Detecting IMAP Credential Stuffing Bots Using Behavioural Biometrics

by

Ashley Barkworth and Rehnuma Tabassum

A MACSEC CAPSTONE PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Cybersecurity

in the Faculty of Computer Science &
Canadian Institute for Cybersecurity

Supervisor: Arash Habibi Lashkari, PhD

THE UNIVERSITY OF NEW BRUNSWICK

August 2021

Abstract

While credential stuffing has been around for some time, it has seen a great uptick in use and is now one of the most common types of cyberattacks. Legacy email protocols like the Internet Mail Access Protocol (IMAP) are particularly vulnerable to this kind of attack as they do not support multi-factor authentication (MFA). We propose a supervised learning system that detects credential stuffing bots using two kinds of behavioural biometrics: mouse and keystroke dynamics. The system records a user's mouse and keystroke events while they complete three tasks in a graphical user interface (GUI) application. To test our system, we developed two types of bots: a simple bot which makes no attempt to appear human, and an advanced bot that uses techniques to simulate human-like mouse and keyboard motions. We evaluated our system using the Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) algorithms and compared them against two data sets: Simple containing human and simple bot data, and Advanced containing human and advanced bot data. The highest accuracy against the Simple and Advanced data sets were both 96.95% but achieved by the KNN and RF classifiers, respectively. The RF classifier showed the best overall performance, achieving the highest precision and mean AUC against the Simple data set and the highest scores across all metrics against the Advanced data set. Our results show that bot detection using mouse and keystroke dynamics is an effective solution as part of a layered defence against credential stuffing bots.

Acknowledgements

We would like to thank Dr. Arash Habibi Lashkari for his immense support, patience and guidance throughout our master's journey. His immense knowledge and ample experience have encouraged us in all the time of our research. We would also like to thank all the faculty members who shaped us to be where we are today.

Lastly, we would like to show our gratitude to Bell Canada for providing us with the opportunity to carry out the research through the Bell Research Intensive Cyber Knowledge Studies Program(BRICKS).

Table of Contents

List of Tables

List of Figures

List of Symbols, Nomenclature or Abbreviations

IMAP	Internet Mail Access Protocol
MFA	Multi-factor authentication
GUI	Graphical user interface
CSV	Comma separated values
ML	Machine learning
MM	Mouse move
PC	Point and click
DD	Drag and drop
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
ANN	Artificial neural network
SVM	Support vector machine
KNN	K-Nearest Neighbors
LVQ	Learning vector quantization
CNN	Convolutional neural network
LSTM	Long short-term model
RNN	Recurrent neural network
MLP	Multi-layer perceptron
GBT	gradient boosted trees
XGBT	extreme gradient boosted trees
RBF	Radial basis function
LPC	Linear predictive coding
EER	Equal error rate
FAR	False alarm rate
FRR	False rejection rate
AUC	Area under the ROC curve
TP	True positives
FP	False positives
TN	True negatives
FN	False negatives

Chapter 1

Introduction

1.1 Background

Credential stuffing is a type of attack that obtains stolen account credentials and attempts to "stuff" these credentials into a large number of other account logins, with the aim of successfully logging into and compromising users' accounts. Once an account is successfully compromised, attackers can carry out different illegal activities such as e-commerce fraud, selling access to compromised accounts, corporate/institutional espionage, and theft. To carry out credential stuffing attacks, cybercriminals use botnets that automate the process of trying several credentials on multiple sites at once. The stolen credentials are sourced from prior data breaches and often acquired through phishing campaigns. Unlike brute-force attacks, where passwords are guessed, credential stuffing makes the educated gamble that users are often so overwhelmed with the number of logins they have (50-200 on average) that they resort to reusing passwords across multiple accounts to lower the burden of memorization [1, 2, 3]. This is an educated gamble: according to a study commissioned by Telesign, 73% of online accounts use duplicated passwords, and 54% of consumers use five or fewer passwords across their entire life, while 22% of users use just three passwords

or fewer [4]. Credential stuffing is now the most prevalent form of account takeover and is remarkably commonplace; there were 193 billion credential stuffing attempts globally during 2020, according to Akamai's '2021 State of the Internet' report [5, 6].

Credential stuffing has become a great matter of concern for the Internet Mail Access Protocol (IMAP). IMAP is a popular method for accessing electronic mail and news messages maintained on a remote server. This protocol is specially designed for users who need to view email messages from different computers since all management tasks are executed remotely without transferring the messages back and forth between these computers and the server. A client program can manipulate remote message folders (mailboxes) in a way that is functionally equivalent to local folders. A major vulnerability in IMAP, along with other legacy email protocols, is that it cannot support MFA and depends on only a username and password for authentication, leaving it susceptible to credential stuffing.

1.2 Problem Statement

Security companies have noticed a surge in the number of credential stuffing attacks targeting email servers like G Suite and Office 365 [7]. A report by Proofpoint revealed that approximately 60% of all Microsoft Office 365 and G Suite tenants had been targeted using IMAP-based password-spraying attacks and, as a direct result, approximately 25% of G Suite and Office 365 tenants that were attacked also experienced a successful breach [8]. Attackers exploit the lack of MFA support in IMAP, as well as the fact that IMAP is on by default in these servers, to bypass the multifactor controls on these systems and compromise users' email accounts.

1.3 Current Solutions and Limitations

Rate limiting has traditionally been employed to stop checking multiple accounts from a single IP address [9]. In unsophisticated attacks, login attempts come from a small set of IP addresses from unexpected regions. These IP addresses can be blacklisted using firewalls or other defences, so rate limiting is effective. However, sophisticated attacks use thousands of IP addresses or proxies to circumvent rate limiting [10]. For example, attackers will frequently use residential IP addresses, which are less susceptible to rate limiting and blacklists for fear of denying service to residential users [11]. Further, a previous study of 182 of the Alexa Top 500 websites found that 131 out of the 182 websites did not properly implement rate-limiting mechanisms; the sites allowed unlimited login attempts if the attacker adjusted the time interval between consecutive login attempts or repeatedly changed IP addresses [12].

Requiring an additional factor for authentication (e.g., two-factor authentication (2FA), MFA) is another common approach but is unsupported by IMAP. Moreover, users are generally reluctant to adopt 2FA and MFA. Lastly, MFA is not foolproof. For example, along with legacy email protocols, another MFA bypass technique is the installation of malicious Azure/O365 OAuth apps. Attacks in June 2020 on the Australian government and businesses leveraged OAuth, a standard technique used for access delegation in apps, to gain unauthorized access to cloud accounts such as Office 365. The attackers developed a malicious Office 365 application and sent it to target users as part of a spear phishing link. On receipt, the malicious app convinced the victim to grant permission to access data in the user's account, most notably: offline access, user profile information, and the ability to read, move and delete emails [10, 13].

Another defense against credential stuffing involves detecting password reuse across accounts and sending password reuse notifications. However, users are re-

luctant to stop password reuse even after repeated warnings to do so [3]. Users are similarly resistant to adopting password managers; a 2019 Google/Harris Poll survey of 3000 U.S. adults found that only 24% of respondents reported using a password manager [6].

1.4 Our Contribution

With the insight that credential stuffing attacks are carried out by bots, a promising countermeasure is to identify and block these bots before they can login. Biometrics can be used to establish a user's identity online based on their physical or behavioural traits. Physical biometrics uses innate human characteristics such as fingerprint, iris, retina, facial, and voice patterns and has been studied previously for intrusion detection systems, but it is relatively intrusive and may require installing additional devices such as scanners. Behavioural biometrics, which refers to the measure of uniquely identifying and measurable patterns in human activities, is not as intrusive and has attracted increasing attention in the security community [14]. Two types of behavioural biometrics to characterize a user's interaction with the computer are keystroke and mouse dynamics. While past research has explored using keystrokes and mouse dynamics for user authentication, applying them for bot detection is a relatively unexplored area [15].

The objective of our research is to prove that mouse and keystrokes dynamics can discriminate between humans and bots. We do this by developing a system which collects mouse and keystroke data from users while they complete three separate activities in a graphical user interface (GUI). This data is parsed to extract a total of 131 keystroke and mouse features. These which are subsequently used as input for supervised machine learning (ML) algorithms that classify each user's feature vector as either human or bot. Our system can be integrated into the login process of email

clients to stop credential stuffing bots before they compromise accounts.

1.5 Thesis Organization

The rest of our paper is organized as follows: Chapter 2 summarizes related work on mouse and keystroke dynamics. In addition, it surveys different bot detection techniques, including ML, which is one of the widely used solutions. Moreover, we summarize the mouse and keystroke dynamics at the end of the respective sections.

Chapter 3 describes the general overview of our system, followed by a description of each of its components, which are the data collection, feature extraction, feature calculation, and classification phases. In the explanation of the system's classification component, our four chosen classification algorithms are described along with their advantages to clarify the purpose of choosing those particular algorithms for our system. Finally, the implementation and testing of our bot detector solution, along with the testing results and analysis, are described in detail. Chapter 4 gives our conclusions and suggests future work.

Chapter 2

Literature Review

2.1 Mouse Dynamics

Mouse dynamics are the patterns and characteristics obtained from a user's interactions with their computer mouse through their movements, scrolls, and clicks. Early works utilized mouse dynamics for user authentication [16, 17, 18, 19, 20, 21]. Researchers have recently harnessed mouse dynamics for other applications such as stress detection [22, 23], attention detection [24], and bot detection [25].

Data for research on mouse dynamics come in two forms. The first form is specific mouse usage data collected in a specific application such as web browsers or applications that require users to perform certain tasks [26]. Gamboa et al. [16] recorded mouse data from their interactions with a memory game. Sayed et al. [20] gathered mouse data from users while they recreated gestures in a drawing application. Antal et al. [15] performed data collection in a JavaScript web application asking users to perform different actions, each associated with geometric shapes. Mohamed and Saxena [21] developed a dynamic cognitive game where users drag and drop objects into target locations. The second form of data is general mouse usage data transparently recorded from users conducting their daily online activities rather

than specific tasks [14]. Shen et al. [18] developed a data collection software program that runs as a background task and records users during their routine computing activities such as Internet surfing, word processing, online chatting, programming, and online gaming. Similarly, Ahmed et al. [17] implemented data collection software to record mouse activity on users' machines while conducting their usual activities without any restriction.

Mouse activity is recorded as a sequence of events such as mouse moves or drags, mouse button presses, mouse button releases, and wheel scrolls. These event sequences are referred to as raw data. In most papers, each event is recorded with three fields: the x and y coordinates of the mouse pointer and the elapsed time since the start time of the recording session. Both [14] and [15] recorded two additional fields describing the current condition of the button and the current state of the mouse. Gamboa and Fred [16] recorded an additional field containing the event type (e.g. mouse move, mouse click).

There are two approaches to processing raw data. The first approach forms logical, semantically meaningful actions from groups of events. In Gamboa and Fred [16], consecutive movements between clicks are grouped together and defined as strokes. Ahmed and Traore [17] aggregated events into four types of actions: mouse move (MM), defined as general mouse movement; point and click (PC), defined as mouse movement followed by a click or double click; and drag and drop (DD), which starts with the mouse button down, followed by movement, and ends with the mouse button up; and silence, defined by no mouse movement. The papers [14] and [26] used the MM, PC, and DD actions as defined in [17]. Shen et al. [18] used the PC, MM, DD, and silence actions and added single-click and double-click actions, which are defined as single and double mouse clicks without any prior movement, respectively. The second approach to processing segments events into fixed-size blocks. Chong et al. [27] grouped all events within a fixed time window, while Antal et al. [15] formed

blocks of 128 events.

After processing, features are extracted from each action or block. Features used in previous works include: duration/elapsed time [16, 17, 19, 20, 21, 26, 14, 25], distance [16, 17, 21, 14, 26, 15, 22, 27, 23, 25], velocity or speed [16, 17, 18, 19, 20, 21, 14, 26, 15, 27, 23, 25], acceleration [16, 18, 19, 20, 21, 14, 26, 23], jerk [16, 20, 14, 26], angle [16, 20, 21, 14, 26, 25], curvature [16, 20, 14, 26], angular velocity [16, 14, 26], straightness [16, 14, 26, 25], and click time [16, 18, 19]. In addition to feature extraction from actions/blocks, features may be extracted over an entire user session, such as the total duration and the total number of actions.

The next step after feature extraction is to feed them as input into a model. This model may be a statistical model. Gamboa and Fred [16] used Parzen density estimation and a unimodal distribution model. Kim et al. [22] used linear predictive coding (LPC). However, the more common approach is to apply a ML model; popular choices include support vector machines (SVM), k-nearest neighbors (KNN), long short-term models (LSTM), and Random Forest. The output of these models depends on what it is being applied for. For intrusion detection systems, the output may be an outlier or anomaly score, whereas user authentication and bot detection systems may output a classification; this classification may be the user's identity (for authentication) or a binary classification of bot or human (for bot detection). Upon obtaining this output, a system can make a decision on the user in question, e.g., whether to grant or deny them access to a service. Table 2.1 summarizes the related works on mouse dynamics and their results, which are either the false acceptance rate (FAR) and false rejection rate (FRR), the equal error rate (ERR), the area under the receiver operating curve (ROC) curve (AUC), or the accuracy.

Table 2.1: Overview of related works on mouse dynamics

Paper	Year	Application	No. Features	Data Collection	Model	Results
[16]	2004	Authentication	63	Specific	Statistical	EER: 0.2%
[17]	2007	Authentication	39	General	ANN	EER: 2.46%
[18]	2012	Authentication	92	General	SVM, ANN, KNN	FAR: 0.37% FRR: 1.12%
[19]	2013	Authentication	74	Specific	SVM, ANN, KNN	FAR: 8.74% FRR: 7.69%
[20]	2013	Authentication	12	Specific	LVQ neural network	FAR: 5.26% FRR: 4.59%
[21]	2016	Authentication	64	Specific	Random Forest	FAR: 2% FRR: 2%
[26]	2019	Authentication	39	Public data sets	Random Forest	AUC: $0.92 - 0.99$
[14]	2019	Authentication	39	Public data set	Random Forest	AUC: 0.92
[22]	2020	Stress detection	31	Public data set	LPC, CNN, LSTM	Accuracy: 62.47%
[23]	2020	Stress detection	22	Specific	Random Forest	Accuracy: 63%
[24]	2020	Attention detection	2	Specific	RNN, LSTM, CNN	AUC: 0.739 F_1: 0.731
[27]	2020	Authentication	2	Public data sets	CNN, LSTM, SVM	AUC: $0.93 - 0.96$ EER: $0.1 - 0.13$
[25]	2021	Bot detection	6	Specific	Random Forest, SVM, KNN	Accuracy: 93%

2.2 Keystroke Dynamics

Keystroke dynamics is another behavioural trait in biometrics that targets recognizing the patterns of rhythm and timing-based features created when a user types something. These features can include the typing speed, the time between pressing and releasing different keys, the pressure exerted when typing, and the hand postures during typing. Similar to mouse dynamics, earlier research works also used keystroke dynamics for the authentication purposes. For example, a user's keystroke analysis can help identify intruders [28] or predict users' educational levels [29]. Recently, amidst the global pandemic, Morales et al. [30] proposed fighting fake news propagation using keystroke biometrics for content de-anonymization. Bergadnano et al. [31] discussed how keystroke dynamics analysis could be used for various purposes such as strong authentication, identity confirmation, and user identification and tracking over the internet with a FAR of 4% and Imposter Pass Rate(IPR) of

less than 0.1%. Similar work has been conducted on user authentication [31, 32], where they extracted keystroke features to analyze, identify and authenticate the user using learning algorithms such as SVM [32] and the error back propagation algorithm in an ANN [33].

Keystroke dynamics can be divided into two primary types: free-text and fixed-text. Fixed-text utilizes the user's typing patterns by entering a preset text such as a username or password, whereas free-text uses the user's typing patterns by entering a predefined text such as writing an email or transcribing a sentence with typing errors. Lu et al. [34] proposed a model leveraging CNNs and RNNs to learn the keystroke data obtained from free text to carry out continuous authentication. Their model produced a FRR of 1.89%, FAR of 2.83%, and EER of 2.36%. On the other hand, Krishnamoorthy et al. [32] leveraged fixed text to classify the behaviour of users accessing computer devices in order to authenticate them. They chose the fixed word '.tie5Roanl' and recorded the typing patterns of 94 users to carry out their research. Solano et al. presented research on behavioural biometrics as well, which is used to develop a Risk-based authentication model for login authentication in web applications. They used both mouse and keystroke dynamics for static authentication using the Random Forest classifier to discriminate legitimate users from attackers. Their model achieved an FRR of 10.73% and a FAR of 23.34%.

For the research, data acquisition is done using either a specific application developed solely for data collection, such as an online application [32, 35], a webpage including a text-based CAPTCHA image with a text box [36]; or collected from daily online activities, such as using webpage-embedded logger in the header of a webpage [37]. The keystroke data consists of a sequence of time-based events where each event contains the timestamp, such as key pressed and key released events. Later, several features are extracted from those sequences of events. Mostly four features are calculated from each sequence: hold latency, inter-key latency, press latency, and

release latency [36, 37, 25].

After data collection, the extracted features are fed into the model developed. As discussed earlier in mouse dynamics, these models can either be a statistical model or a ML model. The ML model can be supervised and unsupervised, including different categories such as regression, classification and anomaly detection. The evaluation of these models depends on the metrics used, such as FAR, FRR, ERR, IPR etc. Table 2.2 shows the summary of the related works done on keystroke dynamics, models used, the results of the works and the limitations of each paper.

Table 2.2: Overview of related works on keystroke dynamics

Paper	**Year**	**Application**	**No. Features**	**Model**	**Results**	**Limitations**
[30]	2020	Reduce Propagation of Fake news	4 temporal	RNN	Accuracy: 95 %	No bot detection
[29]	2018	Educational level of users	2 temporal features, 163 calculated features	Radial Basis Neural Network	Accuracy: 85 %	Longer time to build model
[28]	2017	Dataset to detect intruders	-	-	-	No detailed analysis
[31]	2012	Authentication	6	Statistical analysis	FAR: 4% IPR: 0.01%	Cannot be used for password based authentication
[32]	2018	Authentication	9 Categorical Features	SVM	Accuracy: 97.40% F1 Score: 97.01%	Only one classification algorithm used
[33]	2018	Authentication	7	ANN, Error backpropagation	Accuracy: 69%	Less accuracy
[34]	2020	Authentication	4 feature sequence	CNN, RNN	FRR: 6.61% FAR: 5.31% EER: 5.97%	Small amount of data
[37]	2017	Bot detection	2	C4.5	Accuracy: 99%	Prone to bot attack
[35]	2020	Software bot detection	4	Logistic regression	Accuracy: 93.33%	Limited number of users, simple fixed text
[36]	2019	Bot detection	3	Euclidean Distance proposed	-	Prone to automated attacks
[38]	2020	Authentication	4	Random Forest	FAR: 23.34% FRR: 10.73%	Data collected from experienced users with keyboards. Unaccounted novice users

2.3 Bot Detection Techniques

Bots are automated scripts or programs to carry out activities by software rather than humans. They can be benign and useful, such as search engine crawlers that provide easy access to content from the Internet. However, attackers wield bots to carry out a variety of malicious actions, including credential stuffing. In recent years, Botnet identification has become a prominent study issue. Botnet identification and tracking have been proposed using a variety of techniques and methodologies.

CAPTCHAs, a widely used bot detection mechanism, are challenge-response tests designed to be simple for humans but difficult for bots to solve. Researchers have examined the effectiveness of CAPTCHAs and the attacks against them. Xu et al. [39] provided a review on the development of CAPTCHA technology and discussed CAPTCHA mechanisms from a security and usability aspect. The authors classified the most common existing types of CAPTCHAs as text-based, image-based, and sound-based. They also studied the techniques that bots use to break CAPTCHAs. These techniques included object segmentation, object recognition, and dictionary attacks. The authors ultimately concluded that designing CAPTCHAs that are both usable and secure has become increasingly difficult. Further, they found that most existing CAPTCHAs are vulnerable to cracking due to advancements in ML and deep learning that attackers exploit. Similarly, Guerar et al. [40] investigated the evolution of CAPTCHAs over the past two decades and summarized the different techniques by bots used to break them. The authors found that bots are continuously evolving to break these CAPTCHA schemes. Through advancements in ML and computer vision (CV) bots are now able to solve image-based CAPTCHAs using object and pattern recognition. Bots also build up dictionaries of the correct solutions in order to defeat cognitive CAPTCHAs. Mohamed et al. [41] implemented a dynamic cognitive game CAPTCHA that challenges the user to perform a game-like cognitive task while interacting with a series of dynamic images.

They dissected these CAPTCHAs against two types of fully automated attacks: randomized attacks that attempt solving the them by random guessing, and attacks that use object recognition methods to locate target objects and regions and build up dictionaries of the correct solutions. Their results showed that dynamic cognitive game CAPTCHAs are secure against randomized attacks but are vulnerable to dictionary-based attacks.

Most existing bot detection solutions also use browser fingerprinting as part of their bot detection logic. Browser (or device) fingerprinting gathers and analyzes characteristics unique to a device, including: device information like the hardware and operating system, browser information like browser history and active plugins, and networking information like the origin IP and geolocation [42]. Amin Azad et al. [43] investigated the design and implementation of 15 popular commercial anti-bot services and evaluated their ability to stop attacks, include credential stuffing. Through whitebox analysis of these services' source code, the authors found that all but one service, Cloudflare, used browser fingerprinting. Yet, according to Nathan [44], tech-savvy cybercriminals have been able to leverage stolen user data (e.g. account credentials, cookies, and browser user agents) obtained from keylogging trojans as well as residential IP addresses for their bots to appear innocuous and evade detection from browser fingerprinting methods.

The most popular bot detection techniques used employ honeynets, communication signature detection and detection of abnormal behaviour according to [45]. Honeynets are deployed to collect information from bots in a controlled environment to analyze different characteristics of the bot and the payload of their attack. Additionally, this information is used to discover the command and control (C&C) system, unknown susceptibilities, and other information. Although the honeynet method has a very high accuracy for botnet detection, it cannot effectively detect unknown bots and is useless in real-time systems. Communication signature detec-

tion is the second and most commonly used technique for botnet detection. The most common methods include whitelisting or blacklisting IPs, regular expressions, and n-gram models. In this technique, when a new bot is detected, the patterns are matched with pre-defined patterns and signatures collected from well-known bots. Nevertheless, this method is prone to obfuscation technology as most bots can avoid signature-based detection. The third method is anomaly-based detection, based on the basic idea of detecting the host behaviour or network traffic abnormalities using ML, deep learning, or other statistical analysis.

Several prior works have proposed ML bot detection solutions, each with different features and for different types of bots. Lagopoulos et al. [46] proposed a supervised learning approach to detect web robots on academic publishing websites, based on 13 features from hyper-text transfer protocol (HTTP) requests to web servers such as the total number of requests and time between requests. They applied four models: a SVM with a RBF kernel, a multi-layer perceptron (MLP), gradient boosted trees (GBT), and extreme gradient boosted trees (XGBT). Shi et al. [47] applied semi-supervised clustering to detect social bots based on a user's clickstreams, i.e., the order of clicks when visiting websites. Rather than using quantitative features that can be imitated by bots, they analyzed user behaviour features and identified transition probability features between user clickstreams. The authors compared their approach with a supervised SVM model to verify its efficiency. Tsikerdekis et al. [48] constructed a deep learning model to detect bots in video games based on a combination of game and mouse movement based features. Acien et al. [25] proposed BECAPTCHA, a CAPTCHA model for mobile phones that determines if users are humans or bots based on touchscreen swipe (i.e., drag and drop) gestures. They evaluated their model using synthesized swipe gestures generated by generative adversarial neural networks, and applied SVM, KNN, and Random Forest models to classify six swipe gesture features - duration, distance, displacement,

Table 2.3: Overview of existing ML bot detection solutions

Paper	**Year**	**Type of Bot**	**No. Features (Type of Feature)**	**Model / Algorithm**	**Results**
[46]	2017	Web robots on academic publishing sites	13 *(HTTP requests)*	SVM, MLP, GBT, XGBT	91.81% F1 score 91.33% accuracy 91.23% G-mean
[47]	2019	Social media bots	11 (*quantitative statistics & clickstream sequences*)	Semi-supervised clustering	93.1% precision 97.5% recall
[48]	2020	Video game bots	4 *(game actions & mouse movements)*	LSTM	98.81% precision 99.08% recall 98.94% F_1 score
[25]	2021	Bots on mobile phones	6 (*touchscreen dynamics*)	SVM, KNN, Random Forest	93% accuracy

angle, mean velocity, move efficiency - as human or bot. Table 2.3 summarizes the existing bot detection solutions using ML.

Chapter 3

Design, Implementation, and Testing

3.1 System Design

In this chapter we propose a scalable bot detection technique using supervised machine learning to address credential stuffing attacks on IMAP, which are mostly carried out by botnets. The proposed system can be used as a security layer in organizations trying to prevent attacks carried out by bots. Most modern credential stuffing software evades security by employing thousands of bots to attempt multiple logins across multiple platforms using same username with different passwords from various device kinds and IP address at the same time. Although, the browser type or the IP address can be spoofed easily, the behavioral traits cannot be changed without difficulty. The general approach of the proposed system is outlined below in Figure 3.1. The system's workflow can be divided into four phases:

1. Data Collection: While a user interacts with an application, their mouse and keyboard activity is recorded as a sequence of events. Each user's mouse and keyboard events are saved to two distinct CSV files.

2. Feature Extraction: User mouse and keyboard event data is parsed separately to extract various mouse and keystroke features. For each feature, a set of values is extracted over the user's entire mouse or keyboard session.

3. Feature Calculation: The average, standard deviation, maximum, and minimum values are calculated for each extracted feature set. The calculated mouse and keystroke feature values are combined together to form a single feature row representing a single user.

4. Classification: The feature rows of each user are fed into classification algorithms in order to build an ML model that classifies users as humans or bots.

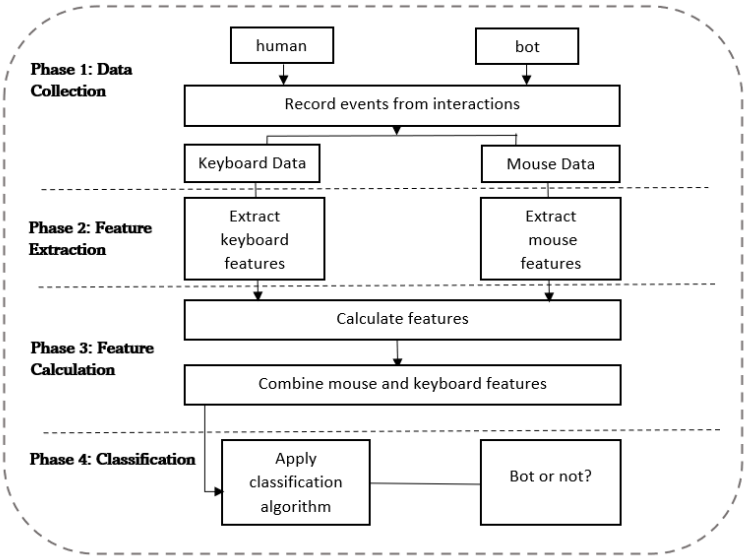


Figure 3.1: Overview of our proposed system

3.1.1 Data Collection Phase

We developed a graphical user interface (GUI) application to record mouse and keyboard events from users. Our application contains three activities: a typing activity (Figure 3.2) for recording keyboard data, as well as a ball activity (Figure 3.3) and sorting activity (Figure 3.4) for recording mouse data.

To record keystrokes, we chose fixed-text keystrokes because of their practical advantages. Service providers widely use fixed-text keystroke dynamics solutions to validate a user's identification while they input their login and password. These solutions, in turn, aid in the prevention of credential theft, credential leakage, and brute-force assaults. The typing activity page asks the user to type the word '123CA-Pabc!' 10 times. The word selected is a combination of uppercase letters, lowercase letters, numbers, and special characters. Nowadays, most login platforms only accept combination passwords as they are considered strong. As a result, most users choose passwords like this. In order to make our data more realistic, we came up with the idea of choosing this combination. The other aspect of the typing activity is typing the word 10 times. This is because the fixed-text keystroke dynamics approach is usually applied to a short text sequence, and the text is repeated for a fixed number of times for every user in order to build an accurate model. While a user types, each key press and release event are recorded. Each key event record contains the time that the event occurred in seconds and the key that was pressed or released.

The first mouse activity is the ball activity, where users are instructed to click on a ball 10 times. Upon being clicked, the ball reappears in a new, random location on screen. After completing the ball activity, users are shown the sorting activity, which requires them to drag images of four animals (cat, dog, beaver, and monkey) and four fruits (apple, banana, strawberry, and orange) to the correct labelled box. As a user completes both activities, their mouse activity is recorded as a sequence of events. Each mouse event record contains the time that the event occurred in seconds; the x and y coordinates of the mouse pointer; the condition of the mouse button; and the state of the mouse or mouse button.

The GUI can be completed by humans to collect human biometrics data. In order to collect bot biometrics data, we coded bot scripts to automatically complete the GUI activities. To compare our bot detection system against attacks with differ-

ent levels of sophistication, we implemented two types of bots: a simple bot and an advanced bot. The simple bot does not try to mimic humans in any way, whereas the advanced bot uses techniques to simulate human-like movements/actions. There are several key differences between the design of the two bots which are based on prior research. First, the simple bot's mouse movements are very fast; this is because when programming a bot, the easiest way to move the mouse is by giving the destination coordinates, which results in nearly instantaneous mouse movements where the mouse jumps between the start and end locations [49]. In contrast, the advanced bot moves more slowly and smoothly. Second, the simple bot moves the mouse in straight lines with zero acceleration (i.e., constant speed except when changing directions), which [37] and [49] found that many bots do. In contrast, the advanced bot's mouse trajectories are defined by Bézier curves. Bézier curves are parametric curves defined by a set of points and are commonly used in computer graphics to draw shapes, as well as in prior research to generate mouse trajectories[50]. When Bézier curves are used to create mouse paths, it becomes much harder to differentiate human mouse movements from bots, both visually and by using detection rules [49]. Lastly, the simple bot has zero randomness or variance in its movements. For example, the simple bot enters keys with even intervals and performs actions without zero delay in between. In constrast, the advanced bot is programmed with random time delays between entering keys; random delays after entering a word; random speeds when moving or dragging the mouse; and random delays between pressing and releasing the mouse button when clicking.

3.1.2 Feature Extraction Phase

After a user (human or bot) finishes interacting with the GUI and their keyboard and mouse event data have been saved to CSV files, the events are processed to extract the user's keystroke and mouse feature values. Four temporal features are extracted from

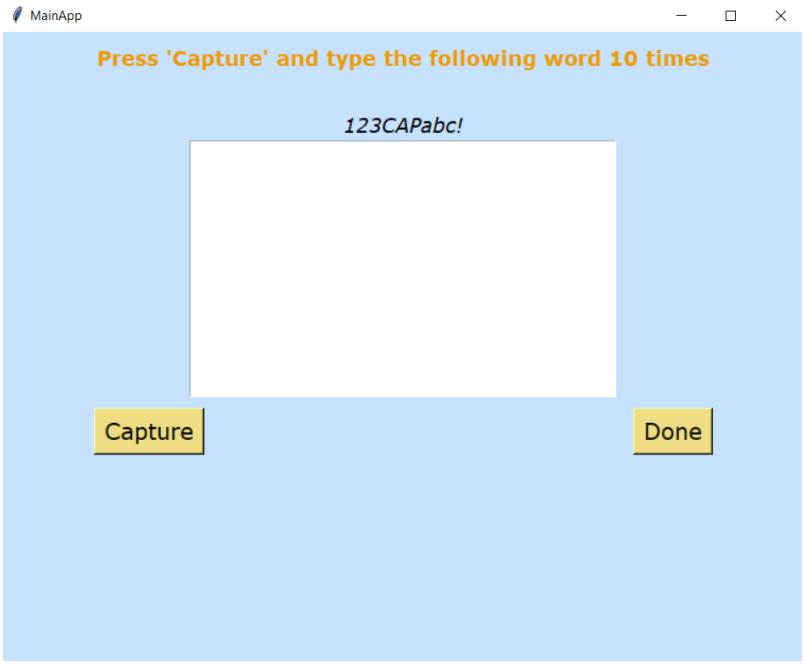


Figure 3.2: Typing activity page

Table 3.1: Features extracted from keystrokes

Name	**Description**
Hold time	The elapsed time between a press and release of the same key
Inter-key latency	The elapsed time between consecutive press and release or release and press events
Press time	The elapsed time between two consecutive press events
Release time	The elapsed time between two consecutive release events

keystroke data: hold time, inter-key latency, press time, and release time. Table 3.1 presents an overview of the keystroke features with their descriptions. According to [30], these four features are used for both free-text and fixed-text keystroke systems. The respective key codes are also added with the extracted features.

To extract mouse features, the mouse events are first parsed and grouped to form actions. Following [14] and [26], we group events into three types of actions: mouse move (MM), point and click (PC), and drag and drop (DD). Each action is defined by the action type and its set of events. Further, the entire set of a user's mouse actions are grouped together and defined as a session. For each action in

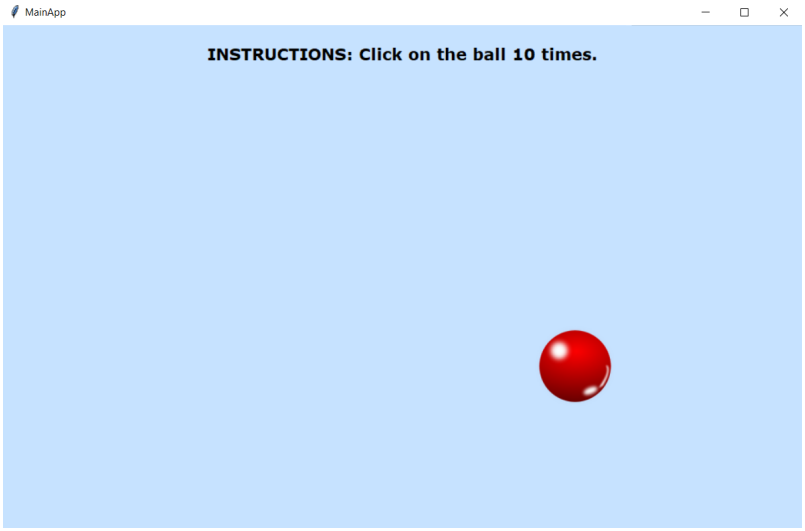


Figure 3.3: Ball activity page

a user's session, ten features are extracted: tangential velocity, acceleration, jerk, duration, straightness, number of events, curvature, the sum of angles, maximum deviation, and click time. For the tangential velocity, acceleration, jerk, and curvature, the value extracted is the average over the action's set of events, e.g., the average tangential velocity between consecutive events. The first nine features are derived using the same definitions from [14], where each action can be represented as a sequence of triplets $(x_i, y_i, t_i), i = 1 \dots n$, where n is the number of events in the action. Some preliminary computations are made before feature extraction. They include the angle of the path tangent with the x-axis, calculated using Equation 3.1 and the path length of the mouse's trajectory, calculated with Equation 3.2:

$$\Theta_i = \text{atan2}\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right), i = 2, \dots, n \tag{3.1}$$

$$s_i = \sum_{k=1}^{i} \sqrt{(x_k - x_{k-1})^2 + (y_k - y_{k-1})^2} \tag{3.2}$$

The click time is extracted for PC actions and is defined as the elapsed time between the mouse button press and mouse button release events. An overview

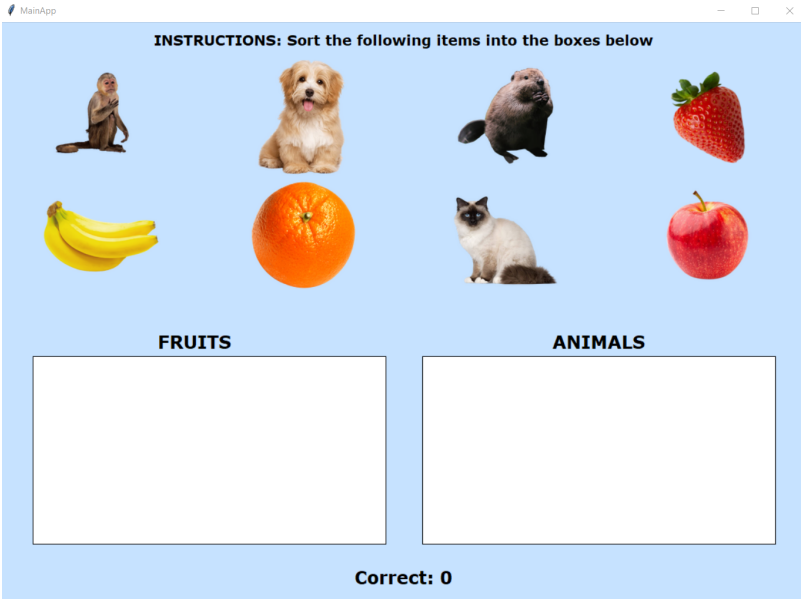


Figure 3.4: Sorting activity page

of the action-based mouse features, including their definitions and descriptions, are presented in Table 3.2.

3.1.3 Feature Calculation Phase

After computing the set of features values for each mouse and keystroke feature, the average, standard deviation, maximum, and minimum values are calculated over each set. For keystroke features, this includes the set of hold times, inter-key latencies, press times, and release times. For mouse features, all actions in a user session are first grouped by action type (MM, PC, DD), and the values are calculated over each group of actions. There is one additional feature calculated for keystrokes: the total time taken for typing. There are two additional features calculated for the mouse: the total time taken to complete both mouse activities and the total number of mouse actions. A total of 131 features are calculated from keystroke and mouse movements. Finally, each user's calculated keystroke and mouse features are combined together to form the user's feature row, which is fed as input into the classifiers to distinguish between bots and humans.

Table 3.2: Features extracted from a single mouse action

Name	Definition	Description
Tangential velocity	$v = \sqrt{v_x^2 + v_y^2}$	Rate of change in distance w.r.t. time
Acceleration	$a = \frac{dv}{dt}$	Rate of change in velocity w.r.t. time
Jerk	$j = \frac{da}{dt} = \frac{d^2v}{d^2t}$	Rate of change in acceleration w.r.t. time
Duration	$t_n - t_1$	Elapsed time between action's start and end time
Straightness	$\frac{\lvert P_1 P_n \rvert}{s_n}$	Ratio of the shortest distance between the first point (P_1) and last point (P_n) in the action and the entire length of the mouse's trajectory s_n
Number of events	n	Total number of events in the action
Curvature	$\frac{d\Theta}{ds}$	Rate of angle change w.r.t. trajectory s
Sum of angles	$\sum_{i=1}^{n} \Theta_i$	Total sum of angles made by the mouse with respect to the x-axis
Maximum deviation	$\max_i d(P_i, \lvert P_1 P_n \rvert)$	Largest distance between the points of the trajectory and the segment between the two endpoints P_1 and P_n
Click time *(PC actions)*	$t_n - t_{n-1}$	Elapsed time between button press (the 2nd last event of PC actions) and button release event (the last event of PC actions)

3.1.4 Classification Phase

There are two types of machine learning (ML) algorithms: supervised and unsupervised. In supervised machine learning, the training data contains output variables (also known as target, dependent, or respondent variables) that correlate to the input variables. The supervised learning algorithm examines the data and learns a function that maps the connection between the input and output variables. Unsupervised learning algorithms are used when the training data does not have an output variable. Such algorithms try to find the intrinsic pattern and hidden structures in the data. Clustering and dimensionality reduction algorithms are examples of unsupervised learning algorithms [51]. Supervised machine learning can be further divided into regression, classification, forecasting, and anomaly detection. Bot detection in our system is framed as a classification problem.

Classification refers to a class, and the goal of classification is to categorize a set of data into classes; for example, when predicting an exam result that is either 'Pass' or 'Fail', or when working on an e-commerce project and it is required to predict the product name for some use case [52]. Our bot detection system's task

is similar, because it needs to identify whether a user is bot or human. They are two types of classification: binary classification and multi-class classification. Binary classification involves two classes (e.g., 'Pass' or 'Fail', 0 or 1). In contrast, multi-class classification involves more than two classes. For our project, we define bot detection as a binary classification problem, where 'human'=0 and 'bot'=1. We selected four algorithms for binary classification - Random Forest (RF), Decision Tree (DT), K-Nearest Neighbors (KNN) and Support Vector Machine (SVM) - because of their clear advantages as mentioned in Figure 3.5. These classification algorithms completely align with our data structure and goals of the project.

3.1.4.1 Decision Tree Classifier

Decision trees can be used for both classification and regression problems. However, they are most commonly employed to solve classification. A DT is composed of internal nodes (also called decision nodes) and leaf nodes (also called terminal nodes). Internal nodes represent data set features, branches represent decision rules, and each leaf node represents an outcome. The internal nodes have multiple branches and are used in decision making, and the leaf nodes are the outcomes of those decisions. A decision tree asks a question, and based on the answer, the tree is further split into subtrees. To predict a value for a data example, the DT algorithm starts from the root node and compares the root node's value to the example's value; based on this comparison, it follows a branch to a child node. The same process is repeated for each node until the algorithm reaches a leaf node of the tree.

3.1.4.2 Random Forest Classifier

RF is one of the most powerful and robust classification algorithms which forms a forest with a collection of decision trees. It is an ensemble learning method that is usually trained with the 'bagging' method. It relies on the idea that a combination

of learning models boosts the overall result. While the DT searches for the most important features in each node, the random forest adds randomness to the model and searches for the best features among a subset of features. RF uses many hyper-parameters, including the number of decision trees in the forest and the maximum number of features considered by each tree when splitting nodes. These hyperparameters are used to increase either the predictive power for the model or the training speed.

3.1.4.3 Support Vector Machine Classifier

SVM classifiers plot data examples as points in n-dimensional spaces, where n is the number of features. Classification is performed by finding a hyperplane (decision boundary) that best divides the classes. It is very important to find the optimal hyper-plane to best differentiate between classes. The hyperplane is a line and is identified by calculating the distance to the nearest element of each class, and the largest distance is taken. Finding a hyperplane is easy when the data is linearly separable, and a straight line can be used. However, when the data is non-linear, SVM uses the kernel trick. SVM uses three kernels: linear, radial basis function (RBF), and polynomial. As the name suggests, the linear kernel is used when the data is linear. RBF and polynomial are used when the data is non-linear. The kernel trick saves time and complexity of transforming the non-linear data to make it linearly separable.

3.1.4.4 K-Nearest Neighbors Classifier

KNN is one of the most versatile and widely used algorithms and has various applications such as finance, healthcare, handwriting detection, image, and video recognition. In KNN, K is the nearest number of neighbors which is the core deciding factor. KNN has the following three basic steps:

1. Calculate the distance between data points using a distance metric (e.g., Manhattan, Hamming, Euclidean, Minkowski)

2. Find the K closest neighbors

3. Vote on a class label among the closest neighbors

KNN performs very well when the number of features is lower. As the number of features increases, it requires more data; this may lead to overfitting as a result. However, this problem can be dealt with using a feature selection algorithm or using a different distance metric such as cosine similarity.

When choosing a classification algorithm, there are some important factors that we have to consider: 1) the size of the training data, 2) accuracy or interpretability of the output, 3) the training time it requires, 4) linearity, and 5) number of features. For data sets with a small amount of training data and a high number of features, algorithms with high bias and low variance, such Linear Regression, Linear SVM, or Naïve Bayes are recommended. Regarding accuracy, it is good to select flexible algorithms that give high accuracy at a low cost of interpretability. One such algorithm is KNN because it produces a wider range of possible shapes of the mapping functions. For example, KNN with $K = 1$ is highly flexible, as it will consider input data to generate the mapping output function [51]. An algorithm such as RF or DT can handle high dimensional and complex data structures when the data is not linear.

3.2 System Implementation

3.2.1 GUI Application

The GUI program was developed using Python's Tkinter framework [53], which is built into the Python standard library and implements widgets such as text labels,

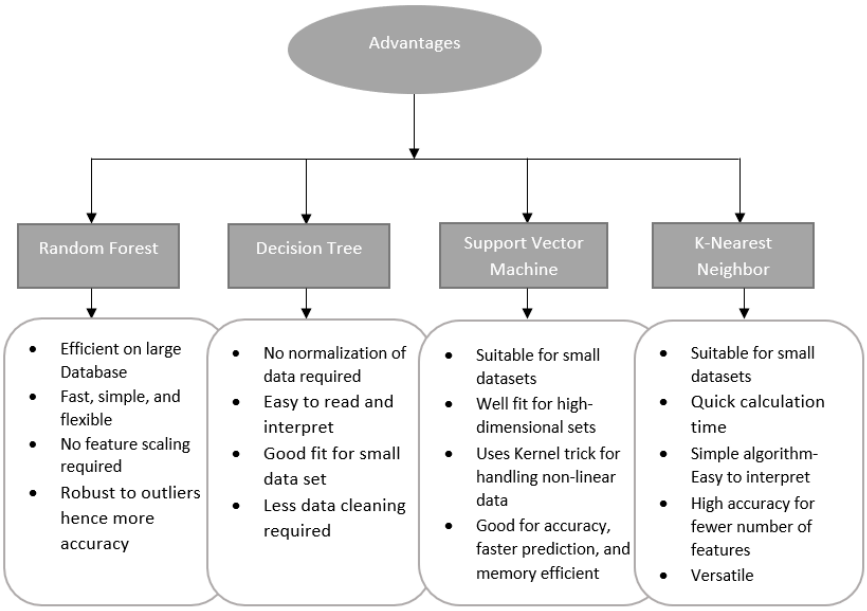


Figure 3.5: Advantages of the classifiers used

buttons, and text entry boxes, all of which are organized on top of a 'frame'. Our application contains five frames: a start page, the typing activity page, the ball activity page, the sorting activity page, and an end page. The GUI switches between frames after clicking a button (the 'Start' button for the start page, the 'Done' button for the typing activity) or completing a task (clicking the ball 10 times in the ball activity, correctly sorting all eight fruit and animal objects in the sorting activity).

The typing activity page contains the instructions, a text entry widget that the user types in, and two buttons: a 'Capture' button to start capturing keystroke activity data, and a 'Done' button to stop capturing keystroke data and switch to the ball activity.

The ball activity page contains the instructions, a 'Start' button, and the ball image object. After clicking the start button, the application starts capturing mouse activity data and the ball appears in a location that is calculated by generated random x and y screen coordinates. The ball is bound to an on-click event that hides the ball, increments a variable storing the number of times the ball has been clicked

on, calculates a new random location, and reappears the ball in the new location. After the ball click variable reaches 10, the GUI switches to the sorting activity.

The sorting activity contains the instructions as well as the four animals and four fruits in randomly chosen locations at the top half of the GUI screen. At the bottom half of the GUI screen are the two sorting boxes: one labelled 'Animals' for animals and one labelled 'Fruits' for fruits. Each object to be sorted is bound to a drag event which, upon releasing the object, checks that the object's coordinates are contained within the rectangular area formed by the correct sorting box. After sorting all eight objects correctly, the application does the following: stops capturing mouse activity, switches to the end page, and closes four seconds later.

3.2.2 Mouse/Keyboard Activity Logging

When the application starts, a unique folder is created under an events directory folder to store the user's biometrics data. Keystroke and mouse data are written to separate CSV files and are saved in this unique folder. To record activity data, we created a logger program in Python that is called from the application program. Mouse and keystroke events are recorded using mouse and keyboard listener threads imported from the pynput library [54]. These threads are instantiated when keystroke and mouse capture starts, and terminated when keystroke and mouse capture stops, respectively.

The mouse listener monitors the mouse for two events: the mouse is moved (move or drag) and the mouse button is clicked (press or release). The keyboard listener also monitors two events: a key is pressed and a key is released. Each event triggers a callback function which calculates the time that the event occurred in seconds relative to when capture started for that listener and writes a new row to the corresponding CSV file. The source code in Figure 3.6 shows the implementation of the callback functions.

```python
def on_press(key):
    global keydata, start_key_time
    keydata = str(key).strip("'")
    current_time = round(time.time() - start_key_time, 5)
    write_key_row(current_time, keydata, 'None')


def on_release(key):
    global keydata, start_key_time
    keydata = str(key).strip("'")
    current_time = round(time.time() - start_key_time, 5)
    write_key_row(current_time, 'None', keydata)


def on_move(x, y):
    current_time = time.time() - start_mouse_time
    if mouse_pressed:
        write_mouse_row(current_time, x, y, 'NoButton', 'Drag')
    else:
        write_mouse_row(current_time, x, y, 'NoButton', 'Move')


def on_click(x, y, button, pressed):
    global mouse_pressed
    global start_mouse_time
    current_time = time.time() - start_mouse_time
    if pressed:
        write_mouse_row(current_time, x, y, button, 'Pressed')
        mouse_pressed = True
    else:
        write_mouse_row(current_time, x, y, button, 'Released')
        mouse_pressed = False
```

Figure 3.6: Callback functions for the key and mouse listeners

The mouse CSV file contains five fields: the time in seconds; the x coordinate of the mouse pointer; the y coordinate of the mouse pointer; the button that was pressed or released ('NoButton' for mouse moves/drags; and the state of the button, if it was pressed or released, or the state of the mouse, if it was moved or dragged. The key CSV file contains three fields: the time in seconds; the key that was pressed ('None' for key release events); and the key that was released ('None' for key press events). Figures 3.7 and 3.8 show snippets of key and mouse CSV files outputted after a user finishes with the application.

```
Time,Key pressed,Key released
1.49153,1,None
2.0311,None,1
2.14246,2,None
2.82635,None,2
2.93911,3,None
3.18427,None,3
3.2972,Key.shift,None
3.30834,C,None
3.87142,None,Key.shift
3.88412,None,c
3.99523,Key.shift,None
4.00633,A,None
4.18396,None,Key.shift
4.195,None,a
```

Figure 3.7: Example key CSV file

```
time,mouse_x,mouse_y,button,state
0.010734081268310547,749.0,138.0,Button.left,Released
0.6990540027618408,749.0,138.0,NoButton,Move
0.7652621269226074,729.0,262.0,NoButton,Move
0.8302030563354492,664.0,194.0,NoButton,Move
0.8963179588317871,644.0,174.0,NoButton,Move
0.9622740745544434,650.0,253.0,NoButton,Move
1.027834177017212,583.0,295.0,NoButton,Move
1.0936000347137451,484.0,322.0,NoButton,Move
1.1581969261169434,481.0,363.0,NoButton,Move
1.224581003189087,481.0,363.0,NoButton,Move
1.235687255859375,481.0,363.0,Button.left,Pressed
1.4005579948425293,481.0,363.0,NoButton,Drag
1.4108710289001465,481.0,363.0,Button.left,Released
2.0765480995178223,481.0,362.0,NoButton,Move
```

Figure 3.8: Example mouse CSV file

3.2.3 Automated Bot Scripts

The design of the bot scripts was informed by prior research on how bots solve CAPTCHAs. Bots use object recognition to detect and locate objects on screen and build dictionaries of the correct answers for solving cognitive games. Hence, our bot scripts can find: the text entry box to type in, the buttons (i.e., 'Start', 'Capture', and 'Done'), the ball, the fruit and animal objects, and the sorting boxes. The bots are also able to sort the fruits and animals into the correct boxes. To control the mouse the keyboard, we use PyAutoGUI [55] to automate actions such as moving or dragging the mouse, clicking, and typing keys. Object recognition is implemented by PyAutoGUI's `locateOnScreen()` function and the OpenCV library [56]; `locateOnScreen()` returns takes an image file of the object as input and returns the four integer tuple (left, top, width, height), which can be passed to a `center()` method to find the (x, y) coordinates of the center of the object on screen. OpenCV provides a *confidence* keyword which specifies the accuracy with which the function should locate the image on screen.

The simple bot and the advanced bot mostly use the same PyAutoGUI func-

tions to carry out actions. Where they differ is the advanced bot's incorporation of random variables and function parameters. For example, the advanced bot program uses random variables to alter the time delays between key press and key release events and after completing each of 10 words. There is also a random delay introduced between mouse button press and mouse button release events when clicking on objects. For dragging objects in the sorting activity, the advanced bot takes advantage of the `duration` and `tween` parameters of PyAutoGUI's `dragTo` function which control how long the drag occurs and the mouse's motion, respectively. The duration is determined by a random variable. The tween for each drag is chosen randomly. The set of tweens used are: "easeInOutBack" - the mouse overshoots the start and destination of the drag; "easeInOutBounce" - the mouse 'bounces' at the start and end of the drag; "easeInOutElastic" - the mouse 'wobbles' towards the midpoint of the drag.

The Bézier curves that define the trajectories for the advanced bot's mouse movements use the SciPy and NumPy libraries [57, 58]. See Figure 3.9 for the Bézier curve source code.

3.2.4 Event Processing

After data collection is complete, user biometric data is stored under an `events` directory across multiple folders - one folder per user - where each folder contains a `key.csv` file and a `mouse.csv` file storing the user's keystroke and mouse events, respectively. The purpose of the feature extraction and features calculation phases of our system is to parse the keystroke and mouse events for each user, generate corresponding keystroke and mouse feature row files, and store them in user folders under a `features` directory.

Feature extraction and calculation begins in a program that loops over all folders in the `events` directory. For each folder, the program: creates a new folder

```python
def moveTo(x2, y2):
    # Moves the mouse from current position to the destination (x2, y2)
    # along a path defined by Bezier curve
    disable_pauses()
    # Start (current) position
    x1, y1 = pyautogui.position()
    # Select a random number of control points
    cp = random.randint(3, 7)

    # Distribute control points between start and destination evenly.
    x = np.linspace(x1, x2, num=cp, dtype='int')
    y = np.linspace(y1, y2, num=cp, dtype='int')

    # Randomize inner points a bit (+-RAND at most).
    RAND = 75
    xr = [random.randint(-RAND, RAND) for k in range(cp)]
    yr = [random.randint(-RAND, RAND) for k in range(cp)]
    xr[0] = yr[0] = xr[-1] = yr[-1] = 0
    x += xr
    y += yr

    # Approximate using Bezier spline.
    degree = 3 if cp > 3 else cp - 1  # Degree of B-spline
    # Finds the B-spline representation of the curve represented by [x,y], degree k
    tck, u = interpolate.splprep([x, y], k=degree)
    # Move up to a certain number of points
    u = np.linspace(0, 1, num=2 + int(point_dist(x1, y1, x2, y2) / 50.0))
    points = interpolate.splev(u, tck)

    # Move mouse. Choose random duration for the mouse movement
    duration = random.uniform(0.2, 0.6)
    timeout = duration / len(points[0])
    point_list = zip(*(i.astype(int) for i in points))

    for i, point in enumerate(point_list):
        pyautogui.moveTo(*point)
        time.sleep(timeout)
```

Figure 3.9: Implementation of Bézier curves

with the same folder name but under the `features` directory; calls a program that extracts and calculates the keystroke features from `key.csv`; and calls a program that extracts and calculates the mouse features from `mouse.csv`.

3.2.4.1 Keystroke features

The keystroke feature extraction and calculation program contains a function that parses the `key.csv` file, processes the keystroke event rows in a for-loop, and calculates the keystroke features. The function defines four lists for feature calculation: `hold_times` stores the times that each key is held down for; `cpr_times` stores consecutive press release times; `released_times` stores key release event times; and `pressed_times` stores key press event times. In addition, a variable `last_pressed_time` stores the time of the most recently processed key press event.

The for-loop first checks that the row is not a typo; typos are defined as key press or key release events for keys outside of the Shift key and the characters in '123CAPabc!' (both shifted and unshifted, e.g. 'C' and 'c', '2' and '@'). It then determines whether the row corresponds to a key press or key release event based on the values of the 'Key Pressed' and 'Key Released' fields and gets the event time. Key press event times are added to `pressed_times` and key release event times are added to `released_times`. For key release events, consecutive press times are calculated by subtracting `last_pressed_time` from the event time and added to `cpr_times`. For hold times, a Python dictionary stores entries that map keys to their most recent press time; entries are added or updated when processing key press events. When a key is released, its press time is retrieved from the dictionary and subtracted from the key release time to calculate the hold time. The total taken by the user to type all 10 words is calculated by subtracting the first event row time from the last.

After all rows are processed, the feature extraction phase for keystroke data is complete. The function then calculates the press and release latencies from `pressed_times` and `released_times` and the average, standard deviation, maximum, and minimum values of the press latencies, release latencies, `cpr_times`, and `hold_times`. The keystroke feature row is formed from all the calculated values. See Figure 3.10 for a screenshot of the feature calculation code after processing key event rows.

After calculating the feature row, the key feature extraction/calculation program creates `key.csv` in the user's folder under the `features` directory and writes the keystroke feature header as well as the keystroke feature row.

3.2.4.2 Mouse features

The mouse feature extraction and calculation program first processes the `mouse.csv` file and forms MM, PC, and DD actions from groups of events. Actions are rep-

```
    # Total time taken to type 10 words
    total_time_taken = float(rows[-1]['Time']) - float(rows[0]['Time'])

# Times between key releases
release_latency = [released_times[i + 1] - released_times[i] for i in range(len(released_times) - 1)]
# Times between key presses
press_latency = [pressed_times[i + 1] - pressed_times[i] for i in range(len(pressed_times) - 1)]

# Average/Maximum/Minimum/Std.Dev. hold times
avg_hold_time, max_hold_time, min_hold_time, sd_hold_time = statistics.mean(hold_times), max(hold_times), min(hold_times), statistics.stdev(hold_times)
# Average/Maximum/Minimum/Std.Dev. consecutive press release times
avg_cpr_time, max_cpr_time, min_cpr_time, sd_cpr_time = statistics.mean(cpr_times), max(cpr_times), min(cpr_times), statistics.stdev(cpr_times)
# Average/Maximum/Minimum/Std.Dev. release latency times
avg_released_time, max_released_time, min_released_time, sd_released_time = statistics.mean(release_latency), max(release_latency), min(release_latency), statistics.stdev(release_latency)
# Average/Maximum/Minimum/Std.Dev. press latency times
avg_press_time, max_press_time, min_press_time, sd_press_time = statistics.mean(press_latency), max(press_latency), min(press_latency), statistics.stdev(press_latency)

# Values of the features that are written to new CSV
feature_row = [total_time_taken, avg_hold_time, max_hold_time, min_hold_time, sd_hold_time,
               avg_cpr_time, max_cpr_time, min_cpr_time, sd_cpr_time,
               avg_released_time, max_released_time, min_released_time, sd_released_time,
               avg_press_time, max_press_time, min_press_time, sd_press_time]
```

Figure 3.10: Keystroke features calculation

resented by a `MouseAction` class and each `MouseAction` object is instantiated with the action type and the list of mouse events. Figure 3.11 shows the source code that parses events into actions. After all events have been processed, the resulting list of actions is filtered to remove actions with fewer than two events.

`MouseAction` objects have a method `calculate_features()`. Each action calls this method to calculate its action features from its events and save these features as instance variables. The features are calculated according to their definitions in Table 3.2.

A `Session` class represents all of a user's mouse actions from a single application run. After features have been calculated for all of a user's actions, a `Session` object is formed from the set of actions. From these actions, the object calculates two session features: the total number of actions and the total duration of the mouse activity, which is computed by subtracting the first event time in the first action from the last event time in the last action.

The `Session` object calls its method `calculate_features()` which computes the features over its set of actions which, combined with the two session features, form the user's mouse feature row. The set of actions are partitioned by action type. For each action type, the average, standard deviation, maximum, and minimum values of each action feature is calculated. For PC actions, the average, standard, maxmimum,

```
prev_event = None
for i, row in enumerate(processed_rows):
    time = row['time']
    x = row['mouse_x']
    y = row['mouse_y']
    button = row['button']
    state = row['state']

    mouse_event = MouseEvent(time, x, y, button, state)

    if mouse_event.state == MouseState.RELEASED:
        # Point and Click Action
        if prev_event is not None and prev_event.state == MouseState.PRESSED:
            mouse_events.append(mouse_event)
            actions.append(MouseAction(ActionType.PC, mouse_events))
            mouse_events = []
        # Drag and Drop Action
        if prev_event is not None and prev_event.state == MouseState.DRAGGED:
            mouse_events.append(mouse_event)
            actions.append(MouseAction(ActionType.DD, mouse_events))
            mouse_events = []
    elif mouse_event.state == MouseState.PRESSED:
        next_event = processed_rows[i + 1]
        # Check if next event is a mouse drag, in which case create Mouse Move action and start new Drag and Drop action
        if MouseState(next_event['state']) == MouseState.DRAGGED:
            if len(mouse_events) > 0:
                actions.append(MouseAction(ActionType.MM, mouse_events))
            mouse_events = [mouse_event]
        else:
            mouse_events.append(mouse_event)
    # Mouse movement (either mouse held down or not)
    else:
        mouse_events.append(mouse_event)

    prev_event = mouse_event
```

Figure 3.11: Parsing mouse events into actions

and minimum click times are calculated as well. The two session features and the calculated action features for each action type are combined together to form the user's mouse feature row.

After calculating the mouse feature row, the mouse feature extraction/calculation program creates `mouse.csv` in the user's folder under the `features` directory and writes the mouse feature header as well as the mouse feature row. (This row is combined with the key feature row before classification; please see Section 3.3.1.)

3.2.5 Classifiers

The RF, DT, SVM, and KNN classifiers were implemented with scikit-learn version 0.24.2 [57], which provides dozens of machine learning algorithms and models for Python. The pandas library [59] was used for creating and manipulating the data sets. The RF classifier was implemented with a forest of 150 decision trees. The SVM classifier was implemented with a linear kernel. The KNN classifier used $K = 3$.

3.3 Testing

In this section, we describe our testing procedure and display and discuss the analysis of our results.

3.3.1 Data Sets

To build and evaluate our bot detection system, we collected data from human participants and the two bots. Our testing involved 20 participants who were instructed to complete the three activities in our GUI and send the keystroke and mouse event CSV files generated from their session. Some participants opted to complete multiple sessions. To gather bot data, we ran the bot scripts in parallel with the GUI. In total, we collected data from 68 human sessions, 30 simple bot sessions, and 30 advanced bot sessions.

After executing feature extraction and calculation, the keystroke and mouse feature rows are stored in `key.csv` and `mouse.csv`, respectively, in unique session folders under the `features` directory as detailed in Sections 3.2.4.1 and 3.2.4.2. To form data sets, each user's keystroke and mouse feature rows are merged together. The merged feature rows are then concatenated together. Because our classification is supervised, a class label for each feature row is required. Therefore, we added a new 'class' field to the feature rows where 'class' is 1 if the user is a bot and 0

otherwise. Finally, the feature rows are concatenated together to form a data set, where each row represents a user and each column represents a feature (except for the last column, which is the class label). Using python's NumPy library [58], random statistical noise drawn from a Gaussian distribution was added to a randomly selected 10% of the data set rows.

To compare the results of our system against the simple bot versus the advanced bot, we formed two data sets from the collected data. The first data set, SIMPLE, contains the data from the 68 human sessions and 30 simple bot examples. The second data set, ADVANCED, contains the data from the 68 human sessions and 30 advanced bot examples. In total, each data set has 98 examples, where each example has 131 features and a class label.

3.3.2 Evaluation Metrics

We used 5-fold cross validation to train and test our classifiers against the SIMPLE and ADVANCED data sets. We evaluate the performance of each classifier in terms of the following metrics:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \times 100\% \tag{3.3}$$

$$Precision = \frac{TP}{TP + FP} \times 100\% \tag{3.4}$$

$$Recall = \frac{TP}{TP + FN} \times 100\% \tag{3.5}$$

$$F_1\ score = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \times 100\% \tag{3.6}$$

$$AUC = \text{Area under the ROC curve} \tag{3.7}$$

where TP = true positives, FP = false positives, TN = true negatives, and FN = false negatives. For each performance metric, we calculated the average $\pm$ standard deviation across all five folds. In addition, we plot the ROC curve for each

fold and report the mean AUC across all folds.

3.3.3 Results and Analysis

Tables 3.3 and 3.4 show the classification results in terms of accuracy, precision, recall, and F_1 score against the SIMPLE and ADVANCED data sets, respectively. The best results for each metric in each data set are in bold.

Table 3.3: Performance metrics across four models for SIMPLE data set

Model	**Accuracy**	**Precision**	**Recall**	**F_1 Score**
Random Forest	96.89% ± 4.68%	**97.14%±6.39%**	93.33% ± 14.91%	94.46% ± 8.74%
Decision Tree	95.84% ± 6.85%	94.29% ± 12.78%	93.33% ± 9.13%	93.57% ± 10.10%
SVM	96.89% ± 2.84%	94.29% ± 7.82%	**96.67%±7.45%**	**95.10 ± 4.50%**
K-Nearest Neighbors	**96.95%±4.50%**	96.67% ± 7.45%	93.33% ± 9.13%	94.85%7.55%

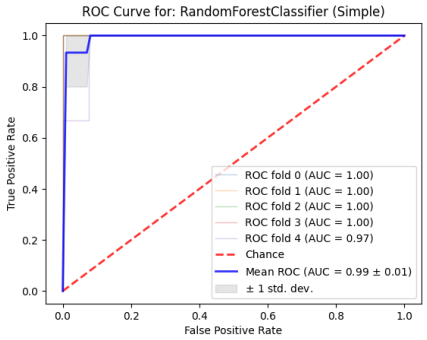
Table 3.4: Performance metrics across four models for ADVANCED data set

Model	**Accuracy**	**Precision**	**Recall**	**F_1 Score**
Random Forest	**96.95%±4.50%**	**96.67%±7.45%**	**93.33%±9.13%**	**94.85%±7.55%**
Decision Tree	92.00% ± 15.25%	88.57% ± 25.56%	86.67% ± 21.73%	87.41% ± 23.40%
SVM	94.89% ± 3.54%	90.95% ± 8.32%	**93.33%±9.13%**	91.77 ± 5.92%
K-Nearest Neighbors	94.84% ± 3.72%	94.29% ± 7.82%	90.00% ± 14.91%	91.10% ± 7.17%

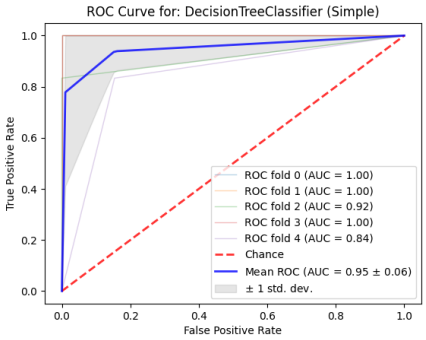
Figures 3.12 and 3.13 show the ROC curves and mean AUC for the classifiers against the SIMPLE and ADVANCED data sets, respectively.

As observed in Table 3.3, KNN achieved the highest accuracy against the SIMPLE data set, while RF achieved the highest accuracy against the ADVANCED data set. The RF classifier showed the best overall performance among all four classifiers, giving the highest precision (97.14%) and mean AUC (0.99) against the SIMPLE data set as well as the highest scores across all metrics against the ADVANCED data set. The SVM and KNN classifiers also performed well; each classifier achieved the highest results for at least one metric against either the SIMPLE or ADVANCED data set.

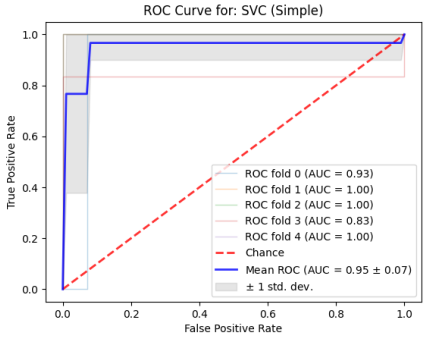
The DT classifier performed the worst amongst all four classifiers. It yielded the lowest (or tied for lowest) scores for every metric against both data sets. In



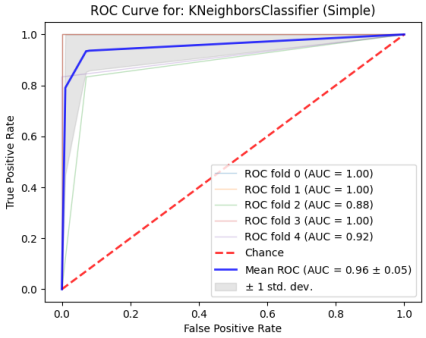
(a) Random Forest



(b) Decision Tree



(c) SVM



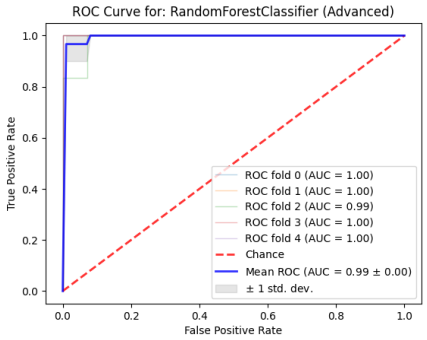
(d) K-Nearest Neighbors

Figure 3.12: ROC Curves for SIMPLE data set

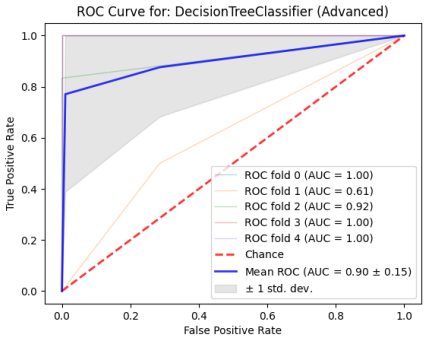
addition, the standard deviation of the results for DT are quite high, particularly against the ADVANCED data set. A possible explanation for this observation is that decision trees are prone to overfitting, which occurs when a model tightly fits the training data so that it performs well on the training examples but poorly against test examples and can cause high variance.

With few exceptions, the results of our bot detection system against the SIMPLE data set were higher than the results against the ADVANCED data set. This indicates that the techniques used in the advanced bot made it slightly more effective at evading detection.

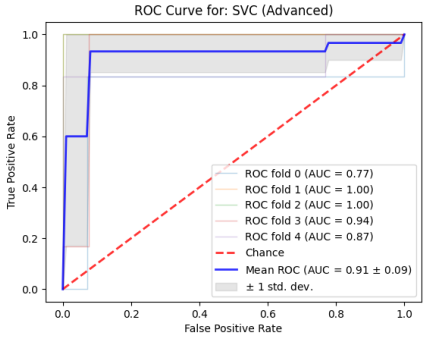
None of the classifiers achieved a 100% precision score against either of the data sets. This implies that some human users are classified as bots, which is a



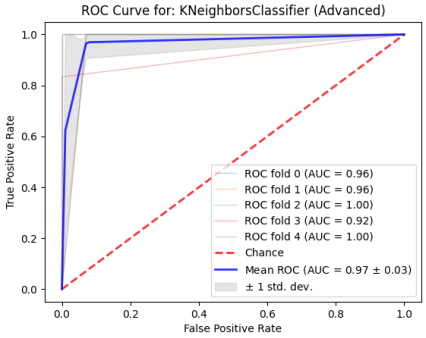
(a) Random Forest



(b) Decision Tree



(c) SVM



(d) K-Nearest Neighbors

Figure 3.13: ROC Curves for ADVANCED data set

concern for login systems.

Lastly, while the classification results for our bot detection system are high ($> 85\%$ across all metrics), none of the models achieved perfect scores. Our system should not be viewed as the sole security solution against credential stuffing bots, but as part of layered defense on top of additional services (e.g., browser fingerprinting, firewalls).

Chapter 4

Conclusion and Future Work

Credential stuffing attacks are on the rise. The massive number of IMAP-based credential stuffing attacks on Microsoft 365 and G Suite accounts have been in the headlines. As botnets carry out the majority of credential stuffing attacks, it has become necessary to implement a bot detection system as a layer of security. The drawbacks of many bot detection techniques such as deploying honeynets, blacklisting IPs, signature-based bot detection, and n-gram models have led many researchers to use behavioural biometrics. In this paper, we aimed to show that two types of behavioural biometrics - mouse dynamics and keystroke dynamics - can effectively distinguish between humans and bots. We introduced a supervised learning bot detection system using mouse and keystroke dynamics and compared the RF, DT, SVM and KNN algorithms for classification. Our entire project is built in Python and consists of four phases: the data collection phase, feature extraction phase, feature calculation phase, and classification phase. For testing, we collected data from 68 user sessions, 30 simple bot sessions, and 30 advanced bot sessions, and formed two data sets: Simple and Advanced. In total, each data set had 98 examples and each example had 131 features.

Our system showed the best overall performance with the RF classifier, giving

96.89% accuracy for the SIMPLE data set and 95.95% for the ADVANCED data set. RF scored the highest precision and mean AUC against the SIMPLE data set and the highest scores for every metric against the ADVANCED data set. The worst performing classifier was DT, which yielded the lowest scores for every metric against both data sets.

For future work, we plan to add logistic regression as the meta layer to develop an ensemble learning model that would be more reliable. The work can be also improved by collecting and training the model with more samples for future work. In addition, the model can also be tested with free-text keystrokes to analyze the case in other scenarios. Deep learning or artificial intelligence can also be used to create more realistic bots.

Bibliography

[1] "What is a password manager? here's why you should be using one," Jul 2021. [Online]. Available: https://www.androidauthority.com/password-manager-1238302/

[2] S. Rees-Pullman, "Is credential stuffing the new phishing?" *Computer Fraud & Security*, vol. 2020, pp. 16–19, 07 2020.

[3] M. Golla, L. Filipe, M. Wei, M. Dürmuth, B. Ur, J. Hainline, and E. Redmiles, ""what was that site doing with my facebook password?" designing password-reuse notifications," *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 1549–1566, 2018.

[4] "Telesign consumer account security report." [Online]. Available: https://www.telesign.com/resource/telesign-consumer-account-security-report

[5] 2021. [Online]. Available: https://www.akamai.com/us/en/resources/our-thinking/state-of-the-internet-report/

[6] K. C. Wang and M. K. Reiter, "Detecting stuffing of a user's credentials at her own accounts," *Proceedings of the 29th USENIX Security Symposium*, pp. 2201–2218, 2020.

[7] S. Gatlan, "Multi-factor auth bypassed in office 365 and g suite imap attacks," Mar 2019. [Online]. Available: https://www.bleepingcomputer.com/news/security/multi-factor-auth-bypassed-in-office-365-and-g-suite-imap-attacks/

[8] "Brute force attack analysis of new cloud attacks: Proofpoint us," 2019. [Online]. Available: https://www.proofpoint.com/us/threat-insight/post/threat-actors-leverage-credential-dumps-phishing-and-legacy-email-protocols

[9] P. Kirkbride, M. A. Akber Dewan, and F. Lin, "Game-Like Captchas for Intrusion Detection," *Proceedings - IEEE 18th International Conference on Dependable, Autonomic and Secure Computing, IEEE 18th International Conference on Pervasive Intelligence and Computing, IEEE 6th International Conference on Cloud and Big Data Computing and IEEE 5th Cybe*, no. August, pp. 312–315, 2020.

[10] S. Mansfield-Devine, "Locking the door: tackling credential abuse," *Network Security*, vol. 2021, pp. 11–19, 03 2021.

[11] X. Li, B. A. Azad, A. Rahmati, and N. Nikiforakis, "Good bot , bad bot : Characterizing automated browsing activity," *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[12] B. Lu, X. Zhang, Z. Ling, Y. Zhang, and Z. Lin, "A measurement study of authentication rate-limiting mechanisms of modern websites," *ACM International Conference Proceeding Series*, pp. 89–100, 2018.

[13] M. Hartwig, "Mfa is not enough - malicious oauth apps in office 365 are here to stay: Vectra ai," Jun 2020. [Online]. Available: https://www.vectra.ai/blogpost/mfa-is-not-enough

[14] M. Antal and E. Egyed-Zsigmond, "Intrusion detection using mouse dynamics," *IET Biometrics*, vol. 8, no. 5, pp. 285–294, 2019.

[15] M. Antal, Norbert Fejer, and K. Buza, "SapiMouse : Mouse Dynamics-based User Authentication Using Deep Feature Learning," *IEEE 15th International Symposium on Applied Computational Intelligence and Informatics*, pp. 61–66, 2021.

[16] H. Gamboa and A. Fred, "A behavioral biometric system based on human-computer interaction," *Proc SPIE*, vol. 5404, pp. 381–392, 08 2004.

[17] A. A. E. Ahmed and I. Traore, "A new biometric technology based on mouse dynamics," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 3, pp. 165–179, 2007.

[18] C. Shen, Z. Cai, and X. Guan, "Continuous authentication for mouse dynamics: A pattern-growth approach," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, 2012, pp. 1–12.

[19] C. Shen, Z. Cai, X. Guan, Y. Du, and R. A. Maxion, "User authentication through mouse dynamics," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 16–30, 2013.

[20] B. Sayed, I. Traore, I. Woungang, and M. S. Obaidat, "Biometric authentication using mouse gesture dynamics," *IEEE Systems Journal*, vol. 7, no. 2, pp. 262–274, 2013.

[21] M. Mohamed and N. Saxena, "Gametrics: Towards attack-resilient behavioral authentication with simple cognitive games," *ACM International Conference Proceeding Series*, vol. 5-9-Decemb, pp. 277–288, 2016.

[22] L. H. Kim, R. Goel, J. Liang, M. Pilanci, and P. E. Paredes, "Linear predictive coding as a valid approximation of a mass spring damper model for acute stress prediction from computer mouse movement," 2020.

[23] L. Pepa, A. Sabatelli, L. Ciabattoni, A. Monteriù, F. Lamberti, and L. Morra, "Stress detection in computer users from keyboard and mouse dynamics," *IEEE Transactions on Consumer Electronics*, vol. 67, no. 1, pp. 12–19, 2021.

[24] I. Arapakis and L. Leiva, "Learning efficient representations of mouse movements to predict user attention," in *SIGIR 2020 - Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 07 2020, pp. 1309–1318.

[25] A. Acien, A. Morales, J. Fierrez, R. Vera-Rodriguez, and O. Delgado-Mohatar, "BeCAPTCHA: Behavioral bot detection using touchscreen and mobile sensors benchmarked on HuMIdb," *Engineering Applications of Artificial Intelligence*, vol. 98, 2021.

[26] M. Antal and L. Denes-Fazakas, "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)*, 2019, pp. 143–148.

[27] P. Chong, Y. Elovici, and A. Binder, "User Authentication Based on Mouse Dynamics Using Deep Neural Networks: A Comprehensive Study," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1086–1101, 2020.

[28] A. Harilal, F. Toffalini, J. Castellanos, J. Guarnizo, I. Homoliak, and M. Ochoa, "Twos: A dataset of malicious insider threat behavior based on a gamified competition," in *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*, 2017, pp. 45–56.

[29] I. Tsimperidis, P. D. Yoo, K. Taha, A. Mylonas, and V. Katos, "R 2 bn: An adaptive model for keystroke-dynamics-based educational level classification," *IEEE transactions on cybernetics*, vol. 50, no. 2, pp. 525–535, 2018.

[30] A. Morales, A. Acien, J. Fierrez, J. V. Monaco, R. Tolosana, R. Vera, and J. Ortega-Garcia, "Keystroke biometrics in response to fake news propagation in a global pandemic," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2020, pp. 1604–1609.

[31] F. Bergadano, D. Gunetti, and C. Picardi, "User authentication through keystroke dynamics," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 4, pp. 367–397, 2002.

[32] S. Krishnamoorthy, L. Rueda, S. Saad, and H. Elmiligi, "Identification of user behavioral biometrics for authentication using keystroke dynamics and machine learning," in *Proceedings of the 2018 2nd International Conference on Biometric Engineering and Applications*, 2018, pp. 50–57.

[33] V. Mishra, R. Gupta, G. Sood, and J. Patni, "User authentication using keystroke dynamics," *Recent Trends Sci Technol Manag Soc Dev*, vol. 73, 2018.

[34] X. Lu, S. Zhang, P. Hui, and P. Lio, "Continuous authentication by free-text keystroke based on cnn and rnn," *Computers & Security*, vol. 96, p. 101861, 2020.

[35] K. Ahuja and V. Todwal, "Software bot detection by keystroke dynamics," *Journal of Critical Reviews*, vol. 7, no. 19, pp. 9975–9982, 2020.

[36] L. A. Alreshoodi and S. A. Alsuhibany, "A proposed methodology for detecting human attacks on text-based captchas."

[37] Z. Chu, S. Gianvecchio, A. Koehl, H. Wang, and S. Jajodia, "Blog or block: Detecting blog bots through behavioral biometrics," *Computer Networks*, vol. 57, no. 3, pp. 634–646, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2012.10.005

[38] J. Solano, L. Tengana, A. Castelblanco, E. Rivera, C. Lopez, and M. Ochoa, "A few-shot practical behavioral biometrics model for login authentication in web applications," in *NDSS Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb'20)*, 2020.

[39] X. Xu, L. Liu, and B. Li, "A survey of captcha technologies to distinguish between human and computer," *Neurocomputing*, vol. 408, 05 2020.

[40] M. Guerar, L. Verderame, M. Migliardi, F. Palmieri, and A. Merlo, "Gotta CAPTCHA 'Em All: A Survey of Twenty years of the Human-or-Computer Dilemma," *ACM Computing Surveys*, vol. 0, no. 0, 2021. [Online]. Available: http://arxiv.org/abs/2103.01748

[41] M. Mohamed, N. Sachdeva, M. Georgescu, S. Gao, N. Saxena, C. Zhang, P. Kumaraguru, P. C. Van Oorschot, and W. B. Chen, "A three-way investigation of a game-CAPTCHA: Automated attacks, relay attacks and usability," *ASIA CCS 2014 - Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, no. June, pp. 195–206, 2014.

[42] J. Solano, L. Camacho, A. Correa, C. Deiro, J. Vargas, and M. Ochoa, "Combining behavioral biometrics and session context analytics to enhance risk-based static authentication in web applications," *International Journal of Information Security*, vol. 20, no. 2, pp. 181–197, 2021. [Online]. Available: https://doi.org/10.1007/s10207-020-00510-x

[43] B. Amin Azad, O. Starov, P. Laperdrix, and N. Nikiforakis, *Web Runner 2049: Evaluating Third-Party Anti-bot Services*. Springer International Publishing, 2020, vol. 12223 LNCS.

[44] M. Nathan, "Credential stuffing: new tools and stolen data drive continued attacks," *Computer Fraud & Security*, vol. 2020, no. 12, pp. 18–19,

2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1361372320301305

[45] Y. Xing, H. Shu, H. Zhao, D. Li, and L. Guo, "Survey on botnet detection techniques: Classification, methods, and evaluation," *Mathematical Problems in Engineering*, vol. 2021, 2021.

[46] A. Lagopoulos, G. Tsoumakas, and G. Papadopoulos, "Web Robot Detection in Academic Publishing," 2017. [Online]. Available: http://arxiv.org/abs/1711.05098

[47] P. Shi, Z. Zhang, and K. K. R. Choo, "Detecting Malicious Social Bots Based on Clickstream Sequences," *IEEE Access*, vol. 7, pp. 28 855–28 862, 2019.

[48] M. Tsikerdekis, S. Barret, R. Hansen, M. Klein, J. Orritt, and J. Whitmore, "Efficient deep learning bot detection in games using time windows and long short-term memory (LSTM)," *IEEE Access*, vol. 8, pp. 195 763–195 771, 2020.

[49] R. Bååth, "Bot or not: Can you spot the automated mouse movements?" Dec 2020. [Online]. Available: https://blog.castle.io/bot-or-not-can-you-spot-the-automated-mouse-movements/

[50] Z. Sun, R. He, J. Feng, S. Shan, and Z. Guo, *Biometric Recognition 14th Chinese Conference, CCBR 2019, Zhuzhou, China, October 12–13, 2019, Proceedings: 14th Chinese Conference, CCBR 2019, Zhuzhou, China, October 12–13, 2019, Proceedings*, 01 2019.

[51] "An easy guide to choose the right machine learning algorithm," https://www.kdnuggets.com/2020/05/guide-choose-right-machine-learning-algorithm.html, august 2021.

[52] J. Rana, "Comparison of classification algorithms (lr, dt, rf, svm, knn)," https://www.kdnuggets.com/2020/05/guide-choose-right-machine-learning-algorithm.html, february 2020.

[53] F. Lundh, "An introduction to tkinter," *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.

[54] "pynput package documentation." [Online]. Available: https://pynput.readthedocs.io/en/latest/

[55] Asweigart, "asweigart/pyautogui: A cross-platform gui automation python module for human beings. used to programmatically control the mouse & keyboard." [Online]. Available: https://github.com/asweigart/pyautogui

[56] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[58] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, p. 357–362, 2020.

[59] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.