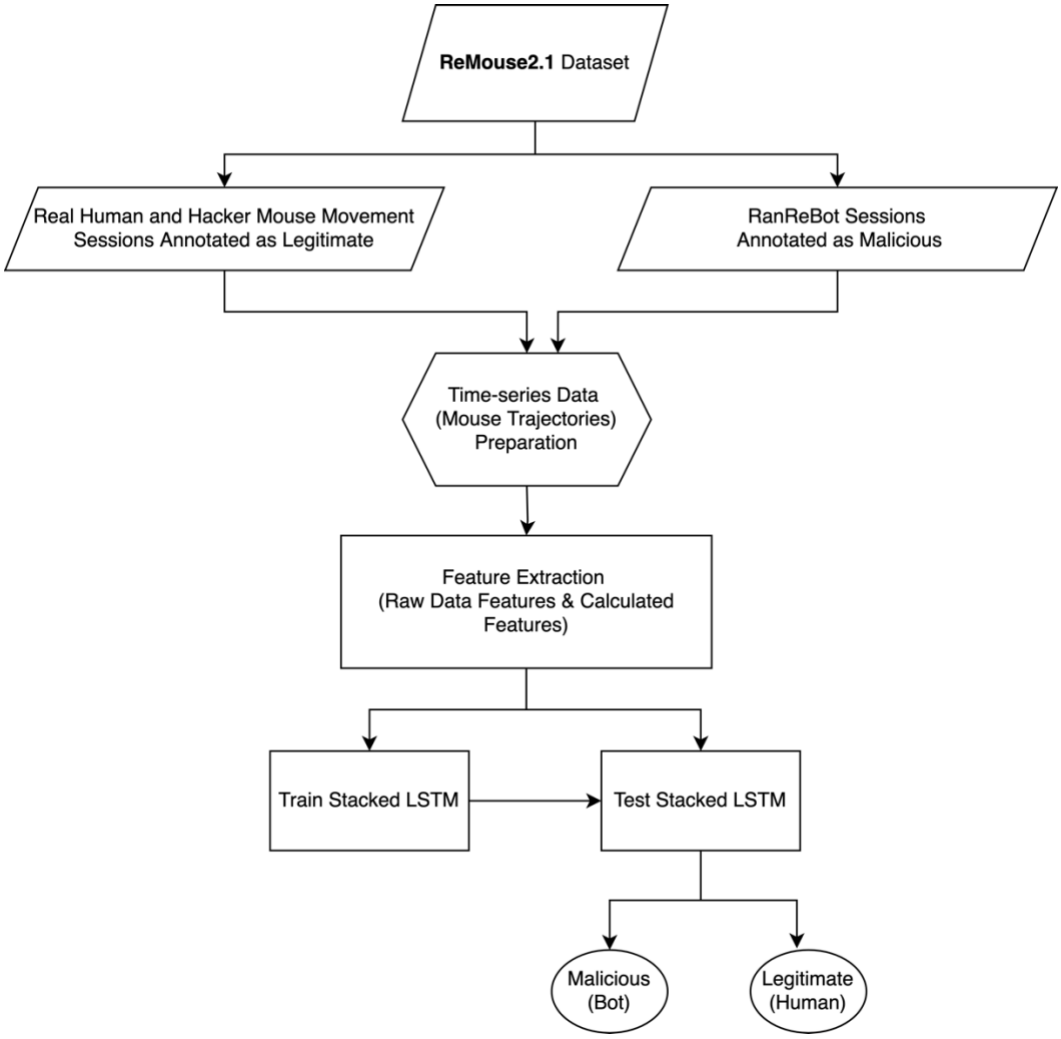


Figure 63. Framework of RanReBotDetector.

We employed a similar approach to train the RanReBotDetector as the one detailed in Section 7.3.2.1 for training the ReBotDetector. It's essential to highlight that before training the LSTM model, we took steps to ensure an even representation of both bot-generated and human sessions in the ReMouse 2.1 dataset. This balancing procedure is crucial as it helps reduce potential biases and enhances our model's training effectiveness. To achieve this, we integrated an additional 60 data points[85] generated by RanReBot (please see Section 8.6 for detailed information about RanReBot) using our in-house software. This resulted in a balanced and synthetically augmented

[85] Each of these 10 hacker's sessions were recorded by RanReBot and they replayed 6 more times.

version of our dataset, known as ReMouse 2.1.1. It's worth noting that our ownership of this software allowed us to generate authentic bot data, a valuable resource for our research. For more dataset details, please refer to Table 14.

After balancing our dataset and before training the LSTM classifier, the input data, which combines 140 human-generated mouse movement data and 90 RanReBot data, each contributing multiple sessions for a total of 1985 sessions, each with 7 distinct features, underwent preprocessing steps. The preprocessing involved data cleaning, normalization, and segmentation into suitable lengths. This process transformed the input data into a 3D representation, as LSTM models require 3D input (refer to Section 7.3.2.1 for preprocessing details).

The LSTM model architecture was structured as a sequential neural network using the Keras API with two layers, containing 256 and 512 units, respectively (see Figure 62[86]).

For model compilation, we used the Adam optimizer with a learning rate of 0.0001 and the binary_crossentropy loss function. To prevent overfitting, we implemented the EarlyStopping callback, which stops training if there's no significant improvement in validation accuracy over 140 epochs and restores the model to its best weights. The model was trained for a total of 200 epochs, with the training process utilizing 66% of the data, while the remaining 33% was reserved for testing.

Table 14. Different versions of ReMouse dataset used throughout this thesis research.

Name of Dataset	No. of Human Data	No. of Bot Data
ReMouse – first dataset collected on "human-likebots.com"	100 MTurk Users	None
ReMouse2 – second dataset collected on **the modified version** of "human-likebots.com"	110 = 100 MTurk Users + 10 Hacker's data (10 distinct iterations of webpage visits by the hacker)	**30 ReBot** data

[86] The data is pre-processed into input sequences defined by the number of samples, temporal sequence length (step_size), and features per time step (num_features). The input layer maintains the sequence's structure, and it is followed by the first LSTM layer with 256 units and 'return_sequences' set to True, preserving temporal dependencies with an output shape of (batch_size, step_size, units). The subsequent LSTM layer with 512 units and 'return_sequences' set to False condenses information into a singular output vector, encapsulating learned temporal features for the classification task (output shape: batch_size, units). To curb overfitting, a dropout layer with a 40% rate is added, randomly nullifying connections during training to enhance generalization. A dense layer with 512 units transforms the LSTM output to a higher-dimensional space, which is crucial for capturing complex relationships in the data. Another dropout layer with a 40% rate reinforces generalization. The final dense layer, using a Sigmoid Activation Function, serves as the classifier, providing probability scores for human and bot classes.

ReMouse 2.1 – a dataset **synthesized from ReMouse2 using RanReBot**	110 = 100 MTurk Users + 10 Hacker's data (10 distinct iterations of webpage visits by the hacker)	**30 RanReBot** data
ReMouse 2.1.1 – a dataset **synthesized from ReMouse2 using RanReBot**	110 = 100 MTurk Users + 10 Hacker's data (10 distinct iterations of webpage visits by the hacker)	**90 RanReBot** data, balanced dataset using actual software

9.3 RanReBotDetector Performance Evaluation

Following the training stage, our RanReBotDetector underwent evaluation on the validation set to gauge its performance. This evaluation encompassed various metrics for assessing the model's effectiveness, including accuracy, precision, recall, and the F1-score. As depicted in Figure 64, our proposed model demonstrated outstanding performance in detecting RanReBot sessions within the ReMouse2.1.1 dataset. Specifically, it achieved an impressive accuracy of 93%, along with a low False Negative Rate (FNR) of 2.5% and a False Positive Rate (FPR) of 9%.

In terms of precision and recall, we observed that for class 0 (genuine users), the model attained a precision of 98% and a recall of 91%. For class 1 (bots), the model achieved a precision of 86% and a recall of 97% (for detailed metrics, please refer to Table 15). These results collectively underscore the high accuracy and robustness of the model in detecting randomized session-replay bot attacks.

It's crucial to note that balancing the dataset played a pivotal role in enhancing the model's detection accuracy. Prior to balancing, the model achieved a respectable accuracy of 86%[87], including relative effectiveness. However, post-balancing the dataset, we observed a substantial boost in accuracy to 93%.

This improvement can be attributed to the dataset balancing practice, which ensures equitable representation of each class. By mitigating class imbalance, this approach reduces the risk of model bias toward one class. Consequently, the model becomes more adept at learning and recognizing the unique patterns and characteristics of each class, resulting in a significant enhancement in accuracy.

[87] Before balancing, the dataset contained 110 human mouse movement records and 30 RanReBot data records, i.e., ReMouse2.1 dataset.

The success of our model underscores the judicious selection of employing the LSTM classifier, as it effectively discerns the critical characteristics and shared features among session-replay bots.

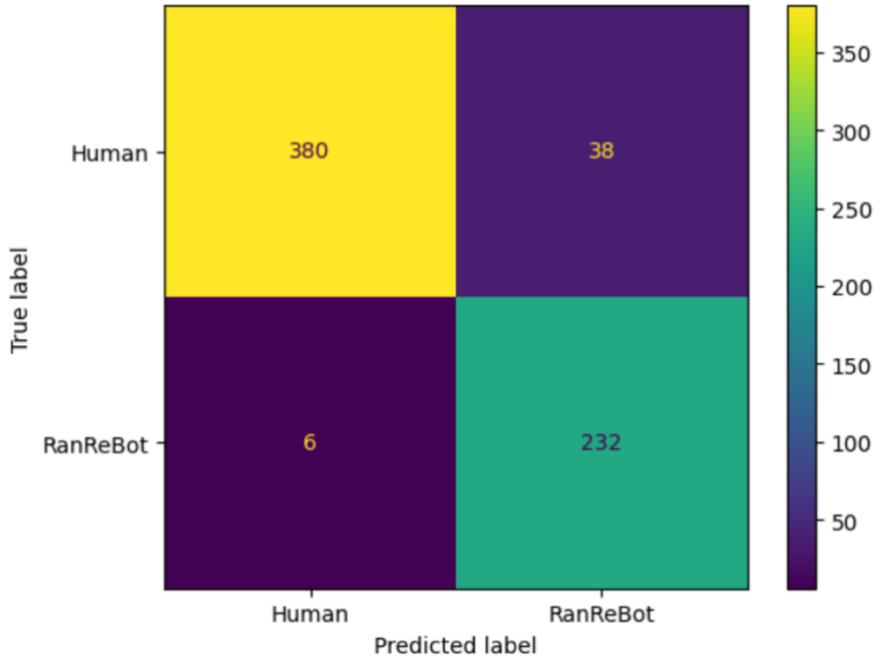


Figure 64. Confusion matrix for 2-class classification, human and RanReBot sessions – ReMouse2.1.1 dataset.

Table 15. Precision, recall, and f-measure for 2-class classification.

	Precision	Recall	F1-score	**Support**
Human (class 0)	0.98	0.91	0.95	418
Bot (class 1)	0.86	0.97	0.91	238
Accuracy			0.93	656

9.4 Creating Synthetic Replay Sessions with TimeGANs

Utilizing our proprietary RanReBot tool, we have been able to produce replay sessions that authentically mimic human interactions without being exact duplicates. However, there are alternative techniques for generating synthetic bot data, including machine learning-based approaches such as Generative Adversarial Networks (GANs) [181]. GANs have proven to be a potent tool for producing synthetic data that closely mimics real-world data.

In this section, we describe our utilization of GANs to create new (bot) replay sessions based on the existing human-generated sessions. This is done with the goal to evaluate the RanReBotDetector's capacity to identify not only the bot replay session that are generated ‘on the fly’ using RanReBot software, but also synthetic bot sessions that are generated ‘off line’ using GANs.

It's important to note that both of our methodologies to generate advanced replay–bot sessions – through the use of RanReBot or the use of GANs – can ultimately be utilized by both attackers and defenders in various scenarios. For instance, consider the scenarios illustrated in Figure 65:

A. Defender's Perspective:

From a defender's standpoint, evaluating a session-replay bot detection system is of paramount importance. The defender can evaluate/test the performance of their detection system by either using RanReBot software or GANs to generate new sessions based on previously recorded session data. Alternatively, either RanReBot software or GANs can be used to balance an existing dataset, if this set is to be deployed for the purposes of training an ML-based detection system.

B. Attacker's Perspective:

Hackers can use software like RanReBot or GAN based solutions to generate mouse trajectories (from a set of pre-recorded sessions) that closely resemble those generated by humans, and then instruct their bots to simply replay/execute these trajectories on the victim server.

In our study, we have taken a dual approach from the attacker's standpoint. Firstly, we designed RanReBot to capture and replay mouse trajectories generated by humans, enabling us to assess the efficacy of our detection model, RanReBotDetector. Additionally (as described in the proceeding sections), we harnessed a generative model named TimeGAN to generate fresh artificial sessions, drawing from our existing data of human-generated sessions.

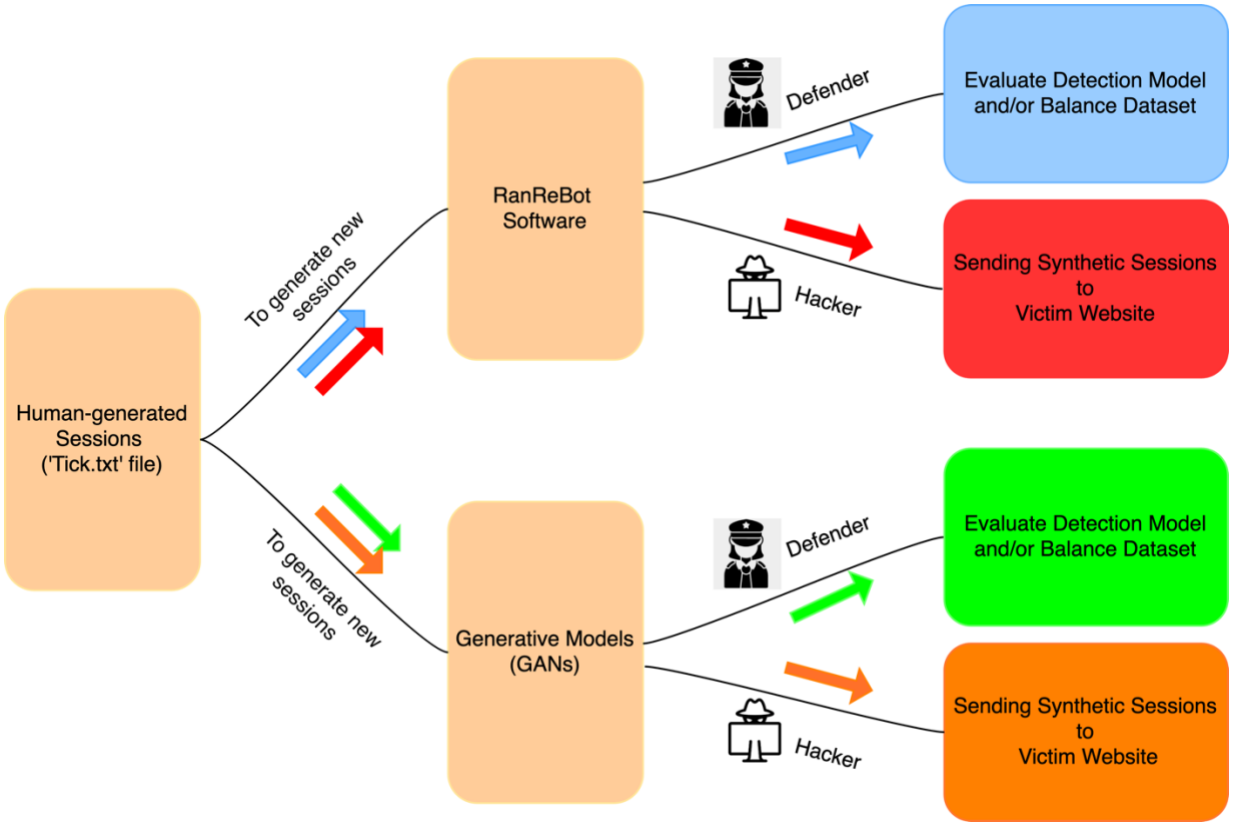


Figure 65. Proposed methods and applications of synthetic replay session generation.

9.4.1 TimeGAN - Concept

Generation using neural networks, specifically Generative Adversarial Networks (GANs) [181], have emerged as a powerful tool for generating synthetic data that closely resembles real-world data. GANs consist of two neural networks - a generator network that produces synthetic data and a discriminator network that distinguishes between real and synthetic data (see Figure 66). By training the generator and discriminator networks in an adversarial setting, GANs can produce high-quality synthetic data that closely resembles the original data distribution. The ability to generate synthetic data with high accuracy has a wide range of applications, including data augmentation, privacy preservation, and anomaly detection.

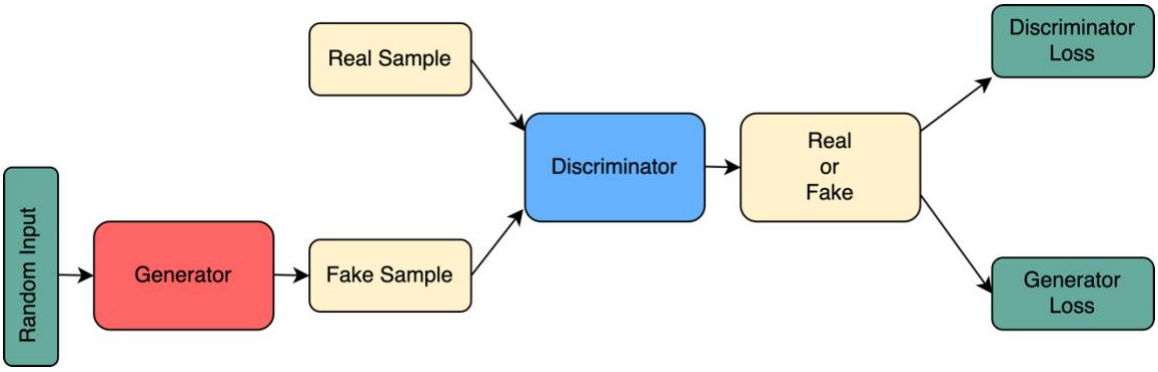


Figure 66. Generative Adversarial Network.

While the majority of popular GAN models focus on generating images, GANs have also demonstrated their capability to generate synthetic time-series data. However, generating synthetic time-series data presents distinct challenges that go beyond those encountered when designing GANs for images. In addition to capturing the distribution of variables at any given moment, such as pixel values or stock prices, a generative model for time-series data must also learn the temporal dynamics that govern the sequential progression of observations [185]. To build a successful generative model for time-series data, it is crucial to capture both the distribution of features at each time point and the relationships among these features over time.

Yoon et al. [183] recently introduced TimeGAN, a novel framework addressing temporal correlations in time-series data. It combines supervised and unsupervised training to learn a time-series embedding space. TimeGAN optimizes both supervised and adversarial objectives, encouraging the model to capture historical data dynamics. It addresses autoregressive characteristics by incorporating unsupervised adversarial loss and supervised loss based on original data, rewarding the model for learning transitions between consecutive time points. The TimeGAN architecture includes Autoencoder and Adversarial Network components for encoding, recovery, and generating sequences. Experiments with various time-series, including historical stock prices, demonstrated TimeGAN's superiority in generating synthetic data.

In the following sections, we leverage TimeGAN to effectively generate synthetic replay sessions from the original human-generated sessions.

9.4.2 TimeGAN – Our Implementation

This sub-section presents a comprehensive overview of our TimeGAN implementation, as it would be deployed by a hacker to generate synthetic bot sessions. Our dataset for this specific research objective consists of sessions recorded by our proprietary software, RanReBot, saved in the Tick.txt file during human/hacker browsing sessions on our experimental website, human-likebots.com. For this study, we collected 30 sessions of hacker mouse movement data, and the data collection process is detailed in Section 6.5.1.

The implementation of TimeGAN was carried out with specific server configurations. We used the Ubuntu 22.04.2 LTS operating system provided by AWS [184], and the instance type was g4dn.xlarge (Tesla T4 GPU). CuDNN version 8.6 and Cuda Toolkit version 11.8 were used to ensure compatibility and efficient GPU utilization.

The initial step in our approach involved data preparation to serve as effective training samples for TimeGAN. Standard preprocessing steps were applied, including transforming raw data into the desired format, filtering out incomplete samples, and scaling features to a defined range. We normalized the width and height coordinates of the data to ensure that all values fell within a standardized range. This involved scaling the coordinates between 0 and 1 based on the maximum and minimum values for both x and y coordinates. To convert categorical variables into numerical representations, we employed one-hot encoding. This process assigns a unique binary value to each category, facilitating effective model interpretation and processing. Additionally, we scaled the time component of the dataset using a Min/Max scaler. This technique ensured that time values were within a consistent range, enabling meaningful comparisons and analyses.

Figure 67 presents a visual representation of our pre-processed dataset used for training in the TimeGAN model. The final dataset represents a refined collection of sessions meeting criteria for data quality and appropriate temporal dynamics. This refined dataset is prepared with the explicit goal of training TimeGAN with noise-free data, recognizing that noise or excessive complexity may impede the model's ability to discern and generate meaningful patterns.

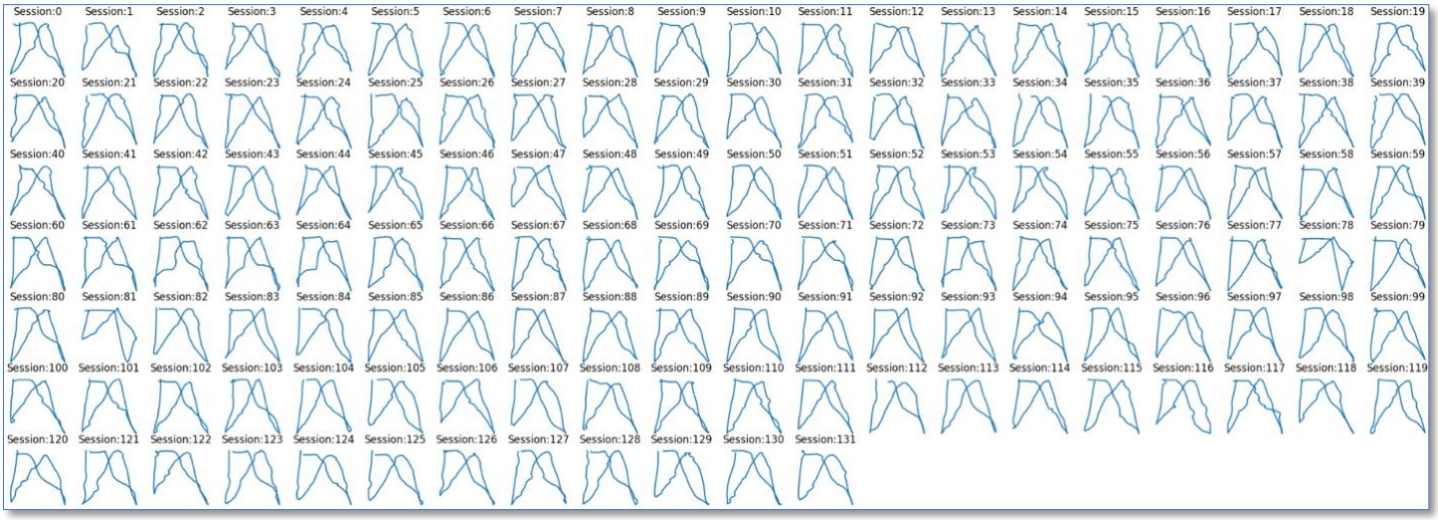


Figure 67. Visual representation of training data for TimeGAN, showcasing a meticulously curated set of sessions that adhere to stringent criteria for data quality and optimal temporal dynamics.

9.4.3 TimeGAN - Training

The TimeGAN model underwent training using our preprocessed dataset, encompassing 132 sessions. Remarkably, the utilization of tabular data allowed us to complete approximately 20,000 iterations in an accelerated timeframe of 20-22 hours, in stark contrast to traditional GAN models that often demand weeks of training.

Figures 68 and 69 illustrate the TimeGAN architecture, including its input and output data structure. Figure 68 portrays the generator, comprising three layers of Long Short-Term Memory (LSTM) units, initialized with a uniform distribution. A fully connected output layer with 128 units per time step follows the last recurrent layer. Our experiments demonstrated that employing three LSTM layers and an additional fully connected output layer yielded superior results.

Similarly, the discriminator, presented in Figure 69, consists of three layers of LSTM units, followed by an output layer comprising 128 LSTM units for the final classification. The output of the last hidden recurrent layer directly feeds into the discriminator's output layer. All LSTM layers maintain an input shape of n by m, with n representing the number of time-steps and m signifying the number of features. The discriminator outputs a value $y \in [0, 1]$.

To optimize TimeGAN's performance, we meticulously tuned several critical parameters, including hidden dimension, gamma, noise dimension, layer dimension, batch size, learning rate, and the discriminator loss on training data. Systematically adjusting these parameters resulted in

an overall performance and accuracy boost, enhancing outcomes in both training and generation tasks. Figure 70 showcases 34 samples of generated synthetic sessions.

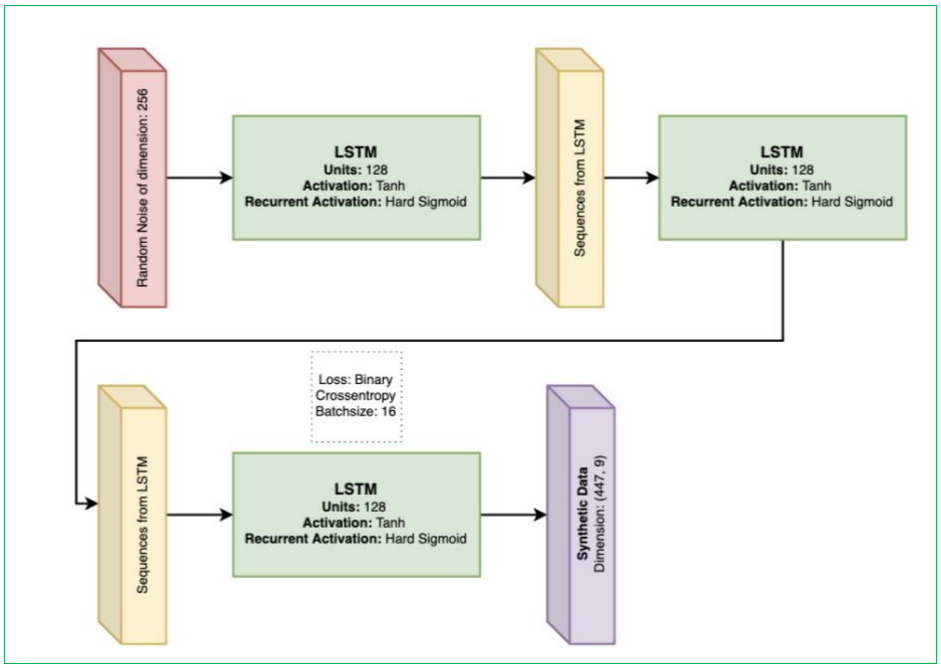


Figure 68. Generator.

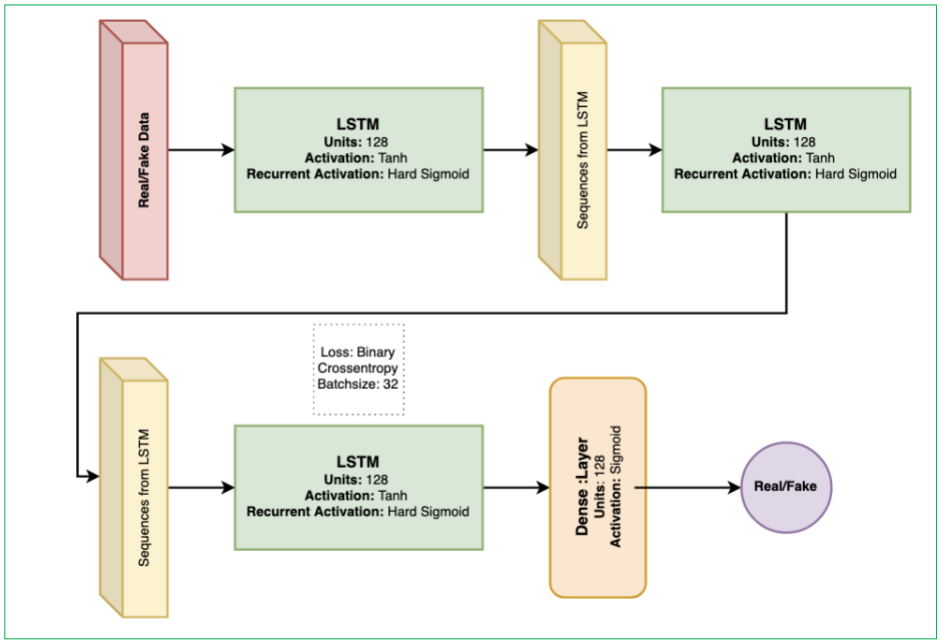


Figure 69. Discriminator.

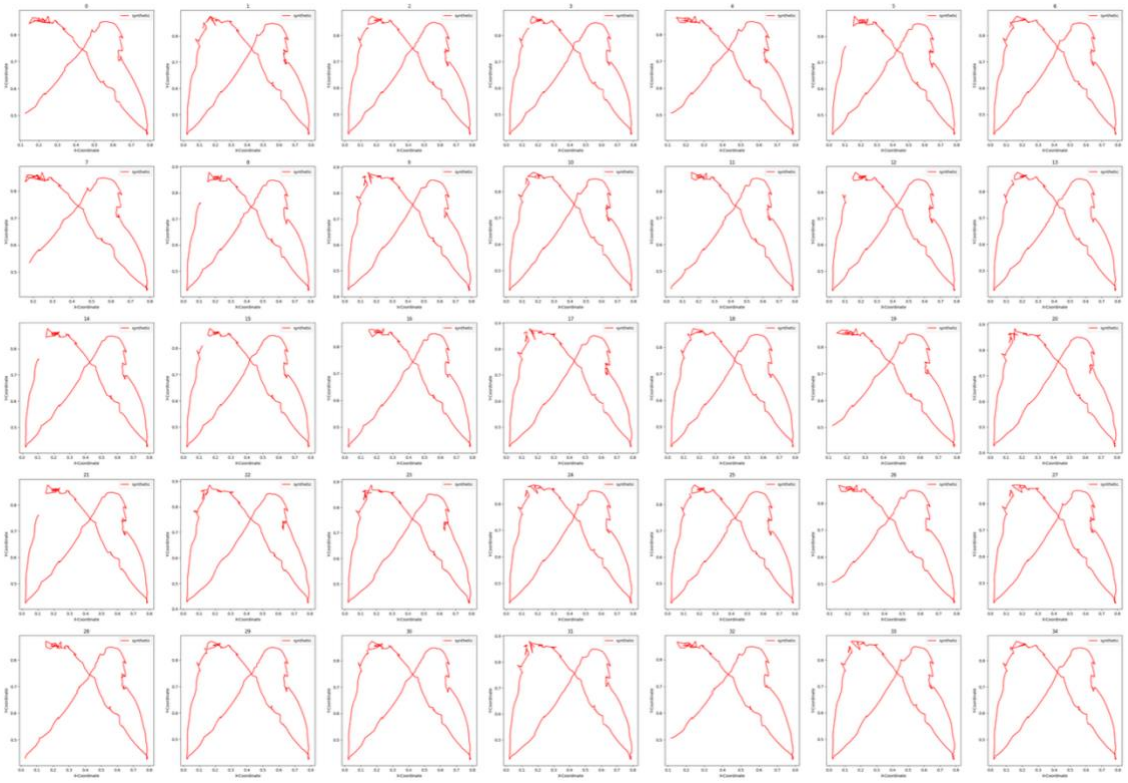


Figure 70. 34 displays samples of synthetic replayed sessions generated by TimeGAN, demonstrating a pronounced visual similarity with the original input data presented in Figure 67.

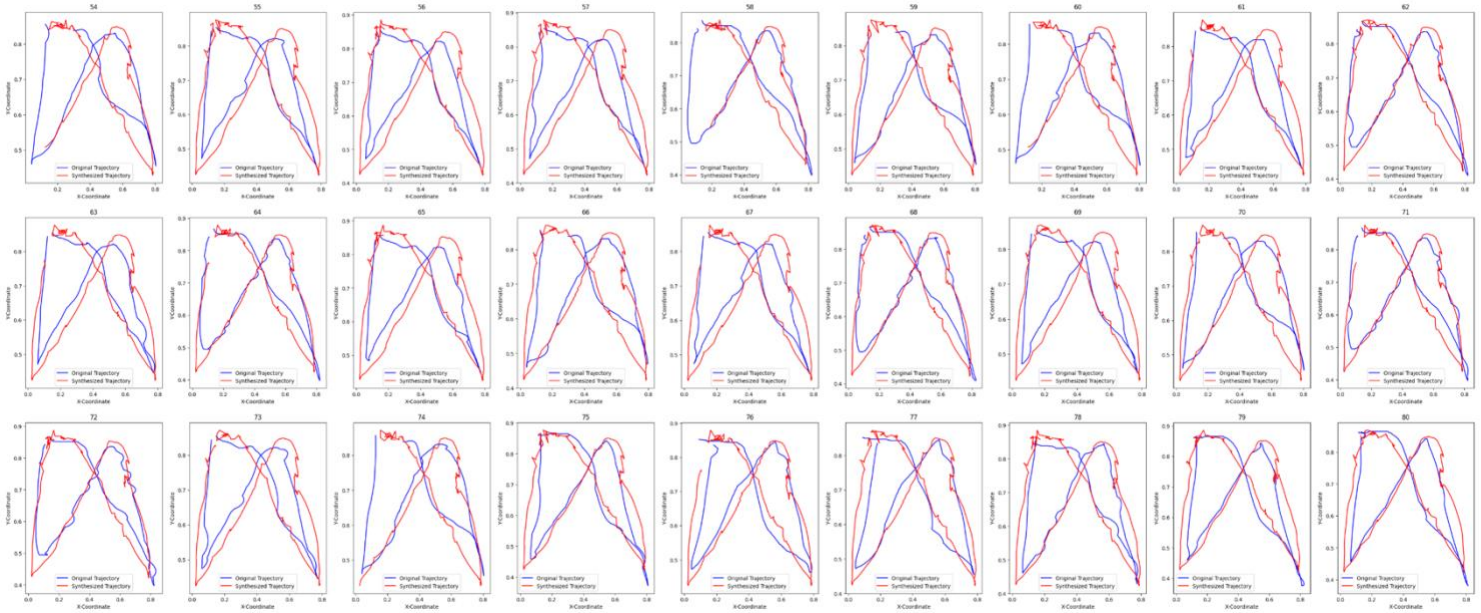


Figure 71. Visual comparison of original hacker's sessions and synthesized replay sessions by TimeGAN.

9.4.4 TimeGAN – Performance Evaluation

To assess the quality of the generated data, we employed established methodologies commonly utilized for evaluating synthetic datasets [183]. These approaches included visual analysis techniques, particularly 't-SNE' analyses, applied to both the original and synthetic datasets. This analysis aimed to qualitatively assess the similarity in the distribution of generated samples compared to the original data within a lower-dimensional space. In addition to qualitative assessments, we utilized a quantitative metric known as the 'Discriminative Score' to measure similarity. The visual exploration involved generating plots and employing t-SNE analyses on 30 original human-generated sessions and corresponding synthetic replay sessions generated by TimeGAN. Figure 71 shows a close overlap between the trajectories of original and generated data, indicating acceptable synchronization. TimeGAN demonstrated the ability to generate distinct yet closely synchronized trajectories, simulating human interactions effectively, albeit not as precise replicas. Figure 72, through t-SNE visualization, further confirmed a degree of similarity between the distributions of original and generated samples.

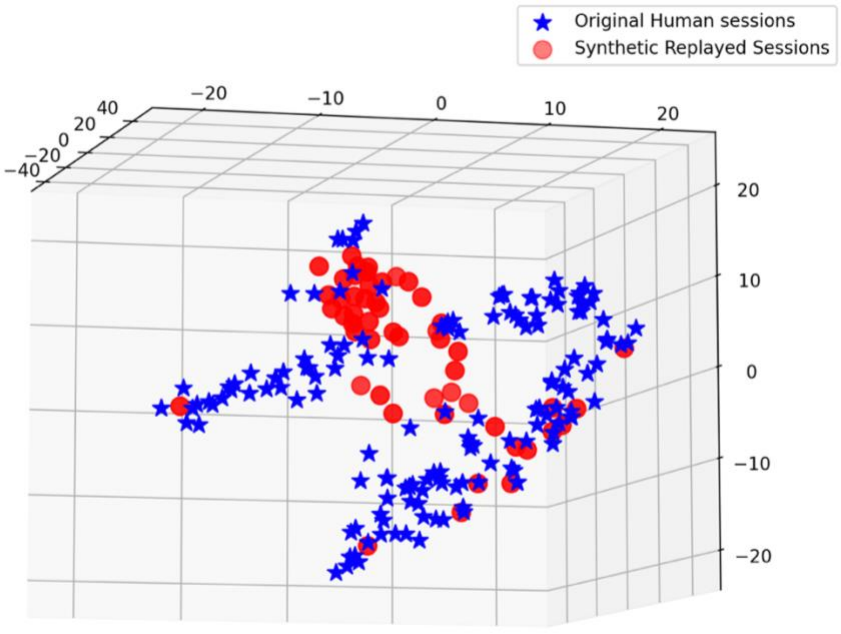


Figure 72. t-SNE visualization of original human-generated sessions and synthetic replay sessions generated by TimeGAN.

To quantitatively assess the fidelity of the generated synthetic time-series, we employed a discriminative score. This score is valuable for evaluating the distinguishability between the original and synthetic sessions, determining if they can be differentiated or are indistinguishable.

We leveraged our developed time-series classifier, the LSTM-based model introduced in Section 7.3.2[88], to conduct this assessment. This classifier plays a crucial role in evaluating the distinguishability between the human-generated sessions and synthetic replay sessions generated by TimeGAN.

To ensure the effectiveness of our model, we adopted a rigorous training and evaluation process. The dataset, comprising 60 samples, encompassing 30 hacker's mouse movement data and 30 synthetic replay sessions generated by TimeGAN (476 in total), was thoughtfully split into training and testing sets. This division, with 67% of the data allocated for training and 33% for testing, was implemented to enable the model to generalize to new data effectively. Following training, a critical evaluation step was conducted on the validation set to gauge the model's performance comprehensively. This evaluation involved the calculation of key metrics, including accuracy, precision, recall, and F1-score.

Figure 73 showcases the results of this classification task, serving as a pivotal means for assessing the distinguishability between human-generated sessions and synthetic replay sessions created by TimeGAN. The 100% accuracy in detecting these bot data indicates that the synthetic sessions are discernible from the original sessions. Despite the general efficacy of GAN models in data generation, this result highlights TimeGAN's unsatisfactory performance in generating synthetic replay-bot sessions, as they can be easily identified using RanReBotDetector.

[88] The LSTM model architecture was specified as a sequential neural network model using the Keras API with two layers, consisting of 256 and 512 units, respectively. The Adam optimizer with a learning rate of 0.0001 was used for compiling the model, and the binary_crossentropy loss function was used. To prevent overfitting, the EarlyStopping callback was implemented, which stops training the model early when there is no significant improvement in validation accuracy over 140 epochs and restores the best weights of the model. The model was trained for 200 epochs.

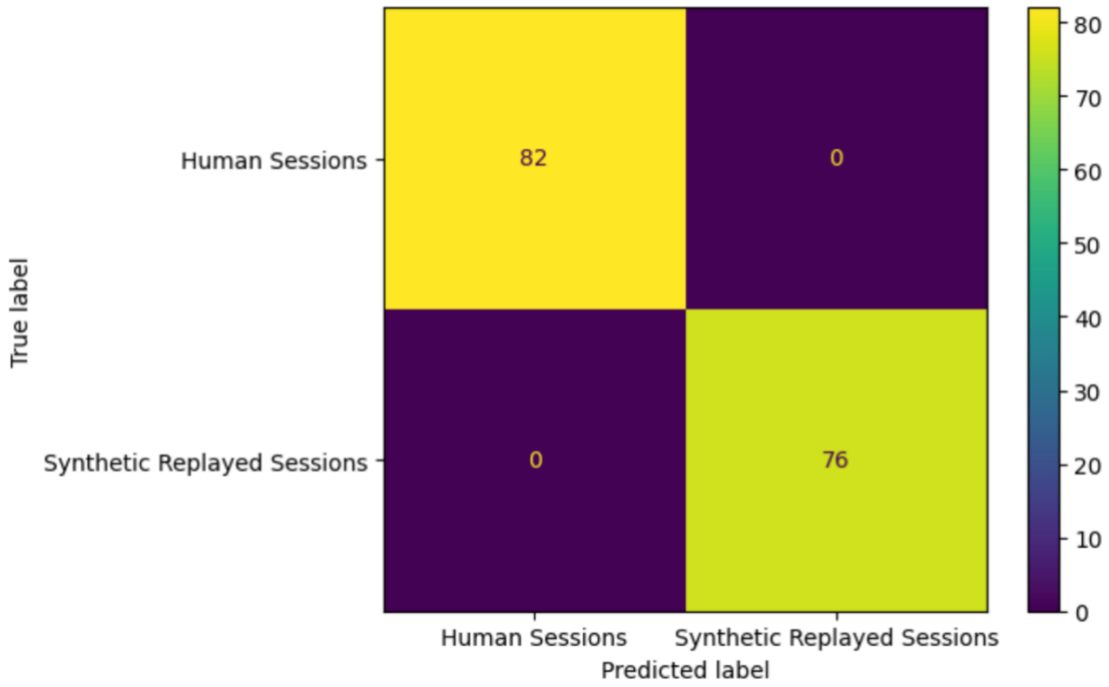


Figure 73. Confusion matrix for 2-class classification, original and synthetic sessions generated by TimeGAN.

9.4.5 Evaluating the Performance of RanReBotDetector Using Synthetic Replay Sessions Generated by TimeGAN

To further validate our assertion, we conducted an evaluation of the LSTM classifier's[89] performance on a synthesized dataset using ReMouse2 data named it as "ReMouse2.2." This dataset comprises 110 human-generated mouse movement data, 80 MTurk users, 30 hackers, and 90 synthetic replay sessions generated by TimeGAN (2003 sessions in total). The dataset underwent a division into a 67% training set and a 33% testing set, which was utilized for both model training and evaluation. This assessment aimed to discern disparities between synthetic replay session data and human-generated data.

As illustrated in Figure 74, the model achieved 100% accuracy in detecting synthetic replay sessions in the ReMouse2.2 dataset. This outcome aligns with expectations, given the clear distinguishability between synthetic replayed sessions generated by TimeGAN and the original

[89] Refer to Section 7.3.2 for more details.

sessions. Importantly, these results underscore the robustness of our detection model in identifying session-replay bot attacks across different levels of evasiveness.

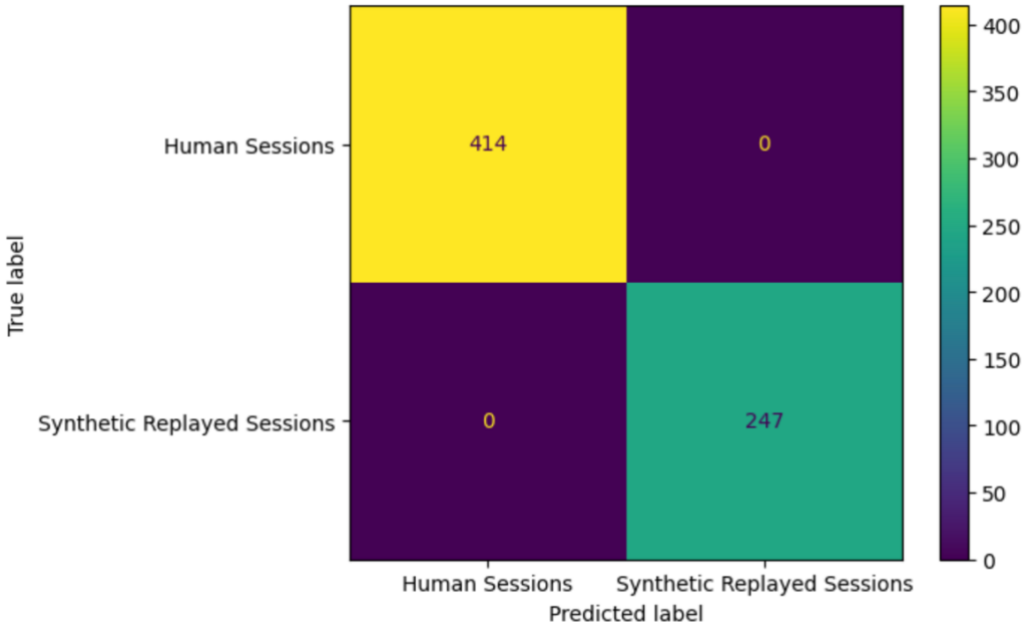


Figure 74. Confusion matrix for 2-class classification, human-generated session and synthetic replay session generated by TimeGAN – ReMouse2.2 dataset.

9.5 Conclusion & Discussion

This chapter has provided an extensive examination of the intricate challenge involved in detecting advanced session-replay bot attacks, introducing our innovative detection model, the RanReBotDetector. Leveraging stacked LSTM-based models for sequence classification, our approach addresses the complex issue posed by randomized/advanced session-replay bot attacks.

Through the rigorous evaluation of our model using RanReBot data within the balanced dataset, ReMouse 2.1.1, we achieved an impressive accuracy rate of 93%. More notably, we conducted tests on our model's ability to detect synthetic replay sessions generated by TimeGAN, responding to the challenge of employing generative models in simulating advanced session-replay bots. In this evaluation, our proposed model demonstrated exceptional performance, achieving a 100% accuracy rate.

These findings very evidently demonstrate that replay-bot sessions, regardless of their degree of randomization, retain fundamental characteristics inherited from their original sessions, and thus cannot escape detection by our RanReBotDetector.

Considering the potential threat posed by malicious actors developing advanced replay bots with insights into machine learning models, the next chapter introduces our final contributions towards the development of web bot detection techniques/systems that are robust against AI powered bots. This approach leverages the concept of webpage randomization so as to introduce additional complexity into the adversary's task of replicating genuine human user sessions.

Chapter 10

RanABD: Webpage Randomization for More Effective Session-Replay Bot Detection

In this chapter, we propose RanABD, a novel front-end webpage randomization technique that aims to support more effective detection of session-replay bots. By building on some general ideas of Moving Target Defence (MTD), RanABD performs continuous randomized micro modifications in the spatial alignment of select visual HTML elements and element attributes in the target webpage, while causing minimal disturbances in the page's overall appearance and functionality. This implies that no two visitations of this webpage - either by the same user or two different users - will encounter the same/identical spatial alignment of the page's elements. By doing so, the technique ensures that the distance between trajectories of any two genuine human-visitors, or trajectories of repeat visits by the same human user, are substantially greater than the distance between legitimate-users trajectories on a page with the same/fixed alignment. This, consequently, facilitates more effective detection of session-replay bots, as their trajectories tend to exhibit a greater degree of similarity (with the original 'parent' trajectory and with one another). For session-replay bot operators, the only way to bypass this defence is by increasing the degree of randomization in replay sessions, so as to increase the distance between their respective trajectories thus make them less 'obvious'. But, this approach is likely to backfire as: a) randomized replay trajectories may be in complete disagreement with the current/actual alignment of the retrieved webpage, and/or b) too much randomization may results in outlier-like trajectories that are even easier to detect.

10.1 Introduction

As discussed earlier in Chapter 8, randomization refers to the introduction of randomness or unpredictability into a system or process. In the domain of web bot design, and with the increasing

sophistication of AI synthesis programs, adversaries are now able to generate more human-like behaving bots by introducing carefully engineered randomization. Specifically, in the context of a session-replay web bot attack, where genuine human browsing sessions are recorded and replayed to fully emulate human behavior on websites or apps, randomization can be utilized to intelligently change/modify the trajectory of bot sessions. That way, the replay sessions appear as more authentic human-user sessions, without being the exact replicas of the original human-generated session. With these advances in bot design, bot-conducted attacks are becoming increasingly more difficult to detect and prevent. However, one area that can still be capitalized on in order to successfully distinguish between sessions of genuine human visitors and 'randomized' session-replay bots is the human's ability to understand the context and then adequately interact with the visual layout of a given webpage. The objective of this chapter is to develop a technique that can capitalize on this human-specific cognitive ability.

In this chapter, we introduce our novel RanABD technique that can facilitate successfully detection of all simple types of session-replay bots as well as the new era of AI-generated session-replay bots. Our technique introduces 'controlled' randomness and unpredictability into the structure of the webpage targeted by bots, thereby increasing the chances that the bots' replayed trajectories are (i.e., end up) being significantly statistically different from those produced by human visitors.

In terms of the existing relevant literature, the only published work that draws some parallels with our newly introduced technique is [185], in which the authors have proposed to implement randomization of invisible DOM elements in a webpage in order to counteract ad-blocker browser extensions. And while the technique from [185] aims and ensures that the visual presentation of the target page remains entirely unchanged, our technique does quite the opposite. Namely, the objective of RanABD is to introduce persistent changes in the position/layout of visual elements in the target webpage, which are small enough not to distort the overall look and functionality of the page but significant enough to cause the genuine human users (who are able to acquire full visual context and semantic understanding of the given page) to produce sufficiently (i.e., detectably) different mouse trajectories and/or a different sequence of mouse events.

The chapter is structured as follows: Section 2 presents the problem statement and rationale behind the development of the RanABD technique. Section 3 details the methodology employed to implement the RanABD technique. In Section 4, we provide a summary of our approach and

present preliminary findings obtained from applying RanABD in an experimental webpage. The effectiveness of RanABD in detecting RanReBot attacks is evaluated in Section 5. Finally, Section 6 concludes the chapter by presenting our key findings and outlining potential avenues for future research.

10.2 Problem Statement

Interactions of different human users with the same webpage may appear indistinguishable to the naked eye, but when measured by a behavioral algorithm they, without exception, exhibit unique characteristics. This is largely due to the fact that the manner in which individuals hold, swipe, or type on their devices, including their personal style of mouse usage, generally varies among users.

In Chapter 5, we have formally demonstrated that web sessions and mouse trajectories produced on the same webpage/site by different human users are guaranteed to be perceptibly different from one another. In Chapter 5, we have also shown that perceptive differences between sessions generated on the same webpage/site are not unique to different users only. Namely, our experimental results have shown that even the same human user, when repetitively interacting with the same webpage, is incapable of producing an entirely identical web session/trajectory twice. In other words, there always exist small but non-negligible pairwise differences between repeat sessions/trajectories generated by any particular human user, as depicted in Figure 75.

In addition, our research form Chapter 5 has shown that trajectories produced by the same genuine human user, even after considering their pairwise deviations, tend to be more similar to each other than to those generated by other human users on the same webpage/site. In other words, the original (i.e., very first) and subsequent (i.e., repeat) trajectories of individual human users tend to form small distinguishable ‘micro-clusters’ across the Feature Space, as illustrated in Figure 76.a). It is important to note, though, that despite their relative ‘dispersion’ in the Feature Space, these micro-clusters only occupy a minor portion of the overall Feature Space.

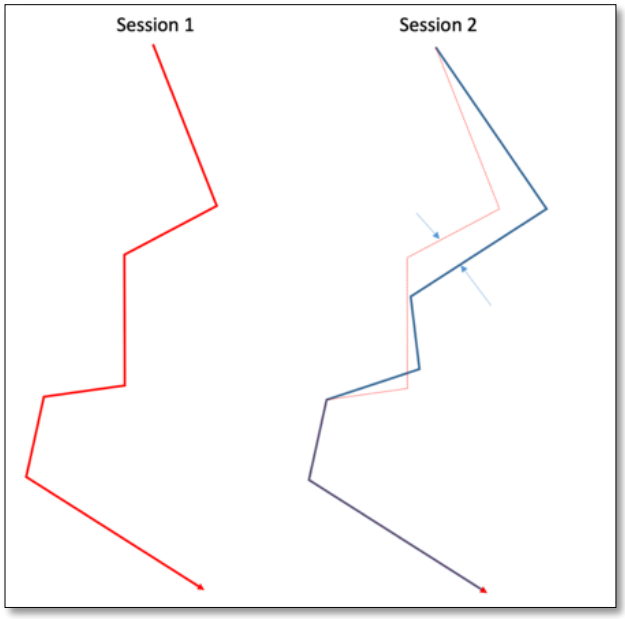
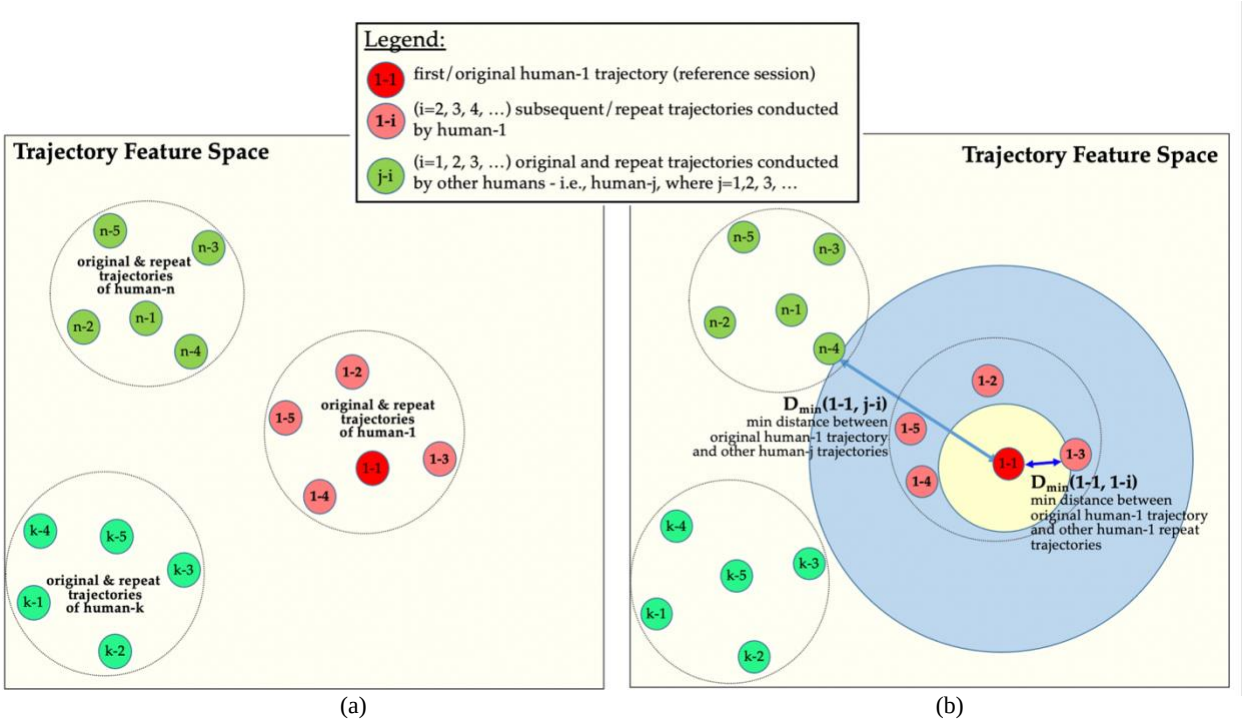


Figure 75. Pair-wise difference between repeat trajectories generated by the same human user on the same (unchanged) webpage.



(a) (b)

Figure 76. a) Micro-clusters formed by original and repeat sessions/trajectories of individual users, b) Distance between repeat sessions/trajectories of the same user vs. session/trajectories of other users.

Figure 76.b) depicts some other important findings from Chapter 5. For example, Figure 76.b) shows that a minimum separation distance between the original (i.e., first) and repeat trajectories of one particular human user (e.g., human-1) generated while visiting the same (unchanged) webpage - we denote this distance as D_{min}(1-1, 1-i, for $i = 2, 3, ...$) - is always smaller than the minimum distance between the original trajectory of this user and any trajectory of any other human users visiting the same webpage (we denote this distance as D_{min}(1-1, j-i) , for $j = 2, 3, ... and\ i = 1, 2, 3, ...$). That is,

$$D_{min}(1\text{-}1,\ 1\text{-}i) < D_{min}(1\text{-}1,\ j\text{-}i) \qquad (1)$$

In Chapter 6, we have shown the operation of a basic/simple session-replay web bot attack is pretty straight-forward – it requires that a single 'genuine human session' be recorded by the hacker on the target webpage/site, and this session is subsequently replayed with no or very minimal modifications and as many times as required by the attack's objective(s). In the context of our analysis, and as illustrated in Figure 77.a), this implies that the originally recorded and subsequently replayed sessions/trajectories of a simple session-replay bot are likely to form a very tightly spaced cluster in the Feature Space used for mapping of individual user sessions on the target web page/site. By a 'tight cluster' we mean a cluster in which the distance between its replay-session points is smaller than the distance between points in repeat-session clusters corresponding to genuine human users. Due to their distinguishable characteristics, these tight-clusters - and thus the presence of sessions-replay bots on the target web page/site - should be relatively easy to detect (refer to Section 7.3).

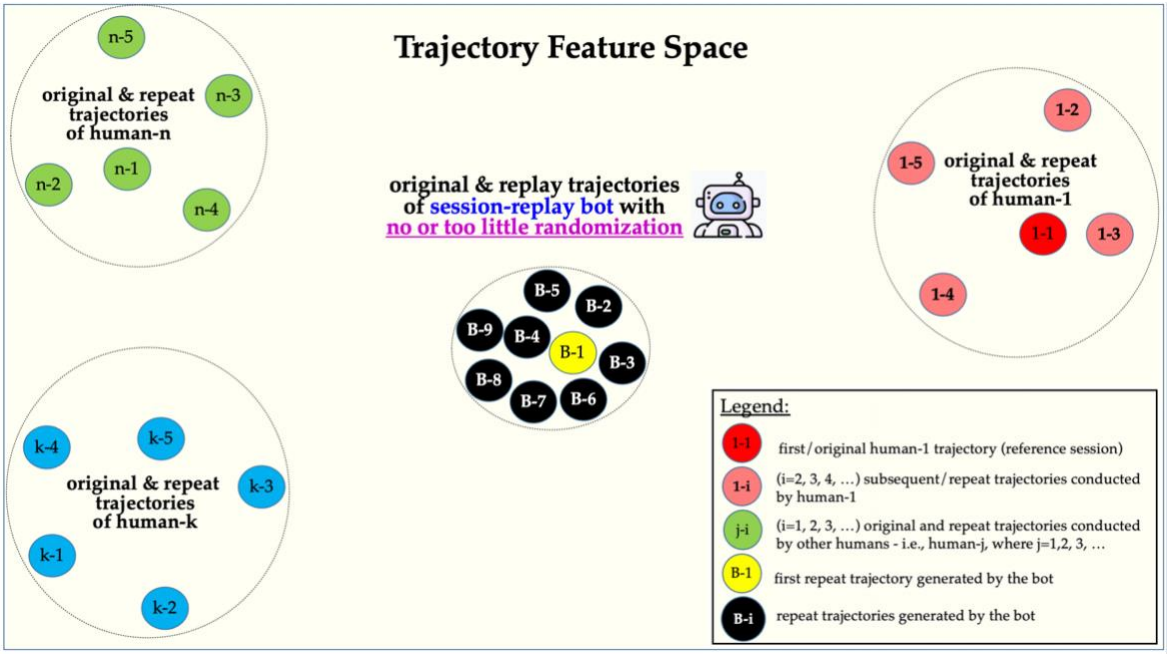


Figure 77. a) Tightly-spaced cluster of repeat-session trajectories generated by a bot with no or little randomization.

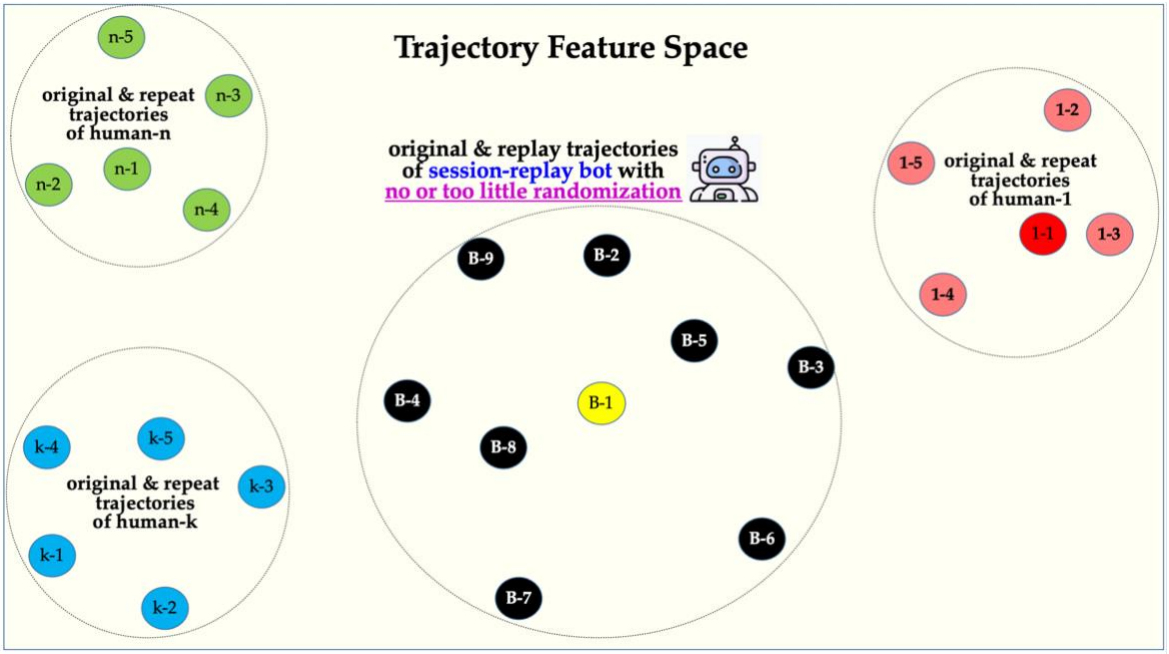


Figure 77. b) Overly spread cluster of repeat-session trajectories generated by a bot with excessive randomization.

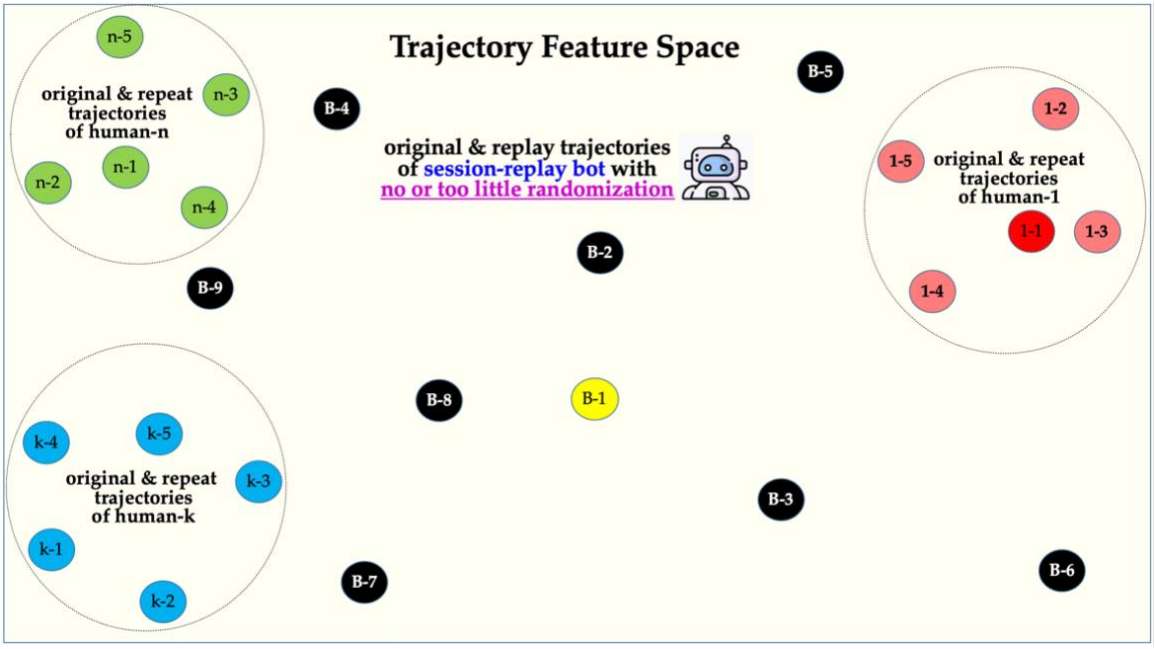


Figure 77. c) Extreme spreading of repeat-session trajectories generated by a bot with extreme randomization.

Clearly, from the attacker's perspective, the strategy of simple session replay is not optimal, as it may lead to a high probability of detection. Therefore, as discussed in Chapter 8, the operators of advanced session-replay bots could resort to the use of 'trajectory randomization' in order to intelligently change/modify the trajectories of replayed sessions, making them appear more like authentic human-user sessions (as was done by RanReBot). In other words, through randomization the attacker could ensure that the spacing among trajectories of individual replay sessions in the Feature Space is such that their respective cluster is more 'spread' and better resembles the clusters of repeat sessions generated by genuine human users, ultimately decreasing the likelihood of detection. However, one major challenge of this strategy for the attacker is how to determine the exact nature and degree of randomization. Namely, too little or too much randomization could result in Feature Space formations that are detectibly different from those corresponding to genuine human sessions.

As previously indicated, with insufficient levels of randomization, the cluster of bot's replay sessions will remain more tightly spaced then the clusters of repeat sessions of genuine human users (see Figure 77.a)). On the other hand, it should be noted that excessive randomization is also problematic for several different reasons: 1) it could cause the bot's replay sessions to form an

unusually large and dispersed, and thus suspicious cluster (see Figure 77.b)); 2) by significantly deviating from the original human (i.e., hacker generated) session some of the bot's replay-sessions may completely fail to achieve the actual objective of bot's operation (e.g., miss to scroll the mouse over or click on a specific area or link in the target webpage, or miss to click on and choose a particular item in a dropdown menu, etc.); 3) extreme randomization could also cause replay-sessions trajectories to map into outlier points far removed from all other points in the Feature Space, potentially triggering intrusion detection alarms (see Figure 77.c)). Given the multitude of potential problems and risks associated with excessive randomization, it seems reasonable to assume that a preferred strategy for the attacker operating a session-replay bot would be to deploy an advanced session/trajectory randomization that is generally small but significant enough to prevent formation of 'tight clusters' in the trajectory feature space.

10.3 RanABD Model: Webpage Randomization for Advanced Bot Detection

In this section, we introduce our novel technique – Webpage **Ran**domization for **A**dvanced **B**ot **D**etection (RanABD) - which aims to successfully detect not only simple but also advanced (randomized) session-replay bots. At its core, this technique itself deploys micro-randomization in the alignment of select visual HTML elements and element attributes in the target webpage. More specifically, the technique ensures that at every webpage retrieval request, the server returns a copy of the page with slightly spatially-shifted (select) visual elements. By doing so, the technique accomplishes two objectives: (a) It achieves additional separation/spreading of trajectory points that correspond to repeat sessions of each genuine human users (refer to Figure 75.a)). This effect occurs due to the fact that at every re-visitation of the target webpage, a genuine human user is now presented with a slightly altered content arrangement/alignment, and thus will inevitably end up generating a different trajectory even while repeating the exact same task or set of steps as in the previous page visitation(s). With greater spreading within clusters corresponding to human repeat sessions/trajectories, a cluster corresponding to sessions/trajectories of a session-replay bot with no or limited level of randomization will be much easier to identify, as also evident from Figure 77.a). (b) For a session replay bot that is programmed to deploy greater levels of randomization (e.g., as an attempt to achieve better separation between the replay sessions/trajectories, and thus make them appear more like repeat sessions/trajectories of a genuine human user), RanABD actually increases

the probability that the replay sessions of this bot ultimately fail in achieving their intended objective. Namely, with greater levels of randomization in the bot's replay trajectories as well as due randomization in the alignment of the target page due to RanABD, it becomes progressively less likely that the mouse cursor of the bot's replay-sessions end up at the right location to (e.g.) execute a click on the right link or choose the right item from a dropdown menu.

It is worth noting that RanABD is designed to align with the concept of "moving target defense" (MTD) [186], which is a proactive defense strategy aimed at imposing a disadvantage on the attacker by introducing randomness, diversity, and dynamism into the targeted system. MTD achieves this by continuously varying system attributes, making it challenging for the attacker to explore and accurately predict the system's state(s), and without those it is hard if not impossible to devise successful attack strategies.

In the case of RanABD, we leverage the principles of MTD to enhance the detection and mitigation of malicious bots, with a specific focus on session-replay bots. By incorporating randomization, diversity, and dynamism into the positioning of HTML elements and element attributes in the target webpage, RanABD disrupts the intended operation of malicious session-replay bots and hinders their ability to accurately replicate human behavior. In other words, the randomization, diversity, and dynamism of RanABD add an extra layer of defense against session-replay bots, which ultimately improve the system's overall ability to spot and block these bots.

10.3.1 RanABD Methodology

RanABD is a front-end server-implemented webpage randomization technique that preserves the overall visual appearance and functionality of the target webpage while introducing spatial micro-shifts to its visual HTML elements and element attributes. That is, after the server prepares/assembles the page content in the usual way, RanABD randomizes the alignment of some select objects on the page before returning it to the user, thereby guaranteeing that no two requests (by the same or two different users) ever receive the exact same page structure/outline.

To ensure that the randomization process does not negatively impact the user's browsing experience, RanABD algorithm is carefully designed to limit the degree of object movement.

Specifically, the spatial shift of individual objects (such as images and text) on the webpage is restricted to a small percentage of the user's viewport (i.e., the user's visible area of the webpage in the browser window) around the object's central coordinates, as depicted in Figure 78.

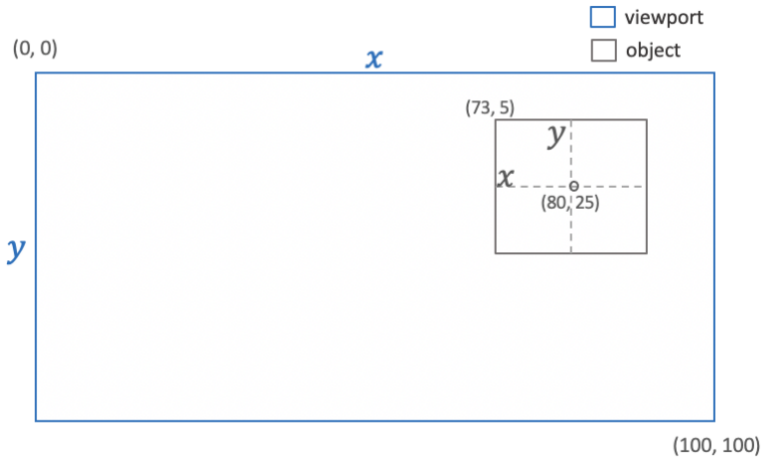


Figure 78. Browser window.

RanABD also ensures that objects which are located close to the edge of the screen are restricted from moving significantly, thereby preventing any potential disruption in the user's overall browsing experience.

A crucial aspect of RanABD algorithm are the following nine variables which ensure adequate randomization of objects' spatial locations in the target webpage. These variables include:

1. **Max x-axis range:** global variable that represents the percentage of the screen width that an object can at most be shifted by along x-axis direction (left or right). In our experimentation this value has been set to 2.
2. **Max y-axis range:** global variable that represents the percentage of the screen height that an object can at most be shifted by along y-axis direction (up or down). In our experimentation this value has been set to 5.
3. **Max positive x-axis shift (object-level control):** the variable that represents the maximum value in the positive x-axis shift, indicating the furthest distance an object can move in the positive direction along the x-axis (rightward). The specific limitations[90] on

[90] In real-world webpage design, various elements are subject to specific limitations in their movements to ensure a cohesive and user-friendly experience. For instance, the footer content is strategically anchored at the bottom of the page with unique restrictions, providing consistent access to essential information. Additionally, resizable text boxes play a crucial role in preserving layout consistency by restricting adjustments in both width and height, preventing issues such as text overflow or compression that could compromise readability. The responsive navigation menu is designed with limitations in both horizontal and vertical movements, guaranteeing adaptability across different screen sizes. Furthermore, advertisement banners embedded in webpages may encounter restrictions on positive y-axis shift to maintain proper placement within the overall design. These limitations collectively contribute to the effective and reliable presentation of web content, ensuring a seamless interaction for users.

the movement vary for each object, in case they have their own unique restrictions regarding this particular direction.

4. **Max positive y-axis shift (object-level control):** the variable that represents the maximum value in the positive y-axis shift, indicating the furthest distance an object can move in the positive direction along the y-axis (upward). The specific limitations on the movement vary for each object, in case they have their own unique restrictions regarding this particular direction.
5. **Max negative x-axis shift (object-level control):** the variable that represents the maximum leftward movement along the x-axis. It refers to the extent to which an object can move in the negative direction along the x-axis (leftward). Each object may have its own restrictions on moving in a particular direction, so the maximum leftward shift is determined individually for each object.
6. **Max negative y-axis shift (object-level control):** the variable that represents the maximum downward movement along the y-axis. It refers to the extent to which an object can move in the negative direction along the y-axis (downward). Each object may have its own restrictions on moving in a particular direction, so the maximum downward shift is determined individually for each object.
7. **Object's x-axis position:** variable unique to each object, and it represents the default position of the object's central x-axis coordinate.
8. **Object's y-axis position:** variable unique to each object, and it represents the default position of the object's central y-axis coordinate.
9. **Object's actual movement:** a tuple (x-shift, y-shift) that is unique to each object, where x-shift and y-shift are chosen randomly between the respective minimum and maximum values. The tuple determines the actual movement of the object around its central coordinates.

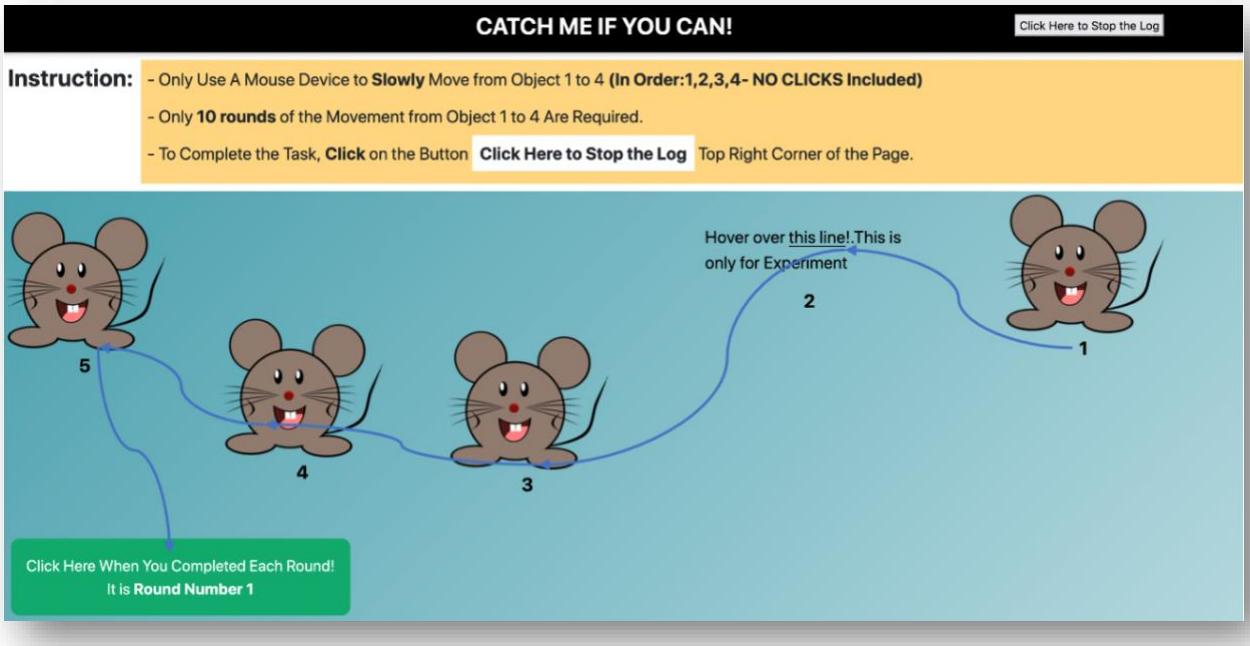


Figure 79. The experimental website "human-likebots.com".

10.4 Experimental Results

The main objective of our experimentation was to evaluate the performance of RanABD algorithm when deployed on the target webpage (i.e., server). Specifically, we wanted to observe how effective RanABD is in increasing the dissimilarity (i.e., Feature Space distance) between repeat sessions/trajectories performed by the same users on the target webpage, compared to the dissimilarity of repeat session observed under normal (i.e., no change in the webpage alignment) conditions. In other words, our hope was to confirm our hypothesis that through the use of RanABD, the clusters of repeat sessions/trajectories corresponding to genuine human users become more spread, thus making the job of 'randomization fine-tuning' for the operator of an advanced session-replay bot progressively more challenging.

Our experimentations were conducted (i.e., RanABD algorithm was deployed) on our custom designed human-likebots.com web page/site, which was also used for the purposes of experimentation in our previous experiments (as shown in Figure 79). The general objective of this site is to simulate a simple 'Catch Me If You Can!' online game – where, in one round of the game, the user is required to click on the shown objects in the designated order (first click on the object

marked with number 1, then 2, 3, 4, and finally 5). The user is generally expected to complete multiple rounds of the game, and for each new round the content of the page is refreshed. This means that in the case when RanABD algorithm is deployed on the server side, every page refreshing will return the same page content but with slightly rearranged spatial positions of its (visual) components. (Further details about the design and operation of human-likebots.com can be found in Chapter 5.)

Figure 80.a.1) presents the visualization of two repeat trajectories generated by a single user on human-likebots.com without deploying RanABD (referred to as the 'normal page' or 'non-randomized page'). In contrast, Figure 80.b.1) displays the respective trajectories on the same webpage with RanABD deployed and actively running (referred to as the 'randomized page'). Furthermore, Figure 80.a.2) and Figure 80.b.2) showcase the cumulative DTW scores [132] between the sessions generated on the non-randomized (normal page) and the randomized page, respectively.

At first glance, the differences between the trajectories depicted in Figures 80.a.1) and 80.b.1) may not be readily apparent. However, upon examining the cumulative DTW scores in Figures 80.a.2) and 80.b.2), a noticeable increase of 50% in the distance between the second pair of trajectories becomes evident. This finding demonstrates the effective application of RanABD in augmenting the dissimilarity between repeat sessions on a webpage. It is important to note that Figure 80 serves as an illustrative example showcasing a single human behavior.

To validate this observation, a thorough comparative analysis was conducted, involving multiple repeat sessions generated by 10 distinct human users. The analysis encompassed both the non-randomized and randomized webpages of human-likebots.com. Each user was instructed to adhere to the same set of procedures, as discussed earlier, and actively engage in multiple rounds of the game.

To examine the pairwise relationships among sessions generated by individual users, we conducted an analysis using DTW distance, as explained in Section 5.4. Specifically, we calculated the minimum DTW distance between all repeated sessions of each user on both the normal and randomized webpages. Subsequently, we computed the average of these minimum distances. In essence, our objective was to determine the average minimum DTW distances between the trajectories generated by human users on the normal (non-randomized) webpage compared to the randomized webpage.

The findings of our analysis, as illustrated in Table 16, provide clear evidence that, based on our collected dataset, the minimum DTW distance between repeat sessions or trajectories generated by the human users on the randomized page is nearly twice as large as the minimum DTW distance between repeat sessions/trajectories produced by the same users on the normal (non-randomized) page.

These experimental findings provide compelling evidence that the implementation of RanABD significantly enhances the dissimilarity between repeat trajectories generated by the same human user, and as such make the adversary's job of fine-tuning the randomization of advanced session-replay bots increasingly more complex.

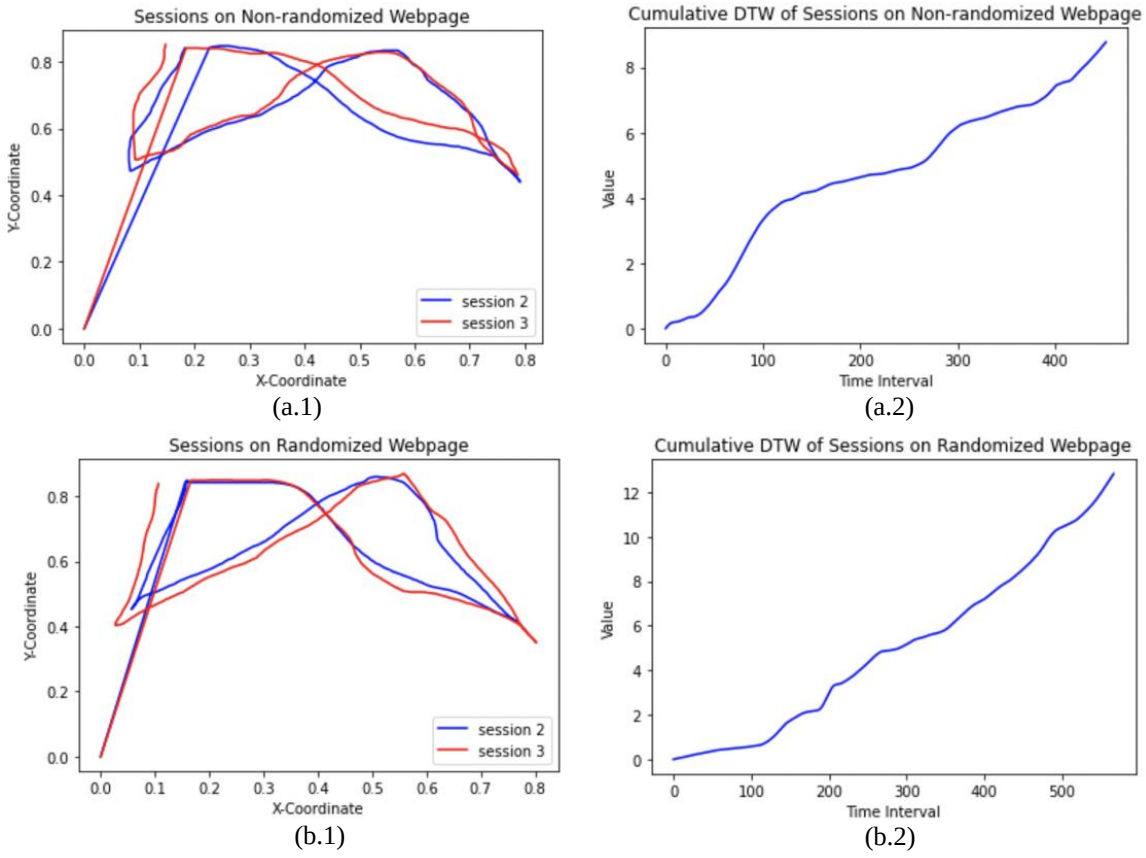


Figure 80. a.1) The visualization of two repeat trajectories generated by a genuine user on the non-randomized (normal) webpage; a.2) Cumulative DTW score between the sessions from a.1); b.1) The visualization of two repeat trajectories generated by a genuine user on the randomized webpage; b.2) Cumulative DTW score between the sessions from b.1).

Table 16. Example of Min DTW Distance calculation of visiting Normal "human-likebots.com" vs. Randomized "human-likebots.com" by the same human user.

	Min DTW Distance Score
Min DTW Distance Between Consecutive Repeat Sessions generated by Human Users on the **Normal** human-likebots.com Webpage	**6.163115**
Min DTW Distance Between Consecutive Repeat Sessions generated by the Same Human Users on the **Randomized** human-likebots.com Webpage	**11.510695**

10.5 Assessing the Efficacy of RanABD in Detecting RanReBot Attack

To assess the efficacy of RanABD in detecting advanced session-replay web bot attack, we have conducted an experiment using a new dataset generated by various human users on our experimental webpage, the randomized human-likebots.com. In this experiment, we have also applied the RanReBot attack to evaluate the effectiveness of RanABD in detecting such bots.

Following a similar methodology as described in Section 6.5 for collecting real human-user data on the original human-likebots.com page using the Amazon MTurk platform, we enlisted the assistance of 100 MTurk users to visit and interact with our randomized 'Catch Me If You Can!' site. The participants were instructed to play multiple rounds[91] of the game, each lasting for 10 minutes. In each round, they were tasked with moving a mouse device from object 1 to object 5 (as depicted in Figure 79) and clicking on the green button located in the bottom-left corner of the webpage. Participants were informed about the changes in the object locations and were instructed to properly follow and hover their mouse over the objects.

Simultaneously, while collecting new real human data through MTurk virtual users, we assumed the role of a potential attacker and periodically visited the modified human-likebots.com webpage, following the same instructions given to the MTurk users. These sessions were recorded and replayed using RanReBot, as described in Section 8.3. We referred to this new dataset as ReMouse3, which included data from 100 MTurk users, 10 iterations of webpage visits by the hacker, and 100 RanReBot data - 2287 sessions in total - (see Table 17).

In order to evaluate the effectiveness of RanABD in detecting RanReBot attacks in ReMouse3 dataset, we utilized the LSTM classifier developed and described in Section 9.2. To

[91] 10 rounds.

ensure the model's generalization capability, the data was split into training, and testing sets, with 67% allocated for training and 33% for testing. Additionally, padding was applied to the data to ensure that all sequences in a batch had a consistent length. The LSTM model architecture was specified as a sequential neural network model using the Keras API. It consisted of two layers, with 256 and 512 units, respectively. The Adam optimizer with a learning rate of 0.0001 was used to compile the model, and the binary_crossentropy loss function was employed.

To mitigate the risk of overfitting, we implemented the EarlyStopping callback, which monitored the validation accuracy and stopped training the model early when there was no significant improvement observed. This callback also restored the best weights of the model achieved during training. The model was trained for a total of 200 epochs to optimize its performance and convergence.

Table 17. Comparing Different Variations of the ReMouse Dataset.

Name of Dataset	No. of Human Data	No. of Bot Data
ReMouse – first dataset collected on "human-likebots.com"	100 MTurk Users	None
ReMouse2 – second dataset collected on **the modified version** of "human-likebots.com"	110 = 100 MTurk Users + 10 Hacker's data (10 distinct iterations of webpage visits by the hacker)	**30 ReBot** data
ReMouse2.1 – a dataset **synthesized from ReMouse2 using RanReBot**	110 = 100 MTurk Users + 10 Hacker's data (10 distinct iterations of webpage visits by the hacker)	**30 RanReBot** data
ReMouse2.1.1 – a dataset **synthesized from ReMouse2 using RanReBot**	110 = 100 MTurk Users + 10 Hacker's data (10 distinct iterations of webpage visits by the hacker)	**90 RanReBot** data, balanced dataset using actual software
ReMouse2.2 – a dataset **synthesized from ReMouse2 using TimeGAN**	110 = 80 MTurk Users + 30 Hacker's data (30 distinct iterations of webpage visits by the hacker)	**90 synthetic replayed data**, balanced dataset using Time GAN
ReMouse3 – third dataset collected on **the randomized version** of "human-likebots.com"	110 = 100 MTurk Users + 10 Hacker's data (10 distinct iterations of webpage visits by the hacker)	**100 RanReBot** data

As depicted in Figure 81, the classifier attained an accuracy rate of 86%, accompanied by a 12.3 % False Negative Rate (FNR) and a 16% False Positive Rate (FPR), as referenced in Table 18. This outcome underscores the model's success in classifying sessions, yet it also reveals some challenges in effectively identifying distinguishable patterns within human-generated sessions and RanReBot's sessions. To the best of our understanding, these challenges likely stem from the presence of two levels/aspects of randomization affecting our dataset: 1) RanABD deployed randomization of the target webpage which induces greater variability/diversity (i.e., noise) into human generate sessions/trajectories, and b) randomization in RanReBot trajectories. It can be argued that the complexity introduced by these two levels of randomization makes it arduous for the machine learning model to extract meaningful patterns from such an intricate dataset.

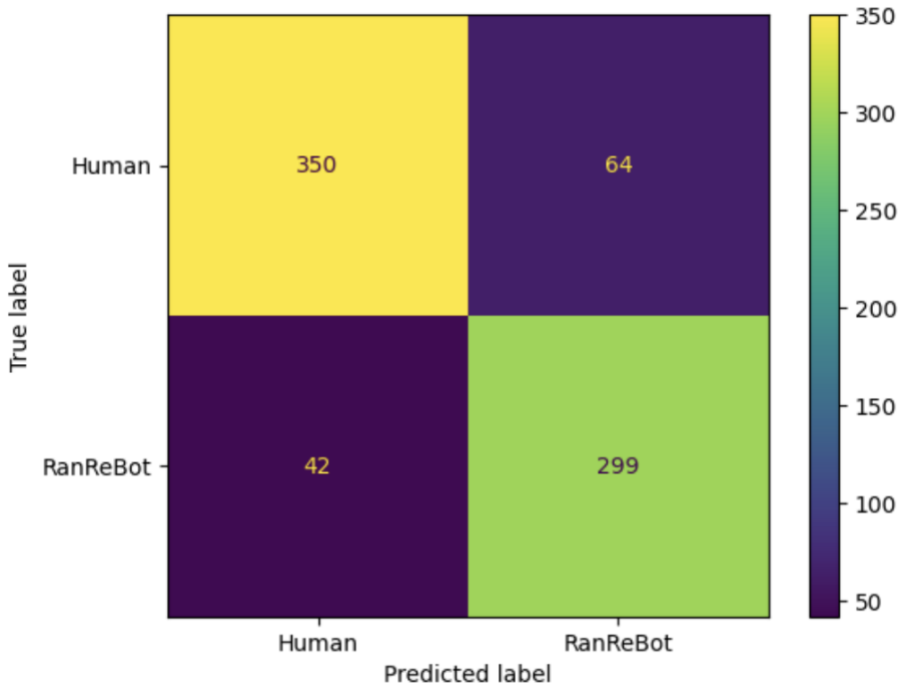


Figure 81. Confusion matrix for 2-class classification, human and RanReBot sessions on randomized webpage– ReMouse3 dataset.

Table 18. Precision, recall, and f-measure for 2-class classification.

	Precision	Recall	F1-score	**Support**
Human (class 0)	0.89	0.85	0.87	414
Bot (class 1)	0.82	0.88	0.85	341
Accuracy			0.86	755

10.6 Conclusion

In this chapter we introduced our novel MTD-based RanABD technique, which aims to facilitate more effective detection of session-replay bots from different level of randomness. The technique deploys controlled randomization in spatial-positions of main visual components/objects in the target page. Our preliminary results demonstrating the effectiveness of this technique on a real-world dataset are provided.

Chapter 11
Conclusion and Research Milestone

In this doctoral thesis, we have thoroughly investigated the intricate landscape of web bot attacks and their detrimental impact on online security. The proliferation of web bots has given rise to a multitude of malicious activities, posing a significant threat to businesses across diverse industries. These bots possess the ability to mimic human behavior, enabling them to execute their nefarious actions covertly and without detection.

Our research journey commenced by exploring the profound consequences of the surging bot traffic, the rise in malicious bot activities, and the escalating risks associated with automated malevolence. This exploration laid the groundwork for recognizing the urgent need to address this burgeoning threat. The thesis then provided a comprehensive overview of successive generations of malicious web bots, highlighting their varying levels of sophistication and the diverse dangers they present across various business sectors. We particularly emphasized the substantial impact of these malicious bots on critical aspects of business operations, including customer interactions, data security, and marketing endeavors. Subsequently, we conducted an extensive examination of web bot detection techniques, underscoring the necessity for innovative strategies and advanced machine learning approaches to effectively counter evolving threats.

Expanding the scope of our study, we endeavored to provide a comprehensive solution to mitigate the adverse impacts of web bots on the broader internet landscape. We proposed and developed our first effective web bot detection systems. Initially, we introduced an unsupervised machine learning-based detection model that incorporated automated feature selection and validation techniques using real-world data. Building on this achievement, our research delved further into the application of behavioral biometrics techniques, with a specific focus on mouse dynamics, to augment the bot detection process.

Our research underscored the urgent necessity for advanced techniques to detect and combat the latest and most sophisticated breed of web bots, known as session-replay bots. These bots mimic human behavior on target websites and applications by replicating previously recorded human mouse movements or sessions. They present a formidable challenge, particularly in online domains

where multiple authentic human users exhibit similar behavioral patterns, such as news, banking, or gaming sites.

In response to this formidable challenge, we have created the ReMouse dataset, a comprehensive repository of human mouse movement interactions recorded from our experimental website. This dataset stands as a significant contribution to the field, as it represents, to the best of our knowledge, the first publicly accessible mouse dynamics dataset that encompasses repeated sessions generated by the same human user(s). Its availability serves as a valuable resource for future research endeavors aimed at enhancing our understanding of user behavior during repetitive interactions with websites. Furthermore, the ReMouse dataset plays a pivotal role in the development of effective detection and defense techniques against session-replay bots.

To simulate session-replay bot attacks under controlled conditions, we subjected the ReMouse dataset to a series of experiments employing our custom ReBot software. This enabled us to scrutinize the behavior of session-replay bots and assess the effectiveness of our detection methods.

Harnessing the capabilities of deep learning algorithms, specifically LSTM Autoencoder, we have conceived the ReBotDetector model to effectively combat session-replay bot attacks. By integrating the ReMouse dataset and drawing insights from its analysis, we have successfully demonstrated the viability of detecting and countering original session-replay bot attacks. These findings underscore the indispensable role of advanced techniques, such as deep learning, in the ongoing battle against the evolving threats posed by web bots.

The advent of advanced AI has ushered in a new challenge in the form of meticulously randomized session-replay bots, capable of emulating human behavior while avoiding repetitive mouse trajectories. Conventional detection methods encounter significant difficulties in identifying these bots, rendering their detection highly formidable, if not nearly impossible. To counter this, we introduced an innovative advanced session-replay type of web bot named RanReBot (Randomized Session-replay Bot). This method injects variability into ReBot sessions' trajectories, augmenting the challenge faced by detection systems in flagging them as malicious.

In our pursuit of effectively countering these sophisticated attacks, we have introduced the RanReBotDetector model, designed for the accurate identification and capture of session-replay bots that exhibit randomized behavior. Employing stacked LSTM-based models for sequence classification, our approach was designed to address the intricate challenge presented by

randomized and advanced session-replay bot attacks. Through a rigorous evaluation of our model using RanReBot data, we achieved a remarkable accuracy rate of 93%. Of even greater significance, we conducted comprehensive tests to assess our model's proficiency in detecting synthetic replayed sessions generated by TimeGAN, effectively addressing the challenge of using generative models to replicate advanced session-replay bot behavior. In this evaluation, our proposed model exhibited exceptional performance, achieving a perfect accuracy rate of 100%.

As our last contribution, we introduced RanABD, an innovative MTD-based webpage randomization technique aimed at countering advanced session-replay web bot attacks. RanABD integrates randomized micro-adjustments in the alignment of specific visual HTML elements and their attributes on the target webpage, all while minimizing disruptions to the overall appearance and functionality of the page. Preliminary results were presented to illustrate the effectiveness of this technique using real-world datasets.

In its entirety, this thesis has made substantial contributions to the field of web bot detection. It has proposed effective detection models, explored the utility of behavioral biometrics techniques, and underscored the significance of dynamic and resilient system architectures. The work has illuminated the specific challenges posed by session-replay web bots and offered valuable insights for the development of robust security measures to protect online systems from evolving bot threats.

Looking ahead, future research endeavors should prioritize the refinement of detection techniques, the enhancement of access to real-world datasets, and the facilitation of collaboration between academia and industry. These efforts are crucial for proactively addressing malicious web bot activities in the ever-evolving digital landscape and ensuring the continued security of online systems.

To pave the way for future research in this field, several promising directions and areas of focus can be explored. One such direction is:

In the realm of ***adversarial machine learning***, it is imperative to anticipate the evolution of bots' behaviors and their potential understanding of the underlying machine learning models employed for detection. As bots become increasingly sophisticated, it is foreseeable that they will strive to deceive these models by exploiting their vulnerabilities. This poses a significant challenge for bot detection systems and other adversarial solutions, as they must adapt and enhance their resilience against potential attacks on the machine learning models. In order to tackle this issue, it

becomes crucial to develop approaches that render these detection systems more resistant and robust.

One aspect to consider is fortifying the defenses against adversarial attacks aimed at compromising the learning model. Bots, once they comprehend the inner workings of the model, may attempt to poison the training data, thereby distorting the learning process and undermining the accuracy of the detection system. To counteract this threat, strategies need to be devised to make ***the detection system more resilient to such poisoning attacks***. This could involve incorporating robust training techniques, such as data augmentation, anomaly detection, or adversarial training, to ensure the model's integrity even in the face of manipulated training data.

Furthermore, enhancing the overall robustness of the detection system entails exploring techniques that go beyond traditional machine learning approaches. This could involve the integration of multiple models or ***ensemble methods***, where different models are combined to leverage their complementary strengths and create a more resilient detection system. Additionally, exploring the use of explainable AI techniques can help identify potential vulnerabilities in the models and enhance their resistance to adversarial attacks.

In conclusion, while RanABD has demonstrated its effectiveness in incorporating randomness and unpredictability into the webpage structure to deter session-replay bots, it is important to acknowledge the inherent limitations of the ML-based technique in fully capturing and showcasing the true effectiveness of RanABD in detecting RanReBot attacks. The complexity of the data, coupled with the presence of high levels of randomness, presents significant challenges for the ML method. Therefore, it is essential to explore alternative approaches that can accurately evaluate and highlight the capabilities of RanABD in detecting and mitigating bot attacks. Additionally, we aim to conduct a more extensive evaluation using a diverse set of data on RanABD to enhance our result.

Bibliography

[1] Chu, Z.; Gianvecchio, S.; Wang, H. Bot or Human? A Behavior-Based Online Bot Detection System. In From Database to Cyber Security: Essays Dedicated to Sushil Jajodia on the Occasion of His 70th Birthday; Samarati, P., Ray, I., Ray, I., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, 2018; pp 432–449.
[2] Iliou, C., Kostoulas, T., Tsikrika, T., Katos, V., Vrochidis, S., & Kompatsiaris, Y. (2021). Detection of advanced Web bots by combining Web logs with mouse behavioural biometrics. Digital Threats: Research and Practice.
[3] 2021 Global Data Privacy Regulation of Physical & Behavioral Biometrics | BehavioSec https://www.goodeintelligence.com/wp-content/uploads/2021/05/2021-Global-Data-Privacy-Regulation-of-Physical-and-Behavioral-Biometrics-REPORT.pdf (accessed 2023-04-28).
[4] Imperva. (2023). Bad Bot Report. https://www.imperva.com/resources/reports/2023-Imperva-Bad-Bot-Report.pdf (accessed 2023-07-20).
[5] Netacea. (2023). Cybersecurity and Bot Predictions 2023 Report. https://netacea.com/uploads/cybersecurity_and_bot_predictions_2023_report.pdf (accessed 2023-07-20).
[6] Mirtaheri, S.M.; Dinçktürk, M.E.; Hooshmand, S.; Bochmann, G.V.; Jourdan, G.-V.; Onut, I.V. A Brief History of Web Crawlers. arXiv 2014, arXiv:1405.0749.
[7] Managing and Mitigating Bots: The Automated Threat Guide—Netacea. 2018. https://www.netacea.com/managing-and-mitigating-bots-guide/ (accessed 2023-04-25).
[8] Netacea Cybersecurity and Bot Predictions 2023. Netacea. https://netacea.com/research-and-reports/cyber-predictions-2023/ (accessed 2023-04-25).
[9] Ultimate Guide to Bot Management. [E-book] Radware. 2019. https://blog.radware.com/wp-content/uploads/2019/09/Radware_UltimateGuideBotManagement_Final.pdf (accessed 2023-04-25).
[10] Website Performance and Availability Monitoring - Pingdom. pingdom.com. https://www.pingdom.com/ (accessed 2023-07-20).
[11] Stay ahead of the curve with Feedly AI. Feedly. https://feedly.com (accessed 2023-07-20).
[12] What Is Semrushbot And Should You Block It? https://www.searchlogistics.com/learn/seo/semrushbot/ (accessed 2023-07-20).
[13] How PayPal Integration works - West Wind Web Store .NET. https://www.west-wind.com/westwindwebstore/docs/_1bn0wb4gs.htm (accessed 2023-07-20).
[14] Everything You Need to Know about Bots in 2020. Netacea. 2020. https://www.netacea.com/evolving-threatguide-2020/ (accessed 2023-04-25).
[15] What is click fraud? | How click bots work. Cloudflare. https://www.cloudflare.com/learning/bots/what-is-click-fraud/ (accessed 2023-07-21).
[16] What Is a Botnet and Its Functionality? | Radware. https://www.radware.com/cyberpedia/bot-management/botnet/ (accessed 2023-04-25).
[17] IBM Developer. What is the cURL command? https://developer.ibm.com/articles/what-is-curl-command/ (accessed 2023-07-20).
[18] PhantomJS - Scriptable Headless Browser. https://phantomjs.org/#:~:text=PhantomJS%20is%20a%20headless%20web,JSON%2C%20Canvas%2C%20and%20SVG. (accessed 2023-07-21).

[19] What is a spam bot? | How spam comments and spam messages spread. Cloudflare. https://www.cloudflare.com/learning/bots/what-is-a-spambot/ (accessed 2023-07-21).
[20] Ad Fraud Stats (2023). Business of Apps. https://www.businessofapps.com/research/ad-fraud-statistics/ (accessed 2023-08-01).
[21] Cyber Security Threat Analysis In Higher Education Institutions As A Result Of Distance Learning. https://ibn.idsi.md/vizualizare_articol/163773 (accessed 2023-07-21).
[22] GlobalDots, D. E. R., Senior Solutions Engineer &. Security Analyst @. Industry Report: Bad Bot Landscape 2019 - The Bot Arms Race Continues. GlobalDots. https://www.globaldots.com/resources/blog/industry-report-bad-bot-landscape-2019-the-bot-arms-race-continues/ (accessed 2023-04-25).
[23] Sadeghpour, Shadi, and Natalija Vlajic. "Ads and Fraud: A Comprehensive Survey of Fraud in Online Advertising." Journal of Cybersecurity and Privacy 1.4 (2021): 804-832.
[24] Sadeghpour, Shadi, and Natalija Vlajic. "Click Fraud in Digital Advertising: A Comprehensive Survey." Computers 10.12 (2021): 164.
[25] Iliou, C., Kostoulas, T., Tsikrika, T., Katos, V., Vrochidis, S., & Kompatsiaris, Y. (2019, August). Towards a framework for detecting advanced Web bots. In Proceedings of the 14th International Conference on Availability, Reliability and Security (pp. 1-10).
[26] Tariq, N. O. S. H. I. N. A., Khan, F. A., Moqurrab, S. A., & Srivastava, G. (2023). CAPTCHA Types and Breaking Techniques: Design Issues, Challenges, and Future Research Directions. arXiv preprint arXiv:2307.10239.
[27] Guerar, M., Verderame, L., Migliardi, M., Palmieri, F., Merlo, A.: Gotta captcha'em all: a survey of 20 years of the human-or-computer dilemma. ACM Computing Surveys (CSUR) 54(9), 1–33 (2021).
[28] Chen, H.; He, H.; Starr, A. An Overview of Web Robots Detection Techniques. In 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security); 2020; pp 1–6.
[29] Eslahi, M.; Salleh, R.; Anuar, N. B. Bots and Botnets: An Overview of Characteristics, Detection and Challenges. In 2012 IEEE International Conference on Control System, Computing and Engineering; IEEE, 2012; pp 349–354.
[30] Singh, K., Singh, P., & Kumar, K. (2017). Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges. Computers & security, 65, 344-372.
[31] Antal, M., & Egyed-Zsigmond, E. (2019). Intrusion detection using mouse dynamics. IET Biometrics, 8(5), 285-294.
[32] How behavioral biometrics is used to identify bots faster than ever before. 2021. https://www.behaviosec.com/behavioral-biometrics-used-identify-bots-faster-ever/ (accessed 2023-04-28).
[33] Von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: Captcha: Using hard ai problems for security. In: Eurocrypt. vol. 2656, pp. 294–311. Springer (2003).
[34] Motoyama, M., Levchenko, K., Kanich, C., McCoy, D., Voelker, G.M., Savage, S.:Re: Captchas-understanding captcha-solving services in an economic context. In: USENIX Security Symposium. vol. 10, p. 3 (2010)
[35] Jin, R., Huang, L., Duan, J., Zhao, W., Liao, Y., & Zhou, P. (2023). How Secure is Your Website? A Comprehensive Investigation on CAPTCHA Providers and Solving Services. arXiv preprint arXiv:2306.07543.
[36] Stop SMS Toll Fraud With Arkose Matchkey (CAPTCHA Software) | Arkose Labs. https://www.arkoselabs.com/arkose-matchkey/ (accessed 2023-07-31).
[37] hCaptcha - Stop more bots. Start protecting privacy. https://www.hcaptcha.com/ (accessed 2023-07-31).

[38] McKenna, S. F. (2016). Detection and classification of Web robots with honeypots. Naval Postgraduate School Monterey United States.
[39] Li, X., Azad, B. A., Rahmati, A., & Nikiforakis, N. (2021, January). Good bot, bad bot: Characterizing automated browsing activity. In 2021 IEEE symposium on security and privacy (sp) (p. 17).
[40] Gržinić, T., Mršić, L., & Šaban, J. (2015, March). Lino-an intelligent system for detecting malicious Web-robots. In Asian Conference on Intelligent Information and Database Systems (pp. 559-568). Springer, Cham.
[41] Distil Networks. 2019. 2019 BAD BOT REPORT: The Bot Arms Race Continues. Retrieved from https://resources.distilnetworks.com/white-paper-reports/bad-bot-report-2019.
[42] Babak Amin Azad, Oleksii Starov, Pierre Laperdrix, and Nick Nikiforakis. 2020. Web Runner 2049: Evaluating Third-Party Anti-bot Services. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, 135–159.
[43] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser fingerprinting: A survey. ACM Trans. Web 14, 2 (2020), 1–33.
[44] Michael Schwarz, Florian Lackner, and Daniel Gruss. 2019. JavaScript template attacks: Automatically inferring host information for targeted exploits. In Proceedings of the Network and Distributed System Security Symposium (NDSS'19).
[45] Pierre Laperdrix,Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the beast: Diverting modern Web browsers to build unique browser fingerprints. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP'16). IEEE, 878–894.
[46] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. 2020. FP-Crawlers: Studying the resilience of browser fingerprinting to block crawlers. In Proceedings of the NDSS Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb'20).
[47] Timonera, K. 6 Best Bot Protection Solutions and Software for 2023. eSecurityPlanet. https://www.esecurityplanet.com/products/bot-protection/ (accessed 2023-08-01).
[48] Suchacka, G., & Iwański, J. (2020). Identifying legitimate Web users and bots with different traffic profiles–an Information Bottleneck approach. Knowledge-Based Systems, 105875.
[49] What are Cookies?. www.kaspersky.com. https://www.kaspersky.com/resource-center/definitions/cookies (accessed 2023-07-30).
[50] Rahman, R. U., & Tomar, D. S. (2020). New biostatistics features for detecting Web bot activity on Web applications. Computers & Security, 97, 102001.
[51] Stefano Rovetta, Alberto Cabri, Francesco Masulli, and Grażyna Suchacka. 2017. Bot or not? A case study on bot recognition from Web session logs. In Proceedings of the Italian Workshop on Neural Nets. Springer, 197–206.
[52] Dusan Stevanovic, Aijun An, and Natalija Vlajic. 2012. Feature evaluation for Web crawler detection with data mining techniques. Expert Syst. Appl. 39, 10 (2012), 8707–8717.
[53] Dilip Singh Sisodia, Shrish Verma, and Om Prakash Vyas. 2015. Agglomerative approach for identification and elimination of Web robots from Web server logs to extract knowledge about actual visitors. J. Data Anal. Inf. Process. 3, 01 (2015), 1.
[54] Alberto Cabri, Grażyna Suchacka, Stefano Rovetta, and Francesco Masulli. 2018. Online Web bot detection using a sequential classification approach. In Proceedings of the 2018 IEEE 20th International Conference on High Performance Computing and Communications.
[55] Bomhardt, C., Gaul, W., & Schmidt-Thieme, L. (2005). Web robot detection-preprocessing Web logfiles for robot detection. In New developments in classification and data analysis (pp. 113-124). Springer, Berlin, Heidelberg.

[56] Suchacka, G., Cabri, A., Rovetta, S., & Masulli, F. (2021). Efficient on-the-fly Web bot detection. Knowledge-Based Systems, 223, 107074.
[57] Rahman, R. U., & Tomar, D. S. (2021). Threats of price scraping on e-commerce Websites: attack model and its detection using neural network. Journal of Computer Virology and Hacking Techniques, 17(1), 75-89.
[58] Suchacka, G., & Sobkow, M. (2015, June). Detection of Internet robots using a Bayesian approach. In 2015 IEEE 2nd International Conference on Cybernetics (CYBCONF) (pp. 365-370). IEEE.
[59] Haidar, R., & Elbassuoni, S. (2017, October). Website navigation behavior analysis for bot detection. In 2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA) (pp. 60-68). IEEE.
[60] Shafiq Alam, Gillian Dobbie, Yun Sing Koh, and Patricia Riddle. 2014. Web bots detection using particle swarm optimization based clustering. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC'14). IEEE, 2955–2962.
[61] Sadeghpour, S., Vlajic, N., Madani, P., & Stevanovic, D. (2021, January). Unsupervised ML Based Detection of Malicious Web Sessions with Automated Feature Selection: Design and Real-World Validation. In 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC) (pp. 1-9). IEEE.
[62] Dusan Stevanovic, Natalija Vlajic, and Aijun An. 2013. Detection of malicious and non-malicious Website visitors using unsupervised neural network learning. Appl. Soft Comput. 13, 1 (2013), 698–708.
[63] Zabihi, M., Jahan, M. V., & Hamidzadeh, J. (2014, October). A density based clustering approach for Web robot detection. In 2014 4th International Conference on Computer and Knowledge Engineering (ICCKE) (pp. 23-28). IEEE.
[64] Zabihimayvan, M., Sadeghi, R., Rude, H. N., & Doran, D. (2017). A soft computing approach for benign and malicious Web robot detection. Expert Systems with Applications, 87, 129-140.
[65] J. Hamidzadeh, M. Zabihimayvan, R. Sadeghi, Detection of Web site visitors based on fuzzy rough sets, Soft Comput. 22 (7) (2018) 2175–2188.
[66] G. Suchacka, Improving clustering of Web bot and human sessions by applying Principal Component Analysis, in: Proceedings of the 33rd International ECMS Conference on Modelling and Simulation (ECMS'19), 2019, pp. 000–000.
[67] Rovetta, S.; Suchacka, G.; Masulli, F. Bot Recognition in a Web Store: An Approach Based on Unsupervised Learning. Journal of Network and Computer Applications 2020, 157, 102577.
[68] Van Balen, N., Ball, C. T., & Wang, H. (2017). A behavioral biometrics based approach to online gender classification. In Security and Privacy in Communication Networks: 12th International Conference, SecureComm 2016, Guangzhou, China, October 10-12, 2016, Proceedings 12 (pp. 475-495). Springer International Publishing.
[69] Kirsh, I.; Joy, M. Exploring Pointer Assisted Reading (PAR): Using Mouse Movements to Analyze Web Users' Reading Behaviors and Patterns. In International Conference on Human-Computer Interaction; Springer, 2020; pp 156–173.
[70] Stillman, P. E.; Shen, X.; Ferguson, M. J. How Mouse-Tracking Can Advance Social Cognitive Theory. Trends in Cognitive Sciences 2018, 22 (6), 531–543.
[71] Kirsh, I. Directions and Speeds of Mouse Movements on a Website and Reading Patterns: A Web Usage Mining Case Study. In Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics; 2020; pp 129–138.
[72] Hehman, E.; Stolier, R. M.; Freeman, J. B. Advanced Mouse-Tracking Analytic Techniques for Enhancing Psychological Science. Group Processes & Intergroup Relations 2015, 18 (3), 384–401.

[73] Kaixin, W.; Hongri, L.; Bailing, W.; Shujie, H.; jia, S. A User Authentication and Identification Model Based on Mouse Dynamics. In Proceedings of the 6th International Conference on Information Engineering; ICIE '17; Association for Computing Machinery: New York, NY, USA, 2017; pp 1–6.
[74] Antal, M.; Fejér, N.; Buza, K. SapiMouse: Mouse Dynamics-Based User Authentication Using Deep Feature Learning. In 2021 IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI); 2021; pp 61–66.
[75] Wei, A.; Zhao, Y.; Cai, Z. A Deep Learning Approach to Web Bot Detection Using Mouse Behavioral Biometrics. In Biometric Recognition; Sun, Z., He, R., Feng, J., Shan, S., Guo, Z., Eds.; Lecture Notes in Computer Science; Springer International Publishing: Cham, 2019; pp 388–395.
[76] Stevanovic, D., Vlajic, N., & An, A. (2013). Detection of malicious and non-malicious Website visitors using unsupervised neural network learning. Applied Soft Computing, 13(1), 698-708.
[77] T. Kohonen, Self-Organizing Maps, 3rd ed., Springer-Verlag, Berlin Heidelberg, New York, 2001.
[78] N. Vlajic, H.C. Card, Vector quantization of images using modified adaptive resonance algorithm for hierarchical clustering, IEEE Transactions on Neural Networks 12 (September (5)) (2001) 1147–1162.
[79] Chambers, B., Zaharia, M. (2018). Spark: The definitive guide: Big data processing made simple. " O'Reilly Media, Inc.".
[80] MIME types (IANA media types) - HTTP | MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types (accessed 2023-08-09).
[81] Harvester User Agents | Project Honey Pot. https://www.projecthoneypot.org/harvester_useragents.php (accessed 2023-08-09).
[82] KLOTH.NET - List of Bad Bots. http://www.kloth.net/internet/badbots.php (accessed 2023-04-27).
[83] Browse our database of 219.4 million User Agents. WhatIsMyBrowser.com. https://explore.whatismybrowser.com/useragents/explore/ (accessed 2023-04-27).
[84] Xu, Z., Huang, G., Weinberger, K. Q., & Zheng, A. X. (2014). Gradient boosted feature selection. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 522-531).
[85] sklearn.ensemble.HistGradientBoostingClassifier. scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html (accessed 2023-04-27).
[86] Breard, G. T. Evaluating Self-Organizing Map Quality Measures as Convergence Criteria. 2017.
[87] Akinduko, A. A.; Mirkes, E. M. Initialization of Self-Organizing Maps: Principal Components versus Random Initialization. A Case Study. arXiv preprint arXiv:1210.5873 2012.
[88] sevamoo. Sevamoo/SOMPY, 2023. https://github.com/sevamoo/SOMPY (accessed 2023-05-01).
[89] Top 10 Powerful Python Libraries for Data Science. Shiksha Online. https://www.shiksha.com/online-courses/articles/top-10-powerful-python-libraries-for-data-science/ (accessed 2023-08-05).
[90] AbuseIPDB - IP address abuse reports - Making the Internet safer, one IP at a time. https://www.abuseipdb.com/ (accessed 2023-04-27).
[91] IP Address Blacklist Check. WhatIsMyIPAddress. https://whatismyipaddress.com/blacklist-check (accessed 2023-04-27).
[92] DNSBL Information - Spam Database and Blacklist Check. https://www.dnsbl.info/ (accessed 2023-08-03).
[93] MX Lookup Tool - Check your DNS MX Records online. MxToolbox. http://mxtoolbox.com/default.aspx (accessed 2023-08-03).
[94] Alpaydm, E. (1999). Combined 5× 2 cv F test for comparing supervised classification learning algorithms. Neural computation, 11(8), 1885-1892.

[95] Choropleth. https://plotly.com/python/choropleth-maps/ (accessed 2023-04-27).
[96] Web application attack traffic by country 2018. Statista. https://www.statista.com/statistics/276425/internet-attack-traffic-by-originating-country/ (accessed 2023-04-27).
[97] Rahman, R. U., & Tomar, D. S. (2020). A new Web forensic framework for bot crime investigation. Forensic Science International: Digital Investigation, 33, 300943.
[98] Acien, A.; Morales, A.; Fierrez, J.; Vera-Rodriguez, R. BeCAPTCHA-Mouse: Synthetic Mouse Trajectories and Improved Bot Detection. arXiv:2005.00890 [cs] 2021.
[99] Bravo, S., Mauricio, D., & Moreno, Á. H. (2017, October). Mouse features for DDoS attacks detection in the application layer. In Proceedings of the 9th International Conference on Information Management and Engineering (pp. 177-181).
[100] Chuda, D., Kratky, P., & Tvarozek, J. (2015, May). Mouse clicks can recognize Web page visitors! In Proceedings of the 24th International Conference on World Wide Web (pp. 21-22).
[101] Hu, T., Niu, W., Zhang, X., Liu, X., Lu, J., & Liu, Y. (2019). An insider threat detection approach based on mouse dynamics and deep learning. Security and Communication Networks, 2019.
[102] Chong, P., Elovici, Y., & Binder, A. (2019). User authentication based on mouse dynamics using deep neural networks: A comprehensive study. IEEE Transactions on Information Forensics and Security, 15, 1086-1101.
[103] Yıldırım, M., & Anarım, E. Novel Feature Extraction Methods for Authentication via Mouse Dynamics with Semi-Supervised Learning. In 2019 Innovations in Intelligent Systems and Applications Conference (ASYU) (pp. 1-6). IEEE.
[104] Tharwat, A. Classification Assessment Methods. Applied computing and informatics 2020, 17 (1), 168–192.
[105] ANTAL, M., & FEJÉR, N. (2020). Mouse dynamics based user recognition using deep learning. Acta Universitatis Sapientiae, Informatica, 12(1), 39-50.
[106] Yildirim, M., & Anarim, E. (2021). Mitigating insider threat by profiling users based on mouse usage pattern: ensemble learning and frequency domain analysis. International Journal of Information Security, 1-13.
[107] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
[108] Pozzana, I., & Ferrara, E. (2020). Measuring bot and human behavioral dynamics. Frontiers in Physics, 8, 125.
[109] BALABIT MOUSE CHALLENGE DATA SET, 2023. https://github.com/balabit/Mouse-Dynamics-Challenge (accessed 2023-04-30).
[110] Antal, M., & Denes-Fazakas, L. (2019, May). User Verification Based on Mouse Dynamics: a Comparison of Public Data Sets. In 2019 IEEE 13th International Symposium on Applied Computational Intelligence and Informatics (SACI) (pp. 143-148). IEEE.
[111] Quraishi, S., Bedi, S. (2019). Mouse Dynamics as Continuous User Authentication Tool. International Journal of Recent Technology and Engineering, 8(4), 10923–10927.
[112] Raj, S. B. E.; Santhosh, A. T. A Behavioral Biometric Approach Based on Standardized Resolution in Mouse Dynamics. International Journal of Computer Science and Network Security 2009, 9 (4), 370–377.
[113] Ahmed, A. A. E.; Traore, I. A New Biometric Technology Based on Mouse Dynamics. IEEE Transactions on dependable and secure computing 2007, 4 (3), 165–179.

[114] Huang, J., White, R. W., & Dumais, S. (2011, May). No clicks, no problem: using cursor movements to understand and improve search. In Proceedings of the SIGCHI conference on human factors in computing systems (pp. 1225-1234).
[115] Almalki, S.; Assery, N.; Roy, K. An Empirical Evaluation of Online Continuous Authentication and Anomaly Detection Using Mouse Clickstream Data Analysis. Applied Sciences 2021, 11 (13), 6083.
[116] Niu, H.; Chen, J.; Zhang, Z.; Cai, Z. Mouse Dynamics Based Bot Detection Using Sequence Learning. In Chinese Conference on Biometric Recognition; Springer, 2021; pp 49–56.
[117] Chu, Z.; Gianvecchio, S.; Koehl, A.; Wang, H.; Jajodia, S. Blog or Block: Detecting Blog Bots through Behavioral Biometrics. Computer Networks 2013, 57 (3), 634–646.
[118] Goßen, D., Jonker, H., Karsch, S., Krumnow, B. and Roefs, D., 2021, November. HLISA: Towards a more reliable measurement tool. In Proceedings of the 21st ACM Internet Measurement Conference (pp. 380-389).
[119] Crowther, S. What Is Puppeteer? Why Developers and Fraudsters Love It. Kasada, 2021.
[120] How Attackers Use Request Bots to Bypass Your Bot Mitigation Solution. Security Boulevard, 2021.
[121] Niu, H., Wei, A., Song, Y. and Cai, Z., Exploring Visual Representations of Computer Mouse Movements for Bot Detection Using Deep Learning Approaches. Available at SSRN 4329684.
[122] Antal, M.; Buza, K.; Fejer, N. SapiAgent: A Bot Based on Deep Learning to Generate Human-Like Mouse Trajectories. IEEE Access 2021, 9, 124396–124408.
[123] Sadeghpour, S.; Vlajic, N. ReMouse - Mouse Dynamic Dataset, 2022. https://ieee-dataport.org/documents/remouse-mouse-dynamic-dataset (accessed 2023-04-30).
[124] Rahman, R.U.; Tomar, D.S. Threats of price scraping on e-commerce Websites: Attack model and its detection using neural network. J. Comput. Virol. Hacking Tech. 2020, 17, 75–89.
[125] Rieniets, N. How Attackers Use Request Bots to Bypass Your Bot Mitigation Solution. Security Boulevard. https://securityboulevard.com/2021/07/how-attackers-use-request-bots-to-bypass-your-bot-mitigation-solution/ (accessed 2023-04-30).
[126] Kılıç, A.A.; Yıldırım, M.; Anarım, E. Bogazici mouse dynamics dataset. Data Brief 2021, 36, 107094.
[127] ANTAL, M. SapiMouse, 2023. https://github.com/margitantal68/sapimouse (accessed 2023-05-01).
[128] Leiva, L.A.; Arapakis, I. The Attentive Cursor Dataset. Front. Hum. Neurosci. 2020, 14, 565664.
[129] Shen, C.; Cai, Z.; Guan, X. Continuous authentication for mouse dynamics: A pattern-growth approach. In Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012), Boston, MA, USA, 25–28 June 2012; pp. 1–12.
[130] Karim, M. Hasanuzzaman A Study on Mouse Movement Features to Identify User. Sci. Res. J. 2020, 8, 77–82.
[131] Home - Django REST framework. https://www.django-rest-framework.org/ (accessed 2023-05-01).
[132] A measure of distance between time-series: Dynamic Time Warping - INFORMS. https://www.informs.org/Publications/OR-MS-Tomorrow/A-measure-of-distance-between-time-series-Dynamic-Time-Warping (accessed 2023-05-01).
[133] Morse, G. Programmatic Identification of Support/Resistance Trend lines with Python. Medium. https://towardsdatascience.com/programmatic-identification-of-support-resistance-trend-lines-with-python-d797a4a90530 (accessed 2023-05-01).
[134] Thomas, P.A.; Mathew, K.P. A Broad Review on Non-Intrusive Active User Authentication in Biometrics. J. Ambient. Intell. Human Comput. 2023, 14, 339–360.
[135] Simonyan, K.; Andrew, Z. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 2014, arXiv:1409.1556.
[136] Keras Applications, 2023. https://github.com/keras-team/keras-applications (accessed 2023-05-01).

[137] Liu, F.;Wang, Y.; Wang, F.-C.; Zhang, Y.-Z.; Lin, J. Intelligent and Secure Content-Based Image Retrieval for Mobile Users. IEEE Access 2019, 7, 119209–119222.
[138] Hands-on Transfer Learning with Keras and the VGG16 Model. https://www.learndatasci.com/tutorials/hands-on-transfer-learning-keras/ (accessed 2023-05-01).
[139] Brownlee, J. Transfer Learning in Keras with Computer Vision Models. MachineLearningMastery.com. https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/ (accessed 2023-05-01).
[140] Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2009; pp. 248–255.
[141] Team, K. Keras documentation: Keras Applications. https://keras.io/api/applications/#vgg16 (accessed 2023-05-01).
[142] Cord, M., & Cunningham, P. (Eds.). (2008). Machine learning techniques for multimedia: case studies on organization and retrieval. Springer Science & Business Media.
[143] Salgado, C. M., & Vieira, S. M. (2020). Machine learning for patient stratification and classification part 2: unsupervised learning with clustering. Leveraging data science for global health, 151-168.
[144] Gupta, R. Deeper Dive into Self-Organizing Maps (SOMs). Water Programming: A Collaborative Research Blog. https://waterprogramming.wordpress.com/2020/07/20/deeper-dive-into-self-organizing-maps-soms/ (accessed 2023-05-01).
[145] Marzouki, K.; Takeshi, Y. Novel Algorithm for Eliminating Folding Effect in Standard SOM. In ESANN; Citeseer: Princeton, NJ, USA, 2005; pp. 563–570.
[146] Brennan, D., & Van Hulle, M. M. (2007). Comparison of flat SOM with spherical SOM. A case study. The Self-Organizing Maps and the Development—From Medicine and Biology to the Sociological Field, 31-41.
[147] Kohonen, T. (2001). Springer series in information sciences, Self-organizing maps (3rd ed.). Germany: Springer-Verlag.
[148] Sangole, A., & Knopf, G. K. (2003). Visualization of randomly ordered numeric data sets using spherical self-organizing feature maps. Computers & Graphics, 27(6), 963-976.
[149] Farid, B., Biela, E. P., & Jack-Gérard, P. (2003, September). Self organizing spherical map architecture for 3d object modeling. In Proceedings of Workshop on Self-Organizing Maps, WSOM03, Kitakyushu, Japan.
[150] Nishio, H., Altaf-Ul-Amin, M., Kurokawa, K., Minato, K., & Kanaya, S. (2005, September). Spherical som with arbitrary number of neurons and measure of suitability. In Proceedings of WSOM (Vol. 5, pp. 323-330).
[151] Nakatsuka, D. (2003). Application of spherical SOM in clustering. In Proceedings of Workshop on Self-Organizing Maps (WSO M'03), Japan (pp. 203-207).
[152] Wu, Y., & Takatsuka, M. (2006). Spherical self-organizing map using efficient indexed geodesic data structure. Neural Networks, 19(6-7), 900-910.
[153] Ito, M. (2000). The characteristics of the torus self organizing map. In Proceedings 16th Fuzzy Systerm Symposium Akita, 2000. Japan Society for Fuzzy and Systems.
[154] Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(11).
[155] Violante, A. An Introduction to t-SNE with Python Example. Medium. https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1 (accessed 2023-05-01).

[156] sklearn.manifold.TSNE. scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.manifold.TSNE.html (accessed 2023-05-01).
[157] Kind, M. C. Somsphere, 2022. https://github.com/mgckind/somsphere (accessed 2023-05-01).
[158] Kind, M. C., & Brunner, R. J. (2014). SOMz: Photometric redshift PDFs with self organizing maps and random atlas. Monthly Notices of the Royal Astronomical Society, 438(4), 3409–3421.
[159] Healpy, a Python Wrapper for Healpix, 2023. https://github.com/healpy/healpy (accessed 2023-05-01).
[160] sklearn.cluster.KMeans. scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.cluster.KMeans.html (accessed 2023-05-01).
[161] sklearn.cluster.AgglomerativeClustering. scikit-learn. https://scikit-learn/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html (accessed 2023-05-01).
[162] Davies, D.L.; Bouldin, D.W. A Cluster Separation Measure. IEEE Trans. Pattern Anal. Mach. Intell. 1979, 2, 224–227.
[163] Drakos, G. Geodra/Articles, 2023. https://github.com/geodra/Articles/blob/master/Davies-Bouldin%20Index%20vs%20Silhouette%20Analysis%20vs%20Elbow%20Method%20Selecting%20the%20optimal%20number%20of%20clusters%20for%20KMeans%20clustering.ipynb (accessed 2023-05-01).
[164] Aguinis, H.; Villamor, I.; Ramani, R.S. MTurk Research: Review and Recommendations. J. Manag. 2020, 47, 823–837.
[165] Rahman, R. U.; Tomar, D. S. Threats of Price Scraping on E-Commerce Websites: Attack Model and Its Detection Using Neural Network. J Comput Virol Hack Tech 2021, 17 (1), 75–89.
[166] Zheng, N.; Paloski, A.; Wang, H. An Efficient User Verification System Using Angle-Based Mouse Movement Biometrics. ACM Transactions on Information and System Security (TISSEC) 2016, 18 (3), 1–27.
[167] Why Selenium Clicks Fail. Lucidchart. https://www.lucidchart.com/techblog/2020/01/21/why-selenium-clicks-fail/ (accessed 2023-08-15).
[168] Siddiqui, N., Dave, R., Vanamala, M., & Seliya, N. (2022). Machine and deep learning applications to mouse dynamics for continuous user authentication. Machine Learning and Knowledge Extraction, 4(2), 502-518.
[169] Arapakis, I.; Leiva, L. A. Learning Efficient Representations of Mouse Movements to Predict User Attention. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval; 2020; pp 1309–1318.
[170] Yang, X., Wan, C., Zhang, T., & Xiong, Z. (2022, August). Feature Extraction of Sequence Data Based on LSTM and its Application to Fault Diagnosis of Industrial Process. In 2022 IEEE 11th Data Driven Control and Learning Systems Conference (DDCLS) (pp. 693-698).
[171] Cho, J.-H.; Sharma, D. P.; Alavizadeh, H.; Yoon, S.; Ben-Asher, N.; Moore, T. J.; Kim, D. S.; Lim, H.; Nelson, F. F. Toward Proactive, Adaptive Defense: A Survey on Moving Target Defense. IEEE Communications Surveys & Tutorials 2020, 22 (1), 709–745.
[172] Cai, G.; Wang, B.; Hu, W.; Wang, T. Moving Target Defense: State of the Art and Characteristics. Frontiers of Information Technology & Electronic Engineering 2016, 17 (11), 1122–1153.
[173] Baudisch, P.; Cutrell, E.; Robertson, G. G. High-Density Cursor: A Visualization Technique That Helps Users Keep Track of Fast-Moving Mouse Cursors. In Interact; 2003; Vol. 3, pp 236–243.
[174] Staribratov, I., & Manolova, N. (2022). Application of mathematical models in graphic design. Math. Inform, 65, 72-81.
[175] Wang, S.; Chen, L.; Hu, H.; McDonald-Maier, K. Doorway Passing of an Intelligent Wheelchair by Dynamically Generating Bézier Curve Trajectory. In 2012 IEEE International Conference on Robotics and Biomimetics (ROBIO); 2012; pp 1206–1211. https://doi.org/10.1109/ROBIO.2012.6491134.

[176] WindMouse, an algorithm for generating human-like mouse motion | ben.land. https://ben.land/post/2021/04/25/windmouse-human-mouse-movement/ (accessed 2023-08-31).
[177] Bézier Curve. Understand the mathematics of Bézier… | by Omar Aflak | Towards Data Science. https://towardsdatascience.com/b%C3%A9zier-curve-bfffdadea212 (accessed 2023-05-02).
[178] Yang, S., Yu, Y., & Liu, Y. (2019). RTbust: Exploiting Temporal Patterns for Botnet Detection on Twitter. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (pp. 1733-1748).
[179] Folch, S. E.; Ibáñez, A. C.; Rabella, N. O.; Escrig, J. E. Web Bot Detection Using Mouse Movement. In 2023 JNIC Cybersecurity Conference (JNIC); IEEE, 2023; pp 1–6.
[180] Karim, F.; Majumdar, S.; Darabi, H.; Chen, S. LSTM Fully Convolutional Networks for Time Series Classification. IEEE Access 2018, 6, 1662–1669. https://doi.org/10.1109/ACCESS.2017.2779939.
[181] Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Networks. arXiv June 10, 2014. https://doi.org/10.48550/arXiv.1406.2661.
[182] Generative Adversarial Nets for Synthetic Time Series Data. Machine Learning for Trading. Retrieved https://stefan-jansen.github.io/machine-learning-for-trading/21_gans_for_synthetic_time_series/ (accessed 2023-05-28).
[183] Yoon, J., Jarrett, D., & van der Schaar, M. (2019). Time-series Generative Adversarial Networks. arXiv preprint arXiv:1902.04630.
[184] AWS Marketplace: Ubuntu 22.04 LTS - Jammy. https://aws.amazon.com/marketplace/pp/prodview-f2if34z3a4e3i (accessed 2023-05-29).
[185] Wang, W., Zheng, Y., Xing, X., Kwon, Y., Zhang, X., & Eugster, P. (2016, November). Webranz: Web page randomization for better advertisement delivery and web-bot prevention. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 205-216).
[186] Vikram, S., Yang, C., & Gu, G. (2013, October). Nomad: Towards non-intrusive moving-target defense against Web bots. In 2013 IEEE Conference on Communications and Network Security (CNS) (pp. 55-63). IEEE.
[187] Arapakis, I., & Leiva, L. A. (2020, July). Learning efficient representations of mouse movements to predict user attention. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 1309-1318).
[188] Jayalaxmi, P. L. S., Kumar, G., Saha, R., Conti, M., Kim, T. H., & Thomas, R. (2022). DeBot: A deep learning-based model for bot detection in industrial internet-of-things. Computers and Electrical Engineering, 102, 108214.
[189] Lindemann, B., Müller, T., Vietz, H., Jazdi, N., & Weyrich, M. (2021). A survey on long short-term memory networks for time series prediction. Procedia CIRP, 99, 650-655.
[190] Gupta, A. A Comprehensive Guide on Optimizers in Deep Learning. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/ (accessed 2023-10-26).
[191] Nguyen, H. D., Tran, K. P., Thomassey, S., & Hamad, M. (2021). Forecasting and Anomaly Detection approaches using LSTM and LSTM Autoencoder techniques with the applications in supply chain management. International Journal of Information Management, 57, 102282.
[192] Hermans, M., & Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. Advances in neural information processing systems, 26.
[193] Pascanu, R., Gulcehre, C., Cho, K., & Bengio, Y. (2013). How to construct deep recurrent neural networks. arXiv preprint arXiv:1312.6026.

[194] Graves, A., Mohamed, A. R., & Hinton, G. (2013, May). Speech recognition with deep recurrent neural networks. In 2013 IEEE international conference on acoustics, speech and signal processing (pp. 6645-6649). Ieee.

Appendix

Appendix A

Mouse Dynamics for User Authentication/Identification: Related Work

This section provides a brief survey of the works that focus on the use of mouse dynamics for the specific purpose of user authentication. We argue that these studies can provide valuable insights for the development of more effective bot detection mechanisms.

Chuda et al. [100] proposed a method for identity recognition based on mouse click features. The study uses three mouse click features to determine a user's identity: pause to click (time between click and last movement event), click duration (delay between pressing and releasing the mouse button), and pause after click (delay between click and next movement). The user's behavior is represented as a set of measured samples, and the model employs a classification algorithm based on distance measures adapted from the Kolmogorov-Smirnov non-parametric test. The distance between two user models is the sum of the differences for each feature. The experimental results indicate that the method can recognize users in small groups (e.g., those who share the same computer) and groups containing hundreds of users.

The article [101] proposes a user authentication method utilizing mouse dynamics behavior and a deep learning model to address insider threat attacks. The approach involves mapping basic mouse actions such as move, click, drag, scroll, and stay into images using a unique mapping method to preserve user-generated features. The resulting image dataset is used to train a 7-layer CNN network to generate classification models. Experimental results demonstrate the effectiveness of the proposed solution in accurately and efficiently performing continuous identity authentication on computer users. Compared to previous models, this method can authenticate users every seven seconds with a low false acceptance rate of 2.94% and a false rejection rate of 2.28%.

In 2017, Kaixin et al. [73] presented an identification and authentication approach based on different mouse movement operations that combines statistical and procedural features to characterize users' mouse behavior. They applied the Support Vector Machine (SVM) algorithm as the basis classifier to train the procedural and statistical features. Using data collected from students and teachers in a laboratory, they evaluated the performance of the proposed approach, which

showed effectiveness as a practical auxiliary authentication mechanism with a FAR of 8.8% and FRR of 5.5%.

Antal et al. presented a novel user authentication system in [74], using the SapiMouse dataset to train and evaluate the model. The system consists of two components: "enrollment," responsible for feature learning, and "verification," which performs the authentication. The authors used a Fully Convolutional Neural Network (FCN) for feature learning and a One-Class Support Vector Machine (OCSVM) for authentication. The authentication performance was evaluated, and the model achieved a best performance of 0.94 AUC for 15 seconds of data.

In [102], Chong et al. conducted a comprehensive study on the use of various deep learning architectures and mouse dynamics data for biometrics authentication systems. They combined convolutional and LSTM layers to build a hybrid model capable of modeling temporal sequences. Different techniques were applied to process data and generate mouse movement sequences, and the experimental evaluations showed that the multi-label 2D-CNN model outperformed other architectures, resulting in a 0.96 average AUC for the Balabit dataset [109].

Antal et al. [110] compared the performance of the DFL dataset with the Balabit and Chao Shen datasets for mouse dynamics-based user verification systems. They evaluated the effect of the quantity of training data and the number of consecutive mouse actions used for user identity predictions, using a Random Forest classifier. The study found that the Chao Shen dataset received the lowest AUC, likely due to the environmental condition of the study. To predict user identity, the authors suggest using approximately 1000 mouse actions for training. They also suggest that mouse dynamics should be considered as an additional security service in such systems, not a single verification indicator.

Appendix B

Figures 82 and 83 present visual results and the corresponding DTW distance values between two mouse trajectories: the original human and respective replay sessions produced by ReBot on the modified version of human-likebots.com. Although these samples serve as illustrative cases, we systematically replicated this comparative analysis across various trajectories spanning diverse users. Notably, our findings remained remarkably consistent across hundreds of such trajectories. This comprehensive comparison effectively confirms ReBot's proficiency in faithfully replicating the human sessions on the modified human-likebots.com platform.

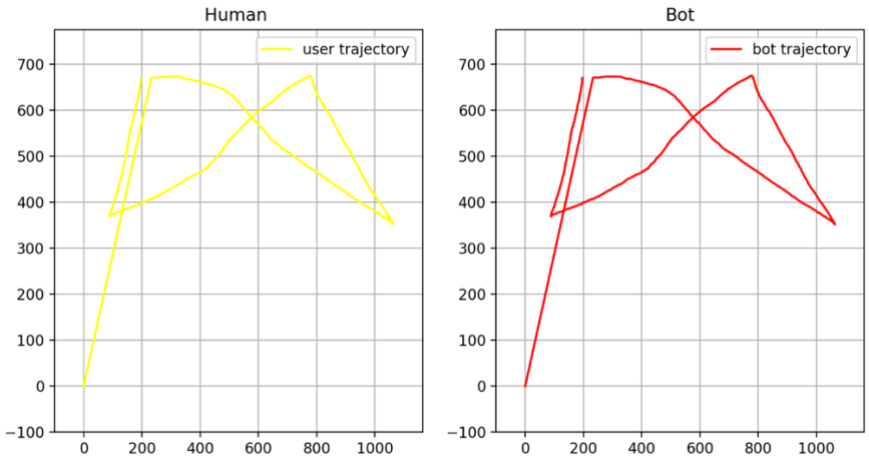


Figure 82. a) Human trajectory, and b) ReBot (replayed) trajectory.

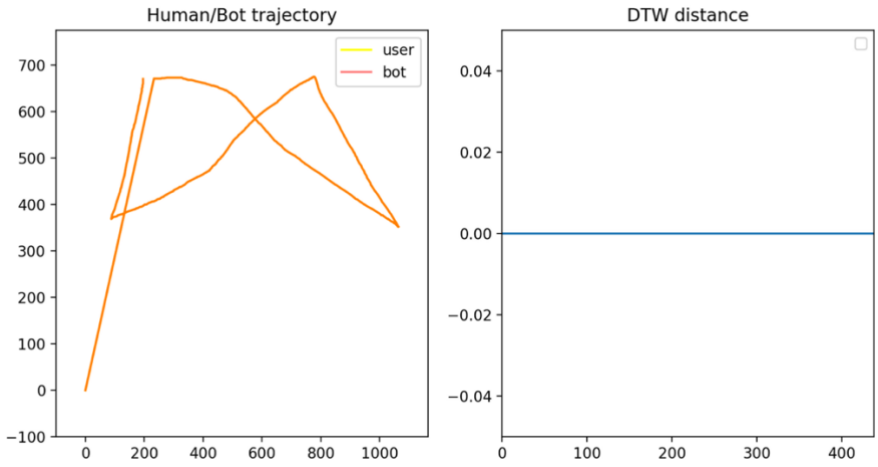


Figure 83. a) Human and ReBot trajectories, and b) DTW distance between the two trajectories.

Appendix C

In [178], LSTM was used to extract features from temporal patterns in Twitter data to detect botnets. The authors proposed a novel approach called RTbust that used LSTM to capture the temporal dynamics of Twitter activity, including user behavior and content, to differentiate between bot and human users. Arapakis et al. in [187] have used LSTM to extract features from mouse movement data as time-series to predict user attention. They proposed a model that used LSTM to capture the temporal patterns of mouse movements and attention levels and achieved better accuracy than traditional feature extraction methods. LSTM has also been used to extract features from time-series data of an industrial process to diagnose faults [170]. The authors in this study, proposed an approach that used LSTM to capture the temporal patterns in the data and extract informative features for fault diagnosis. The researchers in the area of user authentication have studied the usability of LSTM in extracting features from mouse movement data as time-series for continuous user authentication. In [170], the authors proposed a model that used LSTM to capture the temporal patterns of mouse movements and achieved higher accuracy than traditional authentication methods. To detect bot, the authors in [188] presents a deep learning-based bot detection model for the Industrial Internet-of-Things (IIoT). The proposed model, called DeBot, leverages LSTM networks as feature extractors to analyze the temporal patterns of sensor data collected from IIoT devices. The DeBot model consists of three main components: feature extraction, feature selection, and classification. The feature extraction module utilizes LSTM networks to capture temporal dependencies in the sensor data and extract relevant features. The feature selection module selects the most discriminative features using the Fisher criterion. Finally, the classification module uses a feedforward neural network to classify the input data as bot or non-bot traffic. The authors evaluate the proposed DeBot model on a real-world IIoT dataset, and the experimental results demonstrate that the model achieves high accuracy in detecting bot traffic.

Appendix D

The original LSTM model consists of a single hidden LSTM layer succeeded by a standard feedforward output layer. The stacked LSTM, an augmentation of this model, incorporates multiple hidden LSTM layers, each containing multiple memory cells. Stacking LSTM hidden layers imparts depth to the model, more accurately earning the description as a deep learning technique. The overall effectiveness of this approach in tackling a broad spectrum of demanding prediction problems can largely be attributed to the heightened depth of neural networks [192, 193].

Additional hidden layers can be integrated into a Multilayer Perceptron neural network to enhance its depth. These additional hidden layers are known to recombine the learned representation from prior layers and create new representations at high levels of abstraction [193]. A sufficiently expansive single hidden layer in a Multilayer Perceptron can serve to approximate most functions. Nonetheless, augmenting the network's depth offers an alternative solution that demands fewer neurons and facilitates faster training. In essence, increasing depth represents a form of representational optimization.

These advantages can also be leveraged with LSTMs. Considering that LSTMs are designed for sequence data, the incorporation of additional layers introduces varying levels of abstraction to the input observations as they unfold over time. This effectively involves organizing observations across time or representing the problem at different temporal scales [193].

Graves et al. [194] introduced Stacked LSTMs, or Deep LSTMs, in their groundbreaking work applying LSTMs to speech recognition. Notably, they surpassed a benchmark on a challenging standard problem. In their research, they discovered that the depth of the network held more significance than the number of memory cells in a given layer for effectively modeling skill.

This innovative approach has since evolved into a robust technique for addressing complex sequence prediction problems. A stacked LSTM architecture is characterized by an LSTM model composed of multiple LSTM layers. The distinctive feature is that an upper LSTM layer provides a sequence output for each input time step to the LSTM layer below, contrary to a single-value output. This nuanced architecture has proven to be a stable and effective solution for tackling intricate challenges in the realm of sequence prediction.

It is worth noting that empirical observations have shown that tacked LSTMs outperformed shallower architectures when it came to detecting RanReBot sessions.

Appendix E

In our research, we have consistently addressed the scenario where websites (i.e., online services) are susceptible to the generation of similar or repeated sessions by genuine human users. This commonly occurs on platforms such as news, banking, or gaming websites. However, we also need to consider situations where users only visit a website once, without engaging in any interactions. In such cases, if a skilled hacker possesses deep knowledge of the website's structure, they can employ sophisticated techniques to introduce extreme randomization into the user's session. This “extreme randomization”, (see Figure 77.c)), can make the user's behavior appear coherent and logical, thereby reducing the likelihood of detecting anomalous activity.

To tackle this challenge, we propose a solution that specifically addresses the issue of "extreme randomization." Our approach involves identifying and marking areas on the webpage that have a low probability of trajectory passing (referred to as Low Probability of Trajectory Passing or LPTP areas). Each of these marked areas is assigned a score. By aggregating the scores of these LPTP areas and comparing the total score against a predetermined threshold, we can determine whether a session is likely to be a bot-generated session.

To gain a better understanding of this concept, please refer to Figure 84, which highlights the importance of LPTP areas on the webpage. It is essential to note that the locations and characteristics of these areas may vary across different websites, as they are influenced by the unique structure and design of each platform.

By implementing this approach, our aim is to strengthen the detection capabilities of our system, especially in scenarios involving extreme randomization. This additional layer of defense enables us to identify and mitigate potential threats posed by session-replay bot attacks, even on websites that exhibit distinct and unpredictable user behavior patterns.

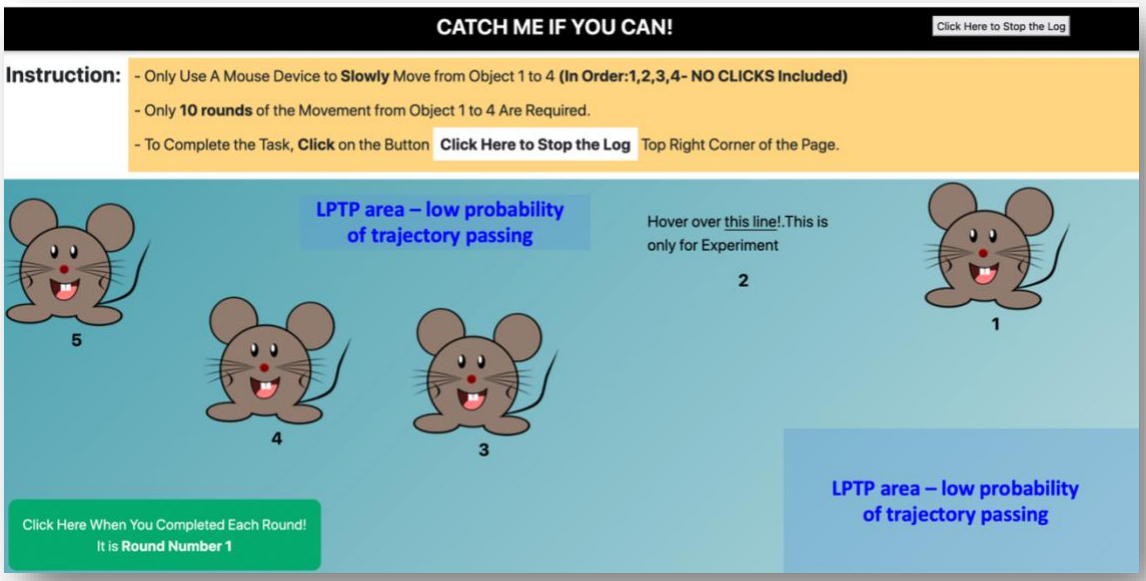


Figure 84. Detection of 'extreme randomization' using Low Probability of Trajectory Passing (LPTP) areas on a webpage.