



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SECURING NETWORKS AGAINST ANONYMOUS INVADERS

by

Nathan J. Richardson and Lucas J. Burke

June 2019

Thesis Advisor:

Peter J. Denning

Co-Advisor:

Vinnie Monaco

Approved for public release. Distribution is unlimited.

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			<i>Form Approved OMB No. 0704-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE June 2019	3. REPORT TYPE AND DATES COVERED Master's thesis		
4. TITLE AND SUBTITLE SECURING NETWORKS AGAINST ANONYMOUS INVADERS			5. FUNDING NUMBERS	
6. AUTHOR(S) Nathan J. Richardson and Lucas J. Burke				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING / MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) The power of anonymity on the internet is being leveraged by criminals, online trolls, and malicious hackers. Cyber criminals masquerade as legitimate users in order to steal valuable private data, commit fraud, and steal money from their victims. This thesis investigates security vulnerabilities that exist as a product of user anonymity and provides a biometric mitigation to one of the most common exploits against DoD personnel—social engineering. To strip anonymity from malicious users, we used a concept that harkens back to World War II called fist of the sender. This technique was used by allied intelligence analysts to identify enemy units by the rhythm of their radio operators when they transmitted Morse code. We applied the same concept to keystrokes along a keyboard and created a digital fist. Our research used a chatbot that posed as a victim, and we had student-subjects play the role of Social Engineers and attempt to phish the chatbot. Our system measures keystroke dynamics to increase user recognition and introduces specific perturbations during a user's typing session—sometimes by making the caret disappear, and sometimes by suddenly moving the caret back a few spaces. We hypothesized that as a user tried to recover from these interactions, we could increase typing attribution since every user's subconscious reaction would be unique. Our results show cases where identification increased over 20% in some cases through the introduction of perturbations.				
14. SUBJECT TERMS biometrics, DARPA, AI, anonymity, keystroke dynamics, social engineering, cyber, botnet			15. NUMBER OF PAGES 93	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

SECURING NETWORKS AGAINST ANONYMOUS INVADERS

Nathan J. Richardson
Lieutenant Commander, United States Navy
BS, University of Southern California, 2009

Lucas J. Burke
Major, United States Marine Corps
BS, U.S. Naval Academy, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
June 2019**

Approved by: Peter J. Denning
Advisor

Vinnie Monaco
Co-Advisor

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The power of anonymity on the internet is being leveraged by criminals, online trolls, and malicious hackers. Cyber criminals masquerade as legitimate users in order to steal valuable private data, commit fraud, and steal money from their victims. This thesis investigates security vulnerabilities that exist as a product of user anonymity and provides a biometric mitigation to one of the most common exploits against DoD personnel—social engineering. To strip anonymity from malicious users, we used a concept that harkens back to World War II called fist of the sender. This technique was used by allied intelligence analysts to identify enemy units by the rhythm of their radio operators when they transmitted Morse code. We applied the same concept to keystrokes along a keyboard and created a digital fist. Our research used a chatbot that posed as a victim, and we had student-subjects play the role of Social Engineers and attempt to phish the chatbot. Our system measures keystroke dynamics to increase user recognition and introduces specific perturbations during a user's typing session—sometimes by making the caret disappear, and sometimes by suddenly moving the caret back a few spaces. We hypothesized that as a user tried to recover from these interactions, we could increase typing attribution since every user's subconscious reaction would be unique. Our results show cases where identification increased over 20% in some cases through the introduction of perturbations.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

1	Introduction	1
2	Background	5
2.1	Anonymity	5
2.2	Biometrics and Feature Collection	11
2.3	Social Engineering	17
3	Design and Methodology	21
3.1	Attacker Engagement	23
3.2	Biometric Collection	25
3.3	Reflexive Keystroke Dynamics	26
3.4	Profile Creation	27
3.5	Profile Matching	29
4	System Implementation and Testing	31
4.1	Operating Environment	31
4.2	Chatbot Design	32
4.3	Web Interface	33
4.4	Keystroke and Mouse Movement Collection	33
4.5	Reflex Collection	34
4.6	Scenarios	35
4.7	Measurements and Calculations	37
5	Results and Discussion	39
5.1	Results	39
5.2	Scenario Test Results	40
6	Conclusions and Future Work	49
6.1	Conclusions	49

6.2 Future Work	51
Appendix A Source Code	55
Appendix B Scenario Instructions	67
Appendix C Sample Chat Session	69
Bibliography	71
Initial Distribution List	77

List of Figures

Figure 2.1	An example social engineering framework.	17
Figure 2.2	An updated social engineering model.	19
Figure 3.1	Biometric capture, creation and matching.	22
Figure 3.2	Definitions of identification rates for the system.	23
Figure 3.3	Profile creation and initial query to database.	28
Figure 3.4	Matching known profile by querying to each user in database. . .	29
Figure 4.1	Game dialog layout in Twine.	33
Figure 4.2	Events bubble up from most specific element to window.	34
Figure 4.3	changeFocus() function call from webpage.	35
Figure 4.4	moveCaret() function call from webpage.	36
Figure 5.1	Normal versus reflex login accuracy - press features	41
Figure 5.2	Normal versus reflex chat accuracy - press features	41
Figure 5.3	Normal versus reflex login accuracy - top 5% of digraphs used . .	43
Figure 5.4	Normal versus reflex chat accuracy - top 5% of digraphs used . .	43
Figure 5.5	Normal versus reflex login accuracy - top 50% of digraphs used .	44
Figure 5.6	Normal versus reflex chat accuracy - top 50% of digraphs used .	45
Figure 5.7	Differences in users' normal versus reflex login patterns	46

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

DISA	Defense Information Systems Agency
DoD	Department of Defense
FBI	Federal Bureau of Investigation
NPS	Naval Postgraduate School
USMC	U.S. Marine Corps
USN	U.S. Navy
USG	United States government

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

First, we would like to thank our thesis advisors, Professors Denning and Monaco, for their countless hours spent helping us explore topics as broad as anonymity on the internet, to the specifics of Manhattan distance measurements for key presses. Your knowledge, guidance and leadership over the past year have been extraordinary, not only to the development of our thesis, but for us as officers and leaders. To our cohort, 368-181 and the members of *Team \$L@MDUN|<*, we could not have made it without your help and friendship. The struggle has been real, but fun. Last, but most important, we would like to thank our better halves, Finesse and Becca, for their love and encouragement throughout our time in Monterey Bay. Your patience, understanding and encouragement through our long hours spent tucked away while programming or studying for Automata exams cannot be overstated. We love and thank you.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:

Introduction

The reality is we've got to get it right all the time, they only have to get it right once. Emails are the number one [threat vector] delivery mechanism globally.

—David Bennett, Defense Information Systems Agency

In January 2018, David Bennett, director of operations at the Defense Information Systems Agency (DISA), speaking at a military-industry information technology conference in Arlington, Virginia, stated that the DoD stops approximately 36 million cyber-attacks per day, including phishing attempts by social engineers [1]. To combat these types of threats, the Fiscal Year 2019 White House budget includes \$8.5 billion for Department of Defense (DoD) cybersecurity for both hardware and software defenses, with approximately \$816 million going towards Department of the Navy (DoN) cybersecurity programs [2].

The Department of the Navy Chief Information Officer estimated that 91 percent of cyber hacking attacks start with a phishing or spear-phishing email [3]. Additionally, the Defense Advanced Research Projects Agency (DARPA) Broad Agency Announcement posted to their Active Social Engineering Defense (ASED) program website stated more than 80% of cyber-attacks target humans rather than network or security flaws [4]. Social engineers are a particularly troublesome subset of invaders who leverage anonymity to trick their targets. They assume a false-persona to remain unknown, or to masquerade as a legitimate user, and elicit sensitive information that can be used for an illicit login, extortion, or even espionage.

Anonymity provides a level of non-attribution and secrecy that makes stopping attackers difficult. In a fictional de-anonymized Internet, users would be assigned an immutable profile that followed all their actions online, and could be verified in a central database. Every action would be tagged with metadata associated with that action if someone else wanted to do so. If a user tried to initiate phishing scheme via email, the original message could have metadata that identified the true user as “John Smith,” with an actual IP address and Internet Service Provider, home address and other contact information. In this imaginary world, the government could visit his house and take legal action against him for trying to

break into a network. But anonymity prevents any of this from happening. Anonymity has a positive side, too, as we will explore in Chapter 2; anonymity is not always bad.

As part of ongoing research to protect DoD users from social engineering attacks, DARPA proposes a defensive botnet that can identify, intercept, and thwart attacks from infiltrating protected networks and shield DoD employees from these attacks [4]. Our research will go one step further and attempt to identify individual attackers by creating a profile from their keystroke pattern, a “digital fist” reminiscent of World War II radio operators, and over time, attribute any attacks to those individual users. We will incorporate “reflexive” biometrics, or how a user reacts to unexpected perturbations, as an added feature that can be used in conjunction with captured keystroke measurements.

While our system can be modified to thwart other types of social engineering points of entry, like chat rooms or websites, email phishing is the most prevalent attack on DoD users [4]. Our operational scenario is based on an environment where network defenses, to include spam filters, would identify phishing emails, and rather than quarantine or discard all of them, would selectively respond to a few of them and try and direct phishers to our chatbot. This would be done by making an excuse that “I am interested, but do not want to engage in this conversation on my government account. Let’s talk in this chatroom: <link>.” From there, our proposed system would take over and try to identify the anonymous attacker by engaging them in an online conversation through the chatbot. One common tactic is for the hacker to pose as a information technology help service employee, get the victim to allow them to remote-in to their computer for “distance troubleshooting,” and launch their attack from there [5]. These attacks require a robust interaction between hacker and victim. Emails that simply try to entice a victim to click a link and install malware are easy to quarantine and delete through standard network defenses, because the attacker does not need to have a conversation with the victim for a successful implant.

Our research will focus on whether we can identify individual attackers infiltrating a system. We will run an experiment to determine if we can strip anonymity from subjects interacting with a chatbot using their keystrokes and introducing perturbations in the system to generate reflexes in the attacker. Our hypothesis: we can better identify individual attackers targeting our system with the reflexive behavior versus measuring keystroke dynamics alone. While there are many ways a social engineer can attempt to gain unauthorized access into a

protected network, our work will focus on phishing attempts that have been flagged as malicious and intercepted by a chatbot. This process takes place prior to an attacker gaining access to a protected network. Chapter 2 will provide background research on the topics of anonymity, social engineering and previous work on keystroke analysis as a means of identification and verification in biometric authentication systems.

In Chapter 3, we will apply the background research towards our problem of identifying attackers trying to social engineer their way into a friendly network using keystrokes and reflexive behaviors. We will explain our use case involving defensive chatbots, our two-step verification and identification process using open-set recognition, and the specific implementation of our experiment to assess our hypothesis. We will also explain receiver operating characteristic (ROC) graphs, and other data analysis techniques using machine learning algorithms.

Chapter 4 will detail our experiment implementation and testing procedures using Google's DialogFlow as our chatbot, a locally hosted website running a Java-based keylogger, and our test subject population. We will explain how we extract keystroke features, subjects' responses to perturbations in the system, and how we build the collected data into distance vectors to determine which features provide the lowest levels of mis-classification. We will conclude the chapter by characterizing the different types of users we expect to see in our data based on how they authenticate with their own keystroke profile, versus other profiles in our database.

We will extensively discuss our results in Chapter 5: what we gathered, objective and subjective patterns with the subject population, and whether we can better identify individuals "attacking" with statistical significance using reflexive keystroke behaviors. We will explain our algorithms along with time and space complexity with our implementation, data storage, and analysis.

Finally, in Chapter 6 we will summarize our findings and highlight key areas that have gone unexplored in our research, and other areas relating to keystroke dynamics, reflexive behaviors, open-set recognition and social engineering for further exploration. Our background explorations into the topics that are helping us develop our experiment have already yielded applications that go beyond the scope of our thesis, and potential areas for future research. We will cover these topics in Chapter 6.

While our system will not be able to provide the true identity of an adversarial social engineer and stop the attacks at the source, it will provide additional information and data-points used in attribution, like cyber-telemetry data that includes IP addresses and “fingerprinting” browsers, by the intelligence community and law enforcement. By stripping away a layer of anonymity and track the number of individual attackers over time, we are attempting to aid DoD cybersecurity experts to better categorize threat vectors and thwart attacks based on individual actors.

CHAPTER 2:

Background

There is no technology in the world, nor will there ever be, that beats social engineering.

—Frank Abignale, renowned security consultant and social engineer

Our research covers multiple, established areas of study, but we will focus on three specific areas in this chapter to provide a background for our core research: anonymity, biometrics, keystroke analysis, and social engineering.

2.1 Anonymity

Anonymity has been used throughout history, with its roots from the Greek word ἀνωνυμία or *anonymos*, meaning “without a name” [6]. From a security perspective, anonymity is a roadblock on the path to the attribution of an action to an individual. In a cultural sense, attribution is related to the tie between an action and the actor. “The ‘what’—in other words the object or content of the action/communication—can even be an anonymous one, provided it gives no clue to identity” [7]. Humanity has grappled with the benefits and drawbacks of anonymity for a long time, and it has been proposed that one of the basic human failings is our inability to act justly under conditions of anonymity. This is an ancient theory that philosopher Glaucon (Plato’s brother) codified in the story of the “Ring of Gyges,” which is a story about a man, Gyges, who uses an invisibility cloak to mask activities he would not have attempted otherwise. His egregious acts include seducing a queen, killing the king and ultimately taking the realm for himself. Plato’s allegory from the story exhorts that under the guise of anonymity, anyone would act unjustly [8].

Anonymity continues to raise questions in behavioral psychology which leads to legal considerations. How does anonymity affect our actions? Should it be protected as a part of an individual’s right to privacy? Who should be able to take away a user’s anonymity to attribute actions to an individual? The internet is one of the largest experiments that has provided data to both sides of these arguments. Anonymity on the internet is often used as

the foundation for such behaviors as antisocial tendencies and pornography addictions, and cyber crimes such as pedophilia and fraud, but also for new online relationships and other instances of self-revelation [8].

2.1.1 The Internet and Anonymity

The internet’s design enables all users to be anonymous by default. Unfortunately, anonymous criminals and attackers have emerged as one of the biggest sources of cyber attacks. Their ability to act unjustly without detection is arguably the most critical flaw in our globally connected network. Online enclaves of bad actors across the public, deep and dark webs have thrived on anonymity to conduct such activities as human trafficking, political warfare, terrorism, childhood bullying, trolling, and social engineering cyber attacks. We will develop an approach that the DoD can use to mitigate the threats posed by anonymous invaders. Our research goal is to investigate the benefits and drawbacks of anonymity, describe how to protect anonymity when it is beneficial, and increase the security of our organization by developing a method to mitigate the threats posed by anonymous users.

The inherent allowance for anonymity has been exploited by hackers, trolls, and malevolent actors. Internet packets are encoded with their destination address, but the authentic originating address of a request is not an essential element. An origin address can be spoofed or shrouded in order to render the requestor anonymous, and the very nature of a packet switched network allows this, and is baked into the DNA of the internet [9]. Building upon the internet’s packet switched nature, web browsers connect computers and servers together in what we recognize as the modern internet. Still, anonymity in the packet switched network prevents content hosts from knowing which users were connecting to them. Users cannot be certain of the identity of an individual that they were conversing with in chat rooms, and that problem has been compounded with the rise of social media and the ability to create fake profiles based on real people, called “catfishing” [10].

Authentication, at a high level of abstraction, usually involves one or more of the following factors in order to enroll users in a system: something a user has (like a common access card), something a user knows (a password), or something a user “is” (biometrics) [11]. Any combination of these factors results in *multifactor authentication*. Websites, like Facebook, normally allow users to opt-in to multifactor authentication, whereby users not only have to

know the registered email address and password, but must also *have* a separate device, like a cellphone, where they can receive an authentication code that will also be required to be entered on the website before they can access their account [12].

These systems are not completely secure; an extremely capable attacker can steal log-in credentials and spoof devices that would otherwise be used for multifactor authentication. Additionally, users that sign up under a pseudonym, or multiple pseudonyms—which is a more deceptive form of plain anonymity with no associated profile—can be completely authenticated on a system without having their true identity known.

2.1.2 Malicious Anonymity

Cyber criminals masquerade as legitimate users in order to steal valuable private data, commit fraud, and steal money. A report by the FBI stated that in 2018, cybercrime reporting averaged almost 900 incidents a day, and an assessed value of \$2.7 billion in losses for the year [13].

According to that same report, Business Email Compromise (BEC) and more generally, Email Account Compromise (EAC), and the associated social engineering attacks resulted in \$1.7 billion in losses alone. Cybercrime is also difficult to prosecute, and for every cyber criminal caught and successfully prosecuted (using 2016 data), 100 go free. Anonymity presents the greatest barrier to prosecution as cybercriminals protect both themselves and their methods of attack. And if the criminal is identified, prosecution is difficult if the perpetrator lives outside of the jurisdiction of the victim [14].

2.1.3 Beneficial Anonymity

The concept of anonymity as a tool that is helpful or hurtful to society depends on the point of view of the individual in society. A Chinese dissident recording and publishing documents showing government corruption in Beijing would be seen as beneficial by fellow dissidents, but as malicious by Chinese leadership. Likewise, U.S. spies harness anonymity to fulfill the roles and responsibilities of the intelligence community. To the countries in which the spies operate, this same anonymity is harmful and represents a sieve of potentially sensitive information falling into unintended hands.

Military operations use anonymity to protect sources and methods of intelligence gathering. As operations move more into cyberspace, anonymity will continue to be the foundation of intelligence gathering, as well as covert and clandestine operations, that protect and further U.S. national interests. Anonymity online has produced wide-ranging effects, and has supplied concrete examples to both the benefits and drawbacks of anonymous discourse. Our system proposes to weaken the anonymity of individuals who are using it as a cover for malicious actions.

2.1.4 Anonymity as a Spectrum

Anonymity is not a binary concept, something that you either have or do not. It is more applicable to describe anonymity along a spectrum of weak to strong anonymity. A popular measurement that is used to describe this is called k -Anonymity. The security concept of k -Anonymity is that a data set provides k -anonymity if a set of data from a single user is common between k users [15]. This concept is important in online interactions because the users of a system like the internet number in the billions today which produces an environment that provides strong anonymity. Anonymity can also be strengthened by an individual or group. This is commonly seen online by the use of pseudonyms, or by the use of internet anonymizing services. Our understanding of anonymity is that it should be a privilege provided to individuals interacting online. In conversations with Vinton Cerf, an architect of the modern internet, one of the author's discussed anonymity using the highway system as a metaphor. Along the highways, even with a license plate, a user retains anonymity to the other drivers on the road. If that driver commits a crime, however, society has authorized a police officer or other representative of the law to view a driver's license in order to attribute that crime to an individual. Our system design is partly based on the idea of k -anonymity, and by gathering enough data to correlate a user's typing behavior across multiple chat sessions, we can attribute any future chat session to that user, thereby stripping a layer of the anonymity from the social engineers who require it to be successful in their attacks.

The internet is a paradox of anonymity, both making it fairly simple to obscure one's actual identity online, but still able to leak identifiable characteristics when aggregating online behaviors and actions, thereby weakening their assumed anonymity. For example, a study by the University of Cambridge and Microsoft analyzed over 58,000 anonymous

users' Facebook like history to determine if they could identify attributes like gender, sexuality, political leanings and relationships status. Their study concluded that they could identify ethnicity (African-Americans versus Caucasian-Americans) with a 95% accuracy, and gender was correctly classified 93% of the time—based only on Facebook likes [16].

k-Anonymity was designed to quantify these unintentional leaks of identifying information; the DoD has a similar concept called “classification by compilation,” where disparate, often open-source, facts when put together yields classified information. Similarly, when combining a user's digital fist through keystroke dynamics analysis and pairing it with other sources of information like an IP or MAC address or online account records, our system could potentially be used to aid in the real-world identification of an attacker. At a higher level of abstraction, our system would use k-anonymity to identify how to *best* correlate identifying pieces of information.

As an individual's anonymity is weakened, we approach the ability to singularize them, with the ultimate goal to identify them. By linking an individual, group, or nation to an otherwise anonymous action through areas like digital forensics and cyber telemetry, online actions can be narrowed down to two possible sources: a machine, like a chatbot, or a human. Our system aims to address the second determination by tying an individual to a machine using a biometric signature and technical forensics.

Technical forensics can be challenging, especially in an environment in which multi-stage cyber intrusions are conducted. Anonymity-enhancing tools such as The Onion Router (Tor) and Virtual Private Networks (VPN) can complicate a forensics analysis even more; such tools obscure technical information that might otherwise be used to identify a unique individual. Impeding technical forensics may serve a socially desirable goal when it protects people who engage in politically controversial dialog, but anonymity-enhancing tools can also be problematic when they help malicious cyber actors to evade responsibility for their actions and get in the way of identifying the actual machines involved in perpetrating an intrusion. Tor is an example of a tool used by both benevolent and malicious actors. Traffic routed through Tor undergoes automatic encryption and then “hops” between distributed global nodes to obfuscate both the path the traffic is taking, and the contents of the traffic itself. Organizations like the Electronic Privacy Information Center and Electronic Frontier Foundation offer lists of tools and best-practices that users can download and implement to

increase anonymity, but also makes it more difficult for law enforcement and intelligence professionals to conduct technical forensics [17].

Once an attacker is granted access inside of a network, stopping the malicious actions becomes a very hard problem. In the constant battle between cyber attackers and defenders, many researchers explore the behavior of malware once it has penetrated a system, and this struggle grows increasingly complicated as attackers find ways to avoid detection from defenders and anti-virus software. Our system is designed to stop an attacker from reaching a point where they can launch their attack. This research will explore the steps taken before malware is deployed on a network because the overwhelming majority of malware implants begin when an attacker targets the human user through phishing.

While phishing is the most common social engineering attack [18], other attacks exist that can be just as effective. There are dozens of different social engineering attacks, broken into categories like human-based versus computer-based, and technical, social or physical engineering attacks. Human-based, physical attacks can be as simple as impersonating an authority figure, face-to-face with a victim, or following behind authorized personnel into a secured area (tailgating). More technical attacks can involve malware installations by a someone impersonating help-desk personnel, or hacking into a Domain Name System (DNS) server and redirecting traffic to a site of the attacker's choosing (DNS poisoning) [5].

As complicated as each scenario is, many attacks begin predicated on one principle—the ability of the attacker to successfully lie about their identity. We will investigate the history of anonymity on the internet as well as the global breakdowns that are the product of online anonymity. As part of the exploration into the effects of anonymity on security, we will explore scenarios where anonymity bolsters the security of DoD systems, and the existing technologies that enable anonymity. These scenarios may be of use to Human Intelligence (HUMINT) collectors since source anonymity is a key aspect to their operations. To address the drawbacks, our research will explore harmful effects of anonymity in our society. A system's security will be fortified if we are able to cut off an attacker's ability to remain anonymous, and we will create an environment that does not allow anonymous users in order to validate this declaration.

Our thesis is going to focus on reducing an attacker or malicious user's anonymity through their keystroke behavior. This discipline of behavioral biometrics is already used by many

security companies and the military to identify legitimate users, or to target malicious ones. Our work focuses on identity creation and attribution within an anonymous environment like the internet. In this environment that is built to support anonymity, users can assume multiple identities in order to hide malicious actions by distributing these actions among multiple profiles. For example, users of social media can be tricked into thinking that a particular topic is trending, but in reality, that is caused by the work of one malicious individual using multiple identities, each of which gradually increase the visibility of the topic. Using the behavioral biometrics that are present in a user’s keystroke, our system will attempt to re-correlate these disparate identities into a single identity, which will simultaneously weaken the malicious user’s anonymity, and increase the ability to attribute their actions.

2.2 Biometrics and Feature Collection

Using keystrokes to identify a user is not a novel concept. In the beginning of the 19th century, when ham radios were gaining popularity, radio enthusiasts were able to transform the human language into electrical pulses sent through the air by using Morse code. In the early days of radio, Morse code was the only method to transmit messages to other users. This code was also loved by ham radio users as a way to have “privacy in public” since Morse was not widely understood by the public [19]. However, these users were able to identify each other by using the rhythm, pace and syncopation of the “dits” and “dahs.” During World War II, a similar methodology related to the tempo and cadence of military radio operators emerged to help them identify who was on the other end of the telegraph sending messages, known as the “Fist of the Sender” [20]. This type of analysis was used to help identify military operations and troop movements during the war. In the weeks preceding the attack on Pearl Harbor, the Japanese Navy—who were aware of this method of identification—sent shipboard radio operators on the aircraft carrier Akagi ashore. This deception tricked American intelligence into believing that Akagi was in the vicinity of the Japanese home islands when, in reality, the carrier would become the Japanese flagship in the attack on Pearl Harbor [21].

Our system proposes to leverage the concept of the “Fist of the Sender” and create a “digital fist” that uses modern keyboard patterns known as keystroke dynamics as a means of reducing the anonymity of a malicious user.

One of the weaknesses that are inherent in behavioral biometrics is that the profile query to a known template can change based on the state of the subject. For example, a user who is under the influence of alcohol will have a different typing characteristic than a sober user, and could be treated as a new attack. This is an especially important recognition when a system is attempting to lift a signature off of an adversary, and why the attacker must not know they are being monitored. It is our assumption that our measurements would be taken while subject is not aware that they are being monitored. Furthermore, we hypothesize that a subject will exhibit more distinct signatures if they are measured while they are subconsciously reacting to a stimulus. To test this, we developed three scenarios that will cause a user to react to a stimulus that is generated by our system. The stimuli will appear like system glitches or malfunctions, and we will measure the user's attempt to recover from the apparent mistake. We hypothesize that these reactions by the user will exhibit robust signature characteristics, and this approach will be explained in more detail in Chapter 3.

To create this environment that does not allow an attacker to be completely anonymous, we will develop a method of user identification that will remain unchangeable even if a user creates multiple accounts. Although we will not have access to an attacker's true identity in the outside world, we believe that if we can assign them an immutable identity within this environment, we will be able to stop more theft of DoD information. We will create this identity within our system by measuring behavioral biometrics of social engineers or phishers. During the interaction between the phisher and the victim, we will be able to observe the behavioral biometrics present in the way an attacker types – known as keystroke dynamics. Once created, this biometric signature will be difficult to mask even if the attacker creates multiple accounts to launch phishing attempts.

An attractive method to gather biometric features is through a “honeybot” that will pose as a phishing target. This will allow us to engage in conversation with an attacker to actively elicit their responses. The honeybot “victim” will lure the attacker into the environment by posing as a trusting user ready to deliver the sensitive details that the phisher desires. During a conversation, we believe that there will be more opportunities to build a robust attacker profile using both behavioral biometrics and system or network patterns.

A chatbot has been used to counter the increasing prevalence of unwanted phone calls. These

calls include traditional telemarketing calls, scams and robocalls which can be deployed to thousands of users very quickly. This countermeasure uses a phone bot that the creators have named Lenny which plays a set of pre-recorded messages to mimic a real persona. Lenny is able to keep telemarketers and scammers on the line for tens of minutes without them suspecting that they are interacting with a robot even though the responses do not incorporate any sophisticated AI [22]. Our system would be able to take a similar approach to attrite the time that a scammer has to spend on a victim. The system design is meant to counter a scammer in two ways, both using time. First, phishing scams rely on such a wide distribution of the scam material, the phisher has a limited time to interact with their victims, and can only focus on the most promising or vulnerable targets. By wasting their time conversing with a chatbot, the human behind the scam cannot engage other potential targets. Second, the more keystroke data our system can collect over time, the more data we can use to build a more complete profile in the database, and as a query for new attackers that may produce 15 minutes of keystroke data, vice two or three minutes. Our approach could be combined with the types of bots that DARPA proposes to use in the ASSED project. A bot that can control a small amount of resources (like a dataset of bank account numbers, the ability to transfer small amounts of funds, etc.) can really sell this kind of deception. Our system could help by extracting even more value from these interactions by establishing a method of identifying individual scammers using multiple pseudonyms to conduct their attack.

2.2.1 Keystroke Dynamics

Keystroke analysis goes beyond conventional keylogging, which is primarily concerned with what keys were pressed, by looking at how the keys were pressed. The dwell time of a key references a single key being pressed, and released; our system will measure each key dwell time. Another metric used to analyze keystroke dynamics is the keystroke latency, or the time between key pair actions. Our research will focus on the four most common types of latency times between key presses, all calculated from two timestamped event traces—a key press down, t^p , its release t^r , with the difference between the two times known as the duration, $d_{key} = t^r - t^p$: [23]

1. **Key Press to Key Release (PR):** This represents the time between the first key being pressed down, released, then a second key being pressed and released, also known as

the latency.

2. **Key Press to Key Press (PP)**: This is the time between a first key being pressed down, then released, and a subsequent key being pressed down. This timing vector may include overlaps, such as the *SHIFT*-key and a letter for capitalization.
3. **Key Release to Key Release (RR)**: Similar to PP, this represents the time from release of the first key to the release of the subsequent key. A trailing overlap here would include the *SHIFT* press, a press and release of a letter for capitalization, and then *SHIFT* release, which may be different and unique from PP timings for that key—does a user release *SHIFT* and the letter key at the same time, or do they continue to hold the *SHIFT* key for measurable time after releasing the letter key?
4. **Key Release to Key Press (RP)**: RP represents the time between the release of the first key and the press of the second. This is also referred to as flight time, because it measures the time it takes for the fingers to travel between keys. This represents the time between the release of a *t*-key

By exploring these keystroke latencies through various combinations of key presses, like timing the PP and PR for the *Q* key and *Q* to *U* keys respectively, thousands of combinations can be measured to create accurate models of an individual user's unique keystroke profile.

2.2.2 Fixed versus Free Text

Keystroke analysis is normally done under two distinct conditions: fixed text and free text. In fixed text, keystrokes are enrolled using the same input, typically a username or password, typed multiple times. This is common in authentication systems where the user not only needs to know the password, but also type it correctly based on the enrollment profile (who they are). Fixed text systems are limited in that they only compare the keystroke latencies of one set of characters to previously recorded latencies for that same set of characters with the assumption that an intruder may “know” the password, but would not “be” the authorized user and prevent a replay attack.

Free text is the opposite of fixed text: the user is able to type long passages of text freely and without limitations from previous text entries, like a password-keystroke, two-factor authentication system has. A free text analysis system must find the most relevant features of the keystroke input and then create a profile that can be matched against a similar, but

not necessarily exact, set of keystroke measurements. Our system is designed as a free text keystroke analysis system since conversations normally do not follow standard patterns [24]. Since conversations do not follow a fixed-text pattern, we will require our system to analyze keystroke timings using free-text conversations with the chatbot.

2.2.3 Biometric Identification versus Authentication

Biometrics includes automatic identification of an individual using physiological or behavioral traits [25]. Basic biometric authentication systems require an enrollment procedure to capture a user's biometric data as the first of two modes, and then queries a user against a profile object. That biometric data, based on a user's actions, is then paired with information that uniquely identifies the user as someone who is authorized within the system. Any anomalies yields a negative match, and that user is not granted access to the network or system, or if already granted access, could be removed from the system [26]. The next mode of a biometric system involves authentication: comparing a biometric query with known profiles based on observed behaviors between a user in the database and their keystroke. The authentication mechanism only allows those users with successful enrollments to access the system. This yields two distinct categories: identification and verification. Identification compares the query against all known templates in a database, and answers the question, "Have you been seen before?" Verification compares the biometric query against the claimed identity and answers the question, "Who are you?" [25] Our system will first rely on the identification features of an authentication system since an attacker's keystroke profile may be known or unknown, and answering the question of "Have we previously seen you attack our network?" Subsequent verification, if the query is known in the database, undergoes a "one-to-one" match to a specific, known adversary, answering the question "Of the attackers, who are you most likely to be?" [27]

We will go into specific details of our system in Chapter 3, but there are four common components: a sensor module, feature extraction module, matching module, and the decision-making module. The sensor module serves as the human-computer interface that will capture the biometric data [25]. The feature-extraction module converts the captured data into features that can be compared with the matching module. This module will take the converted data from the feature-extraction module and compare it with all enrolled profiles in an identification system, or those associated with the query in a verification system. We

will continue to use these terms when describing our system and methodology.

We have recognized at least two DARPA projects that have expressed interest in this methodology, and we will use similar methodology that have been developed in their programs to construct our proposed environment [4], [28]. In particular, DARPA’s ASED project seeks to employ defensive bots as an interactive border guard between the protected users and the identified attacker. The ASED project is broken up into multiple technical areas. We intend to leverage techniques developed by DARPA’s first technical area (TA1)—-attack detection—so that we can focus our experiment on the active investigation of social engineers. We will also leverage past DARPA projects, like their Active Authentication project and explore current practices of collecting biometric data.

2.2.4 Reflexive Keystroke Dynamics and Behavioral Biometrics

An under-explored area of biometrics research looks at reflexive movements for authentication; when subjects are exposed to specific, known stimuli in a biometric authentication system, and measuring the responses. An example that we will use in our work is changing the mouse cursor position while a person is typing, and seeing the reflexive actions a user takes to move the cursor back to the correct position. Other perturbations can be done passively, such as an observer or system that monitors for anomalies in a user’s keystroke dynamics. Our experiment will utilize active measures to induce reflexes in the user, and measure their responses against previously cataloged reactions. These measurements are then incorporated into a more common biometric profile, like an iris scan, to reduce two common attacks against a biometric authentication system: impersonation and replay. Impersonation involves mimicking a biometric signature to gain access to a system; replay involves capturing a user’s biometric signature as data, and re-send it to the authentication system. By measuring the focus times for a retina to external stimuli, for example, researchers were able to create a challenge-response biometric system that incorporated reflexive eye movements to authenticate users quickly and low error rates [29]. While our research is less concerned with the two different attacks since the system itself is under attack, we will use reflexive keystroke movements to external perturbations to determine if we can better define and identify unique profile, while remaining subtle enough to be undetected by the attacker.

A challenge/response format can also take the form of an “invisible challenge” - an event invoked into an online session that elicits a subconscious reaction that can be recorded as part of the attacker’s identity. An Israeli security company, BioCatch, uses this concept by causing a user’s cursor to disappear from the screen, and will measure the subsequent jitter as a user tries to find the cursor [30]. Prior limited research into behavioral reflexes and its application to biometrics will form the nucleus of our own research to better identify unique attackers as a metric for system security.

2.3 Social Engineering

Social engineers have been around in one form or another since time immemorial. Two of the most historical examples that include the tactics of trust and baiting include the Book of Genesis and the Devil eliciting trust from Adam and Eve in the Garden of Eden; in Greek mythology the Trojan horse was a baiting tactic used during the Battle of Troy. Various frameworks exist for social engineers to gain access to protected systems, [31] and our research assumption is that the attacker will utilize social engineering prior to gaining access to the protected DoD enclave. Social engineers continue to pose a threat to secure DoD systems, and by helping to identify them, one of our goals is to correlate their attack patterns and stop them before they are able to interact with an unsuspecting service member, civilian or contractor using a protected network. One prevalent framework was designed by security researcher Kevin Mitnick [32] shown in Figure 2.1.

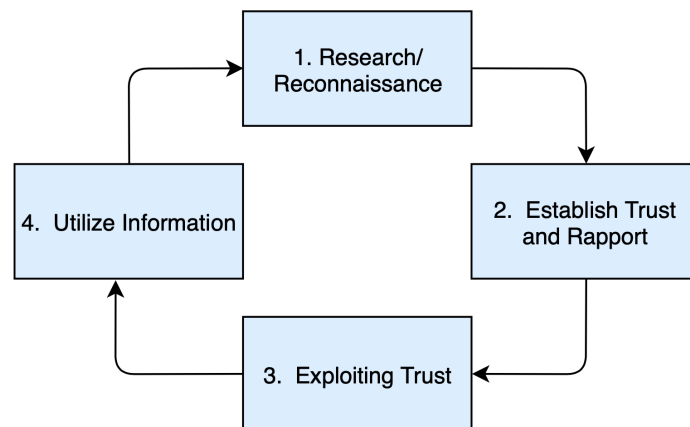


Figure 2.1. An example social engineering framework.
[32]

2.3.1 Phishing Attacks

The Department of Homeland Security (DHS) defines a phishing attack as “an attempt by an individual or group to solicit personal information from unsuspecting users by employing social engineering techniques” [33]. Phishing attacks were first seen in the mid-1990s during the rapid growth of the internet through America Online (AOL), and some malicious actors who congregated on the platform to exchange pirated files and conduct illegal activities. The first attacks began with social engineers utilizing AOL’s built-in conversation capabilities, like Instant Messenger, and inherent anonymity through the use of pseudonyms to pose as AOL personnel “verifying” billing information and other sensitive and private data [34]. With the success of these early attacks, social engineers began targeting more people and businesses on a larger scale, with a report from the Federal Bureau of Investigation reporting at least \$2.3 billion in damages from October 2013 to February 2016, spanning 17,642 businesses in 79 countries [35].

2.3.2 Reverse Social Engineering

From a friendly (DoD) point of view, our system is a form of reverse social engineering in two regards: first, the system is designed to have an attacker join up with our chatbot on a chat site. A deception story could be developed whereby the DoD employee victim is on a presumably “secure” government network, and only feels comfortable discussing the details of a social engineering email on a secure chat client. While the social engineer made the first attempt at contact through the phishing email, by having them connect to a government-controlled site that can extract and create a keystroke profile, we have the advantage. In one framework proposed by Mouton et. al. and built upon security expert Kevin Mitnick’s original process, building rapport between an attacker and target is a separate step from first establishing communication between two individuals [31]. We are looking to exploit communication establishment and rapport building by inserting our chatbot before the attacker could engage in conversation with a real human (green box in Figure 2.2).

By having a chatbot engage the attacker in conversation to elicit enough keystrokes to build a unique profile and then attempt to match it with previously seen profiles, the chatbot, in effect, begins to social engineer the attacker [36]. DoD counterintelligence (CI) and human intelligence (HUMINT) professionals are well-versed in the best ways to steer a

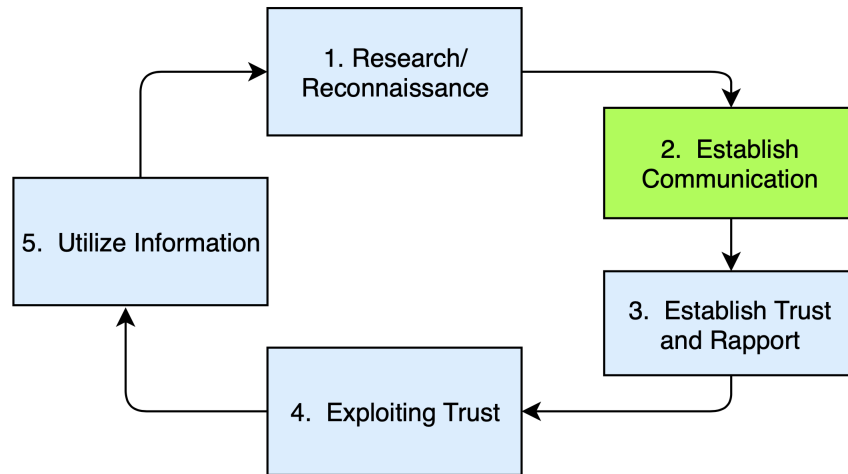


Figure 2.2. An updated social engineering model.
[31]

conversation to evoke targets to give up sensitive information. While our system only uses simple conversation templates that are not meant to gather sensitive information, by incorporating CI/HUMINT personnel in the design and development of chatbots, potentially more useful information can be obtained and correlated with the keystroke profile to build a more accurate attacker profile.

While email remains a common attack vector for social engineers to infiltrate a secure network, they are not limited to that medium [37]. Other attack vectors include baiting, quid pro quo, and vishing (voice phishing) [31]. Because our system is designed to be used after first-contact (attack) with a social engineer, any of the aforementioned vectors can be redirected to our chatbot system and be used to generate an attacker profile. After enough keystroke data is collected, the system can go into a defensive mode that acts in an abnormal way that deters an attacker from further conversation by informing them that their attacks are being logged or blocking their IP, for example. Ultimately, our primary goal is to design a system that could be deployed as a defensive botnet, ready to intercept and identify social engineers before they are able to gather enough information to infiltrate DoD networks.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:

Design and Methodology

Give a man stolen credit card and he'll eat like a king for a day. Teach a man to phish and he'll be set for life.

—Mark Russinovich, Chief Technology Officer, Microsoft Azure

There are various methods to measure keystrokes as a method for identification or authentication. Studies have generated keystrokes by having subjects copy long text documents with formal sentence structure, random words, or in the case of biometric authentication studies, repeated input of a username and password. In order to accomplish this type of biometric authentication, keystroke loggers are either client-side (on the local machine) or server-side (at the point of authentication). Client-side loggers can capture keystroke and mouse movements, and buffer them before flushing that data to the server for processing at regular intervals. Server-side loggers are similar, but capture keystroke timings from the packets being sent from the client and derive timings based on received time. Without robust enrollment procedures inherent in an authentication system, like entering a password multiple times to capture unique keystroke data, the enrollment of an unknown attacker will be based on deception: the attacker does not know they are engaging with a bot and covertly having their keystrokes analyzed.

Our system relies on the assumption that we are already interacting with a user who has been determined to be malicious, and does not perform an attack detection. By interacting with our system, an attacker unwittingly leaks more details about themselves through their keystroke dynamics, and we collect and store that signature. Collected signatures can then be used in the attack detection phase to thwart an attacker attempting to use a different identity. The system will determine if an attacker has been seen before through keystroke analysis to generate a profile, and then compare that profile to other logged attacker profiles in an attempt to group multiple attacker pseudonyms into a single identity; an overview of the system is shown in Figure 3.1. There are many military-based applications for this type of determination: determining if single-user accounts providing human intelligence are actually used by multiple persons, or determining the number of unique, malicious actors

actively trying to attack friendly networks. Our paper will cover the latter scenario in an effort to increase security of Department of Defense networks.

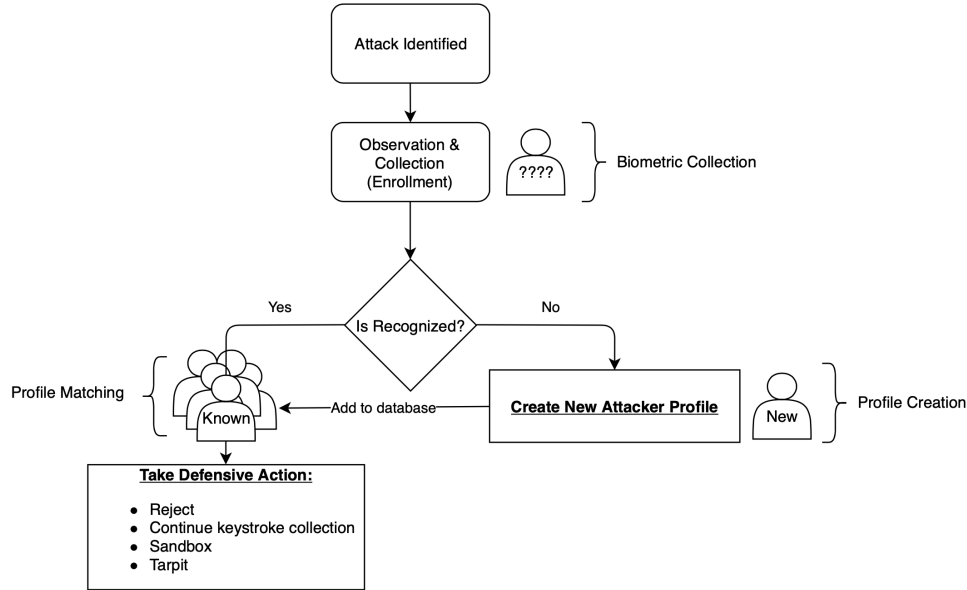


Figure 3.1. Biometric capture, creation and matching.

Multiple studies [25] [28] [38] have been done over the last quarter-century to determine what biometric measurements describe an individual well enough to be used as an authentication method. Previous research has shown that there are five qualities of a biometric characteristic that would make it suitable for identifying an individual: robustness, distinctiveness, availability, accessibility and acceptability. He correlated robustness with Type I errors and distinctiveness as Type II errors, and these two qualities combined will be the most relevant to developing our algorithms for capturing and creating an attacker’s keystroke profile. Type I errors are the probability that the queried sample will not match the correct profile in the database. Type II errors are the probability that a sample query will match to the wrong enrolled profile in the database [39]. Type I errors correlate to false positive (FP) rate and for our system is based on a previously unseen attacker incorrectly classified as someone who *has* been seen before, divided by the total number of attacker profiles in our database. The false rejection rate (FRR) is a Type I error based on the likelihood that the system will incorrectly label an unknown attacker as known. Type II errors correlate to the false negative (FN) rate and for our system would indicate an attacker who was previously seen, incorrectly classified as a “new” attacker divided by the total number of attacker

profiles. The false acceptance rate (FAR) is a Type II error based on the likelihood that the system will incorrectly label a known attacker as unknown. The true positive (TP) and true negative (TN) rates are the standard: an attacker correctly classified as being seen (TPR), or unseen (TNR), in prior attacks (Figure 3.2).

$$\begin{aligned} \text{True Positive Rate} &= \frac{\text{Known attacker classified as known}}{\text{total number of known attackers}} \\ \text{True Negative Rate} &= \frac{\text{Unknown attacker classified as unknown}}{\text{total number of known attackers}} \\ \text{False Positive Rate} &= \frac{\text{Unknown attacker classified as known}}{\text{total number of known attackers}} \\ \text{False Negative Rate} &= \frac{\text{Known attacker classified as unknown}}{\text{total number of known attackers}} \end{aligned}$$

Figure 3.2. Definitions of identification rates for the system.

When used in conjunction with other modalities for identification used for cybersecurity, like an attacker's IP address, MAC address, or other cyber "breadcrumbs," keystrokes can be used to augment an intelligence or law enforcement's approach to finding and prosecution of malicious, anonymous users trying to gain unauthorized access to our system.

3.1 Attacker Engagement

To create an environment that allows us to better identify individual social engineers attempting to infiltrate our friendly network, we will develop a method of user identification that builds on keystroke authentication methods as well as open set recognition.

Open set recognition is most commonly used in computer vision, where the generally, there are known objects that a system is attempting to identify, like a car or pedestrian, among a myriad of unknown objects that could be present in an environment, like a busy downtown area [40]. Modern authentication systems, like those that use keystrokes as a form of biometric authentication, use closed set evaluation: either the biometric query is matched against an enrolled profile and access is granted, or the query is not matched and access is not granted. Closed set recognition is used when a system has a sample from every

recognized class; this environment does not exist in our system because it is unknown at the time of query whether or not an attacker has a template already enrolled in our system.

By using a chatbot to elicit keystrokes from the attacker, we are able to gather a variety of key combinations, including punctuation and numbers, that are required to create nuanced keystroke profiles.¹ We will use a chatbot based on Google's DialogFlow to take advantage of Google's Natural Language Processing (NLP) capabilities, [41] with the elicitation timeline of approximately five minutes established by Alan Turing's behavioral intelligence test, colloquially known as the Turing Test [42].

Deception operations that follow a narrative that is both feasible and plausible tend to be the most successful [43]. In our system, at least initially, the attacker believes they are in charge. As they begin to interact with the chatbot, reverse social engineering takes place as anonymity is stripped through keystroke analysis. In this way, our system uses deception in two ways:

1. The attacker is deceived into believing they are interacting with a human victim.
2. The attacker is deceived into believing they are still acting completely anonymous.

Current research into the use of honeybots for offensive social engineering purposes has shown the median attack duration for a normal conversation with a human is approximately five minutes [44]. One goal of a defensive bot in our system would be to waste an equal amount of time of a human social engineer, and collect enough data to unique identify if the attacker has been seen before, and then identify the unique user and any other attacks they may have perpetrated.

While our experiment with subjects is limited to approximately to 20 minutes total, in reality a chatbot trained with custom content specific to social engineering and utilizing the latest NLP algorithms would be beneficial for prolonging conversations with an attacker, thereby collecting more keystroke data to provide a more robust attacker profile.

Our experiment will involve three separate elements to that end, all of which will record keystrokes and mouse-clicks: a log-in page that will accept the same password from each

¹For our experiment, the subjects will know they are having their keystrokes logged in accordance with our human subject testing protocols. An assumption, and rule for participants in the study, is that they not deliberately try to alter their typing behaviors to deceive the keylogger.

user based on prior work on keystroke dynamics: **.tieRoanl**, [45] and two separate chatbots. The ability to “copy” and “paste” from the keyboard or mouse will be disabled within our keylogging environment.

Our hypothesis: keystroke dynamics provides a means to accurately identify individuals, in the absence of *a priori* knowledge of their identity. Furthermore, we will explore the generation of a keystroke profile with greater fidelity through the use of reflexive challenges, and use it to analyze and identify whether or not a social engineer attacker has been seen before. To test our hypothesis, we will use a system with three distinct steps: biometric collection, profile creation, and profile matching.

3.2 Biometric Collection

Our experiment will utilize a Javascript keystroke logger that will measure the keys being pressed by the user. The lifecycle of a keystroke is as follows:

1. The key is pressed on the keyboard and closes an electrical circuit at a location that is mapped as key switches. The keyboard polls the circuit, and interprets the noise of a closed switch as a keycode integer.
2. The keycode integer is passed along to the keyboard controller, which encodes the signal for the computer.
3. On non-Universal Serial Bus keyboards, a keypress results in an interrupt on the interrupt request line (IRQ), which is mapped to the interrupt vector and is handled by the kernel via the Interrupt Descriptor Table (IDT).
4. At this point, a driver notes the key event and passes it to the application that is active and waiting for the keystroke.

Once at this level of abstraction, the keystroke has reached our test subject’s browser application, which is running the Javascript keylogger. The browser, by this time, has parsed the Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and Java code and if the user is typing into the "Password" input box, the application will send the signals to that box. Chapter 4 will go into more detail regarding the specifics of our website, chatbot and Javascript keystroke logger implementation.

The first step in our process will focus on keystroke analysis. Modern keystroke analysis

leverages advances in machine learning to increase the accuracy of identifying distinct individuals. Systems that use keystroke analysis as a form of biometric authentication are based on a closed-world assumption: a keystroke profile that is not matched to a profile contained in the known set n does not belong, and frequently used in keystroke anomaly detection. This breaks down into a two-step process: has the attacker's keystroke pattern (queried as a profile) been seen before, and if yes, which known profile does it most closely match? If the attacker has not been seen before, the initial profile that is generated for the query to the database is then added to the recognized profiles, otherwise it will go to a one-to-one correlation in step two [46]. Summarily, the first step determines *if* an attacker has been seen before, and if so, to *whom* they are correlated.

3.3 Reflexive Keystroke Dynamics

Prior research into using a challenge-response scenario to force a user to react to some external stimulus has primarily studied eye-movement as an augment to iris and retinal biometric authentication. One study used a visual stimulant to create a baseline for reflexive eye-movements in test subjects that would successfully lower authentication times and error rates, while increasing the detection of imposter (recorded) eye movements up to 99.94%. One of the goals of our system is to elicit the same results through reflexive behaviors using keystrokes [29].

To assess whether or not the introduction of unanticipated anomalies in our experiment help better define individual attackers, we will introduce two categories of perturbations: motor and cognitive.

The motor anomalies are designed to induce a reflexive behavior that our "attacker" subjects will have to correct in order to continue their social engineering attack:

- Losing focus (moving cursor to background)
- Jumping Cursor (moving cursor to start of input box)

These two motor reflexes will be split evenly across our samples size of $n=30$, with 15 subjects being exposed to the loss of window focus, and the other 15 subjects being exposed to a cursor that jumps back to the start of the input box. By measuring the reactions of the subjects to these perturbations and recover their cursor back to the appropriate position, we

hypothesize that we will be able to create unique attacker profiles that can be easily matched by themselves, but difficult to match by others. The key features we will collect involve what Banerjee, et. al. refer to as “editing processes,” which include three categories of events: use of the delete key, arrow keystrokes, and mouse clicks [47].

To measure cognitive reflexes, we will take place entirely using the FBI scenario chatbot. This will allow us to measure the keystroke timings between being asked unknown questions by the chatbot, and providing an unnatural answer that the subject will have to type from a physical piece of paper. Previous studies have shown that attackers will type deceptive content more slowly than a baseline speed captured while typing truthful or innocuous content [48]. For our study, we will assume the cognitive load of reading the deceptive answers from the piece of paper is negligible, or equal across subjects. Further research could explore measuring look-up times for deceptive content.

Specific details and implementation of our reflexive challenges will be covered in Chapter 4.

3.4 Profile Creation

To generate user profiles, we will transform keystroke timings into vectors and measure the distance between vectors using a Manhattan distance function to calculate nearest neighbors. These keystroke data points have been described as timing vectors, and the set of total users already classified is based on multi-class classification, shown in Figure 3.3.

Our experiment utilizes Manhattan distance calculations (*L1 norm*) over Euclidean (*L2 norm*) due to the high dimensionality of keystroke timings used to build the attacker profiles [45].

To minimize noise and variance within generated profile, we will utilize hundreds of elements within the vectors to create an attacker’s initial profile, and use that to query our database of known profiles (step one in our process). The free-text design of our system allows for more features that would otherwise be static in fixed-text entry (like the log-in page), but complicates the types of features that can be extracted as the keys, and number of combinations of keystrokes, must be typed often enough during the first step of our process to be statistically sound for comparison in later stages [49].

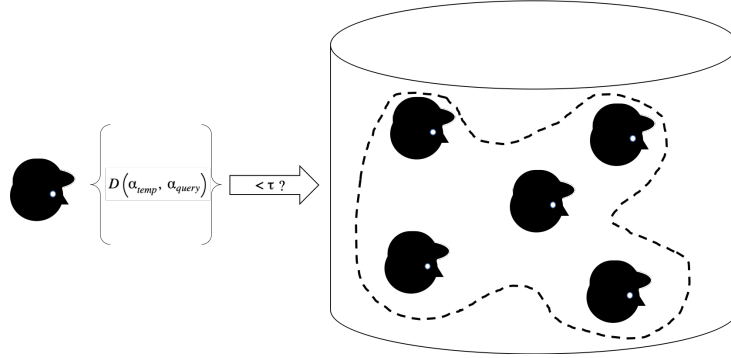


Figure 3.3. Profile creation and initial query to database.

A user’s signature vector will be modeled on previous work that relies on a hierarchy of digraphs to create vector features [50]. We will look at digraphs from the fixed-texts password collection, and a user’s recovery from perturbations. We will define recovery as the time of anomaly introduction which will either move the caret from its current position, or in the case of losing focus, to the caret returning to the correct location. They have a list of over 200 features that are generated from press duration times, transition times, and also use the means, and standard deviations of press times. We will leverage some of the digraphs that they were able to use, but will also generate features from mouse activity such as number of mouse clicks, scroll usage, and general mouse usage. Features can also be created based on total length of input time for a certain known string like a password or phrase, as well as a percentage-based feature generation for non-letter keys such as arrows, shifts, and backspace. We will be able to use the percentages of non-letter keys alongside mouse movement to determine a user’s preference for error correction which will be particularly useful during our reflex measuring portion [50]. In order to focus on the reflexes that a user will have to the system-generated perturbations, we will create reflex-specific vectors that generate features based on the time, and method that a user chooses to recover from the typing perturbation. More details about the generated perturbations are included in Chapter 4, and a full list of vector features is provided in the Appendix. Once vectors are created for a user’s trials, we will determine the distance that the vector has to others.

After the attacker begins a conversation with our chatbot and generates a profile query, α_{query} we will measure the Manhattan distance from the mean timing vector serving as a template for the average attacker profile, α_{temp} to determine if the attacker has been seen

before (Figure 3.4). Previous work has shown that a Scaled Manhattan classifier provided the lowest miss-rate for classification [51]. If this distance is below some threshold τ that we will chose prior to the simulation and keep it the same throughout, $D(\alpha_{query}, \alpha_{temp}) < \tau$, we will move on to the next step of the identification process. Until this point, the process is similar to established keystroke authentication systems based on fixed text sequences. This can be found in log in credential verification systems, where two-factor authentication is used: knowing the username and password, as well as typing it correctly. If the distance is greater than or equal to τ , we will enroll the captured profile as a new social engineer profile and adjust α_{temp} accordingly.

3.5 Profile Matching

Once an attacker has been identified as already in the database, the next step involves correlating them to the established profile. We will compare α_{query} to each cataloged attacker's template, v_{temp} in the database and link them to the assessed profile using a minimum distance function: $D_{min}(\alpha_{query}, \alpha_{temp})$ *if* $< \tau$. By continuously checking α_{query} against a threshold and then determining to whom the profile most closely matches, we can utilize this function as a free text identification mechanism, and establish a one-to-one correlation between α_{query} and v_{temp} (Figure 3.4).

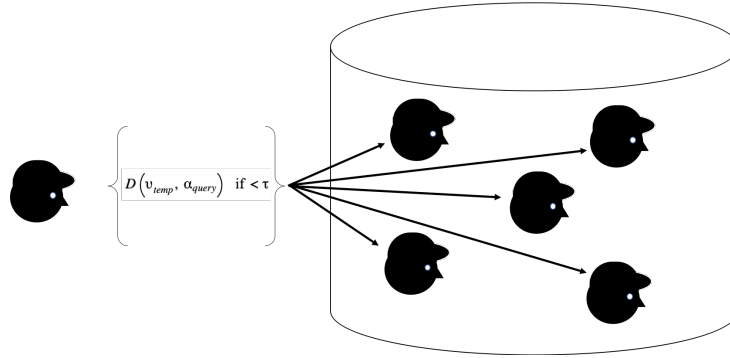


Figure 3.4. Matching known profile by querying to each user in database.

We will rely on receiver operating characteristics (ROC) graphs to determine the performance of our algorithm to match a user to a keystroke profile that may or may not exist in the profile database using a variety of training data captured from subjects interacting with our chatbot. For the first step in our experiment, we will be concerned with identifying a previous attacker in our data set (TPR) and the mis-classification of a previous attacker

being viewed as a new attacker that we previously defined as a false negative. We can adjust the ROC curve threshold to determine the ideal performance of our initial classifier. For step two, we will be concerned with using the ROC curve to characterize whom among all attackers, our queried attacker most closely correlates. For the analysis in step two, we can graph the TPR versus the FNR which is the rate that an attacker that has previously been seen is correlated to the wrong person. Since our focus in this stage is accuracy, we will tune our algorithm to maximize the zero-miss false-alarm rate, or the threshold where “the false-alarm rate is minimized under the constraint that the miss rate be zero” [45].

In order to compare the different ROC thresholds, we will use the area under the ROC curve (AUC) to reduce the ideal performance characteristic for each classifier, which is two-dimensional, into a single scalar value [52]. For the second step in our experiment we will utilize threshold averaging, which takes the threshold of each ROC graph and averages them across the data set. This threshold will be predetermined before the simulation is run, and will remain constant for distance vector calculations and determining membership in the database, as well as best-match to an already-enrolled attacker in the database.

CHAPTER 4:

System Implementation and Testing

As soon as man was born, he began to fight. When he began hunting, he had to paint himself different colours to avoid being eaten by a tiger. From that point on, maskirovka [‘something masked’ in Russian] was a part of his life. All human history can be portrayed as the history of deception.

—Maj Gen Alexander Vladimirov, vice-president of Russia’s Collegium of
Military Experts

Chapter IV will explain how our design is implemented in practice, system parameters, and detail test subjects and databases used during collection and analysis. Source code is attached in Appendix A.

4.1 Operating Environment

We designed the experiment and received approval by our Institutional Review Board for human-subject testing for current Naval Postgraduate School (NPS) students only. To alleviate scheduling concerns and time constraints, as well as protect student data, our experimentation environment was restricted to on-campus access to the testing website and keylogger, and implicitly controlled for time of completion to school hours without explicitly stating when students had to complete the scenario to provide maximum flexibility and increase student participation. By allowing our participants to conduct the experiment and interact with the chatbot at random times, but constrain the environment to an internal network, we were also able to mimic the deployment of the chatbot and collection of data against attackers at randomized times, but simulated attacks follows a circadian cycle, as they would in real life. This allowed us to test our experiment when network traffic and latencies in the system were representative of a typical day at NPS. We found that the DialogFlow chatbot sometimes had issues with latency and delays, causing our subjects to re-type their previous entry.

The process itself will include a known combination of machine learning components to accomplish its task: representation, evaluation and optimization. Representation is choosing

the classifiers that a machine learning algorithm can learn. For our system, we chose distance vectors comprised of keystroke duration and latency times. For evaluation, we chose a k-Nearest Neighbor algorithm, and a threshold determined by both the frequency and number of individual key presses, as well as digraphs between key pairs. Finally, our optimization takes place using receiver operating characteristics with the varying thresholds to determine the best mix of keys pressed, and frequency for both fixed- and free-text scenarios [53].

4.2 Chatbot Design

This system uses two chatbots to facilitate an interaction with a system user. In concept, we wanted to develop a chatbot that would pose as a vulnerable user, and a desirable target to a social engineer. We dubbed this bot a “honeybot” and programmed it to guide a would-be social engineer through a series of interactions designed to elicit both specific responses for analysis of keystroke dynamics for a structured pattern or phrase, and also allow for free-form phrases and responses. To accomplish this, our bot leads the user through a “choose your own adventure” style game. The user plays the role of a social engineer, and is instructed to use the identity of an FBI agent conducting an investigation as a pretext to get the honeybot to give up valuable personal data. The specific instructions for the game are included in Appendix B. The honeybot ends up posing questions to the user by asking them to establish bona fides. These questions will also serve to elicit a cognitive reaction from the user, and details about those types of reactions could be addressed in future work. A sample conversation between the honeybot and a participant is included in Appendix C.

Figure 4.1 shows the structure of the game using Twine—an application used for creating choose your own adventure games [54]. The goal was to visualize and create an interactive experience that caused the test subject to enter a diverse amount of data from the keyboard, and was focused on eliciting both familiar and unfamiliar or unexpected responses.

Dialogflow includes pre-made bots that are already trained and coded to handle common scenarios like small talk, banking transactions, and hotel reservations [41]. We leveraged the small talk bot and banking bot into our system to provide a way for the user to interact in an unstructured manner with the “victim.” Our second chatbot was only enabled to perform small-talk. Dialogflow allowed us to program the bot to lead the user through the game by creating several progressive states. Each state can be entered and exited by the user entering

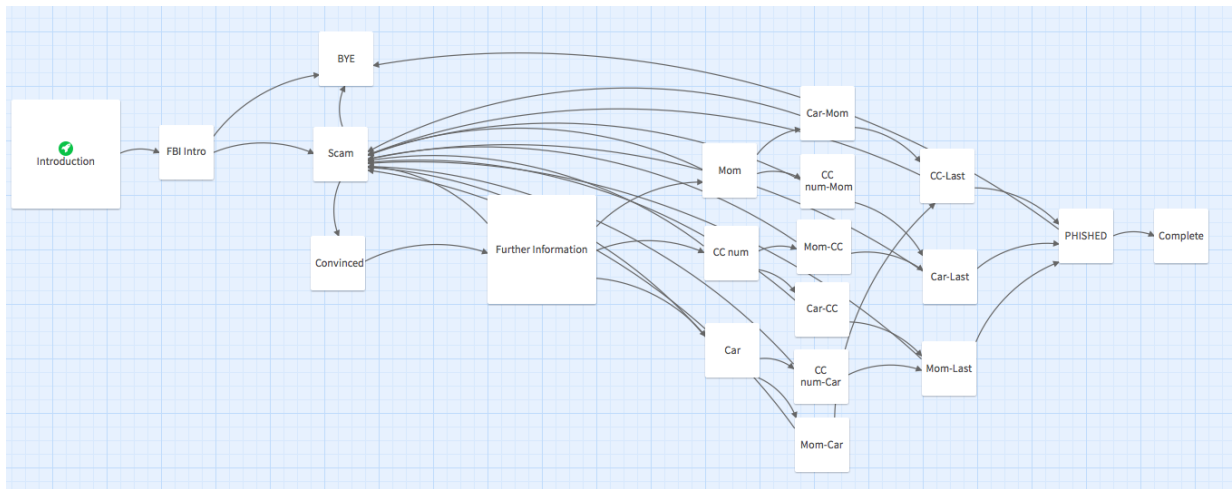


Figure 4.1. Game dialog layout in Twine.

specific phrases that the bot is expecting. The bot gives hints and clues, or requests for specific information to accomplish this. We used a feature called slot-filling to program the bot to ask for certain information before going forward. For example, if a user says, “I want t-shirts,” a bot that uses slot-filling would be able to ask for shirt size, number and/or color before moving forward to a transaction phase.

4.3 Web Interface

The web interface uses Flask, which is a web framework built in Python. This allowed us to build a Python web server, run JavaScripts on the page, and integrate the chatbot from Dialogflow. When a user enters a query to the chatbot, the text is sent to our Dialogflow bot via a JavaScript Object Notation (JSON) request. The Dialogflow bot will process the query, and send the response in another JSON response.

4.4 Keystroke and Mouse Movement Collection

Keystrokes and mouse events collection are enabled by using a JavaScript function. Each time a subject presses down on a key, an interrupt request is sent to the CPU. The CPU then maps the interrupt request to the interrupt function that is supplied by the kernel and executes. During execution, the interrupt function will read which key was pressed from the keyboard, and decide to pass control to the program that will take action on the key press.

When the program taking action is a web browser, we are able to link JavaScript functions to actions taking place within that window as well.

Our system sets a JavaScript callback on the window object of the running browser. This allows us to collect all keystrokes that are made while that process is an active window, not just keystrokes that are part of a particular element within the window. The most specific element within the window will see the key event first, and as shown in Figure 4.2, that event will then “bubble upwards” to the window object. A callback is made on every event type that we have specified, namely: key-up, key-down, mouse click-up, mouse click-down, mouse-motion, and mouse-wheel. Events are logged and placed into a buffer on the client machine, and are pushed every 2 seconds to a server which will put the logged events into a comma-separated values (CSV) data file, and clear the buffer to prepare it for more input.

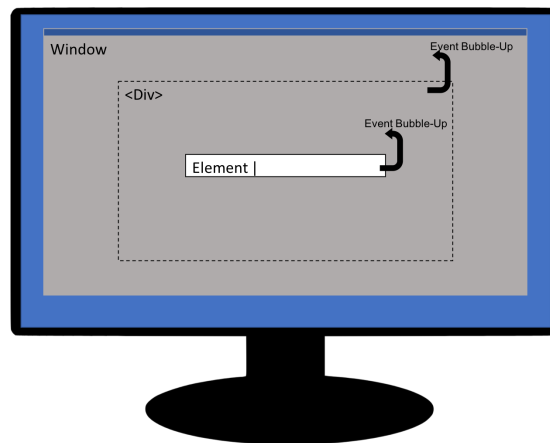


Figure 4.2. Events bubble up from most specific element to window.

4.5 Reflex Collection

Perturbations that elicit reflexive reactions in a subject’s typing style to the are also implemented with JavaScript. The first perturbation that we introduce is one that will cause a loss of focus on the current text box element within the web page in which the subject is typing. As shown in Figure 4.3, the function `changeFocus()` is called on the key-up event, and will be called after every keystroke made by the subject. `changeFocus()` will check if the subject has typed enough characters by comparing the length of their string at the time of call to a local threshold variable that we set to three. After three characters have been typed, the active element (the text box in this case) will be blurred and to the subject, it will

appear as if the cursor has left the text box, and their keystrokes are not being logged. In the back end, we are still able to see the user's keystrokes after the blur. We expect that the user will attempt to recover by moving the mouse back into the text box and clicking somewhere towards the end of their visible text.

```
$("#password").on("keyup", changeFocus);
JavaScript Function:
function changeFocus(e, thresh = 3) {
  if ( typeof this.done == 'undefined' ) {
    this.done = false;
  }
  var chars = $(this).val().length;
  if ((!this.done) && (chars >= thresh)) {
    document.activeElement.blur();
    console.log('Changed focus');
    this.done = true;
  }
};
```

Figure 4.3. changeFocus() function call from webpage.

The second perturbation that we introduce is called moveCaret() shown in Figure 4.4, and will cause the caret to jump backwards causing an error in what the user is typing. We have chosen to jump the caret backwards to the position immediately following the first character typed by the subject. So, if the subject attempts to type the word “PASSWORD” without looking up at the screen, the caret jump will cause their entry to be “PSWORDAS.” With this perturbation, we are going to observe how long it takes a user to identify and correct the error. We anticipate the difference between catching the perturbation immediately, versus a continuation of typing to be the difference between a touch-typist and non-touch typist.

4.6 Scenarios

The first chatbot scenario asks the subject to become a social engineer posing as an FBI agent, trying to social engineer the chatbot using information that we have presented as previously collected open-source information about the honeybot. The information is printed on paper, to dissuade any attempts to copy and paste data, even though those two

```

$("#password").on("keyup", changeFocus);
function moveCaret(e, thresh = 3, caretPos = 1) {
    if ( typeof this.done == 'undefined' ) {
        this.done = false;
    }
    var chars = $(this).val().length;
    if ((!this.done) && (chars >= thresh)) {
        var elem = this;
        if(elem.selectionStart) {
            elem.focus();
            elem.setSelectionRange(caretPos, caretPos);
        }
        else
            elem.focus();
    }
    console.log('Moved caret');
    this.done = true;
};

```

Figure 4.4. moveCaret() function call from webpage.

editing features are disabled in our experiment. The users will be able to reference the victim data as they interact with the victim chatbot. Their goal is to extract pre-scripted information from the chatbot by asking the right questions, and also providing the right answers as the chatbot questions their identity as an FBI agent. This scenario will introduce a cognitive reflex during the anonymous FBI-agent phishing scenario. By engaging the user in a challenge/response scenario, we hope to put a cognitive load on the test subject as they have to respond to questions, look-up answers, and type exact numbers and phrases into the text box to continue the game.

The second scenario involving the “small talk” chatbot is meant to establish a second set of control and test reflexes on the login page, as well as gather keystroke dynamics from the regular interaction with the chatbot as a free-text control dataset.

For the scenarios: the “control login” is without perturbations and “control chat” is our small-talk chatbot; the “motor-reflex login” is with our active perturbation, and the “cognitive-reflex chat” is the spoofed-FBI agent game.

1. Scenario A: Control login, cognitive-reflex chat
2. Scenario B: Motor-reflex login, control chat
3. Scenario C: Motor-login, cognitive-reflex chat
4. Scenario D: Control login, control chat

4.7 Measurements and Calculations

Our experiment will capture keystroke digraphs and mouse movements that will help determine if adding perturbations to a user chat session will increase a system's ability to identify, and then verify, an individual attacker. To create useful information from the raw keystroke data, we will calculate mean dwell times for a single key press, as well as latencies between two key presses known as flight times. We can create a feature vector comprised of various and specific types of latency and dwell times from that information. Our implementation of the system uses the following features:

1. Key Press Duration: Mean and Standard Deviation of a key press
2. Key Press Percentage: The percentage that a particular key was pressed out of the total keystrokes.
3. Flight times between two keys: The press-press, press-release, release-press, and release-release times between every pair of keys typed by a user.

Every one of these features is assembled into a feature vector, which can be compared against other vectors. A user is said to match when that distance is small, and will not match as that distance increases. For comparison, we tested vectors based on whether we introduced a perturbation and in doing so, we expect to see an increase in match scores in sessions that contain cognitive or motor reactions.

Our collected data is split into two sets: a training set and a test set for feature extraction. We ran the training data through a k-Nearest Neighbor classifier included in a Python *scikit-learn module*, and then ran our test data to determine the fit of our model [55]. Initially, our system assigns a "0" to any feature that was not observed during a session. Fall-back measurements [50] could serve to increase the match scores of user data in future work. Whether using a "0" placeholder or the fallback values for keystrokes, we ensure that our feature vectors are complete before they are run through our classifier. We will discuss fallback measurements with respect to our results in Chapter 5.

We extract vectors that are comprised of dwell and digraph latency, as well as timing measurements as features in columns within a CSV file. We pair the “feature” with each “user” in order to train the classifier. We then run our test data through the same extraction process, but when we run the test data through the classifier we compare the results to the true labels to determine how many users it correctly classified. We verify the accuracy of the model using the unique identification label assigned to each user. The number of users correctly classified divided by the total number of users (“known attackers” in our system), becomes the identification accuracy of our classifier for those selected feature vectors.

CHAPTER 5:

Results and Discussion

Companies spend millions of dollars on firewalls and secure access devices, and it's money wasted because none of these measures address the weakest link in the security chain: the people who use, administer and operate computer systems.

—Kevin Mitnick, cybersecurity researcher and social engineering expert

5.1 Results

We extracted all of our features from a data set of all keystrokes typed by the 25 users that completed all four typing sessions. The full Python implementation is attached in Appendix A. For every user, we created a vector using the features described in Section 4.7. We then broke out the data based on whether or not we had introduced a perturbation in the typing to cause either a motor or cognitive reaction in the user. The feature vectors were collated into the following four sets:

1. Normal login
2. Reflex login
3. Normal chat
4. Reflex chat

When a feature was not observed - for example the (q,p) digraph was never used—we inserted a 0 in that feature's place. Ideally our system would implement the fallback procedures that has been used in prior keystroke research [50], after enough iterations and learning has occurred. If this method is implemented, the system would fall back a level in a typing hierarchy to determine a feature to use in place of the missing one. To substitute the flight times for the (q,p) example, the system could substitute the average flight times for any digraph that used the left hand first, then the right hand. Although this is an approximate substitute, it would increase the effectiveness of the feature vectors. Our study did not collect data to accurately reflect these types of digraphs, and we will discuss more of this in

Chapter 6 for future work. Instead, our system varies the number of zeros that are included in a feature vector over the entire dataset. For example, with a 0 threshold of 30%, if the digraph (q,u) is not observed for 35% of all users then it would be discarded. We drop any column that contains all zeros, and then try running the data through the classifier 10 different times, each time decreasing the percentage of zeros we allow within a particular column.

5.2 Scenario Test Results

Once the vectors were created, we ran them through a K-Nearest Neighbors classifier, and took the mean of the match scores from 100 iterations of the classifier results. We also calculated results when varying the proportion of 0-substitutions in a user’s digraph. To adjust 0-substitutions, we dropped all columns that contained only zeroes beyond a defined threshold, between 0.1 and 0.9. This meant that when the classifier ran, it would not consider any columns that contained more than some percentage (between 10 and 90%) of zeroes, and only looked at data that was produced by user keystrokes rather than non-events as a way to normalize the results among all users. Single key-press zeroes would indicate the key was not pressed at any point during testing. Using this methodology, we ran the classifier for incremental increases of 10% for all variations of our experiment: normal versus reflex logins, and normal versus reflex chats. When plotted on a graph, it shows the 0-substitution threshold that results in the greatest difference in accuracy between normal/reflex features.

5.2.1 Only press and percentage of key press

Figure 5.1 shows the change in accuracy when adjusting the threshold of 0-substitutions for both Normal and Reflex Login using only the press duration features. The separation distance is noted between the red (Reflexive classification accuracy) and blue (Normal typing classification accuracy) lines. The results show that when only comparing user-supplied keystrokes, the classification accuracy of our system was consistently higher with the introduced perturbations versus normal login conditions, across all 0-substitution thresholds.

Using only press duration features, Figure 5.2 shows the results for the chat sessions. Unlike the login, the difference with cognitive reflex typing was less pronounced than the login page. Because we disabled the “copy and paste” functionality in the system, we theorize

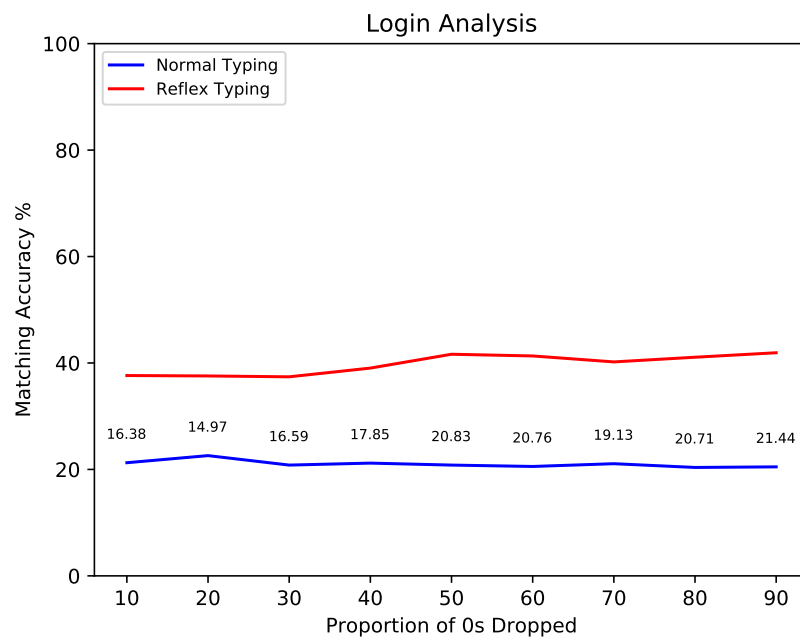


Figure 5.1. Normal versus reflex login accuracy - press features

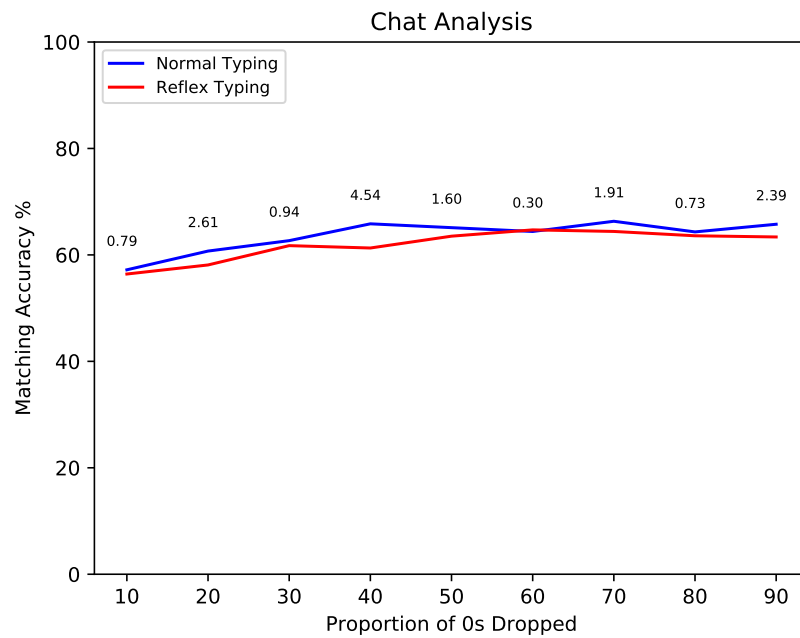


Figure 5.2. Normal versus reflex chat accuracy - press features

based on simple observation that the movement back and forth between our provided “Victim Information Sheet” and test subjects’ keyboards could have initially increased latency between key presses. The normal chat scenario was comprised of short questions that the users could ask the chatbot, and did not require them to look back and forth at a data sheet to determine what to type. If the subjects performed the second “attack” in quick succession with the first, they may have retained some of the information and would not have as much hesitation between the chatbot’s responses, which would be more indicative of their normal typing style.² It is important to note that for the chat sessions, the greatest accuracy difference between the reflexive and normal chats was when we adjusted the 0-substitution threshold to 40%.

5.2.2 Results with Digraphs

Before we ran our classifier using digraphs, we extracted features using the top 5% and top 50% of all digraphs observed, by use. Figure 5.3 indicates that as empty values are dropped from the columns of top-5% of digraphs using the login classification, accuracy increases at a greater rate for reflexive keystroke dynamics than the normal login, which remained fairly even across 0-substitution threshold. For the login perturbation, our research shows a greater level of classification accuracy over normal keystroke behavior accuracy as a subject’s reaction to perturbations was similar across both reflexive chat logins. This could be attributed to the increased number and variety of keys pressed to correct errors, over the regular password input with no errors. We found that lowering the 0-substitution threshold below 40% yields no discernible difference between the reflex and normal logins.

Our data shows that the top-5% of digraphs used during the chat sessions yielded results that showed that the classification accuracy for our controlled normal chat was much higher than the reflexive chat, with an increase of almost 14% across our dataset (Figure 5.4). As the 0-substitution threshold moves past 70%, the gap in classification accuracies between the two chats decreases as more common digraphs yield a better classification accuracy for the reflexive chat and 0-substitution noise is eliminated.

²We wanted to maximize our pool of student-subjects by allowing them to participate on their own time, so we can only theorize on any cognitive chat discrepancies based on those subjects we were able to monitor in a controlled setting.

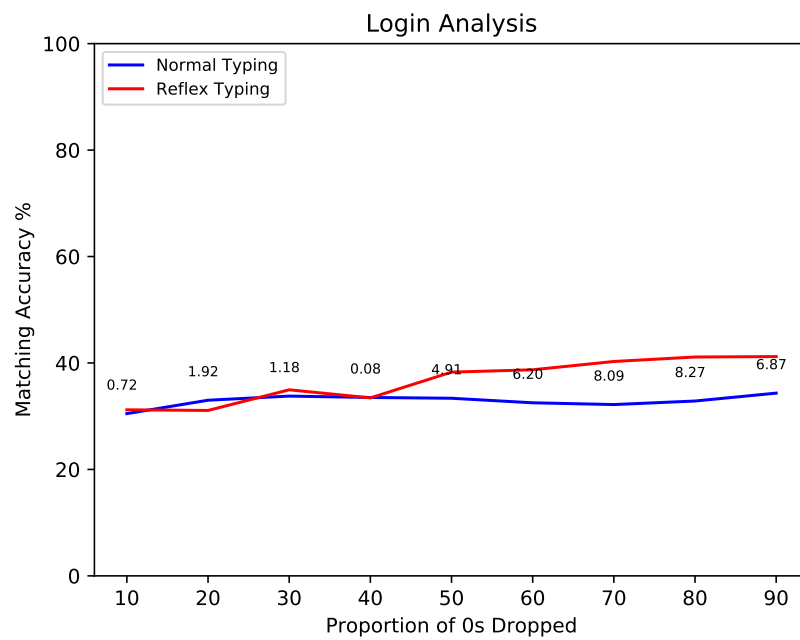


Figure 5.3. Normal versus reflex login accuracy - top 5% of digraphs used

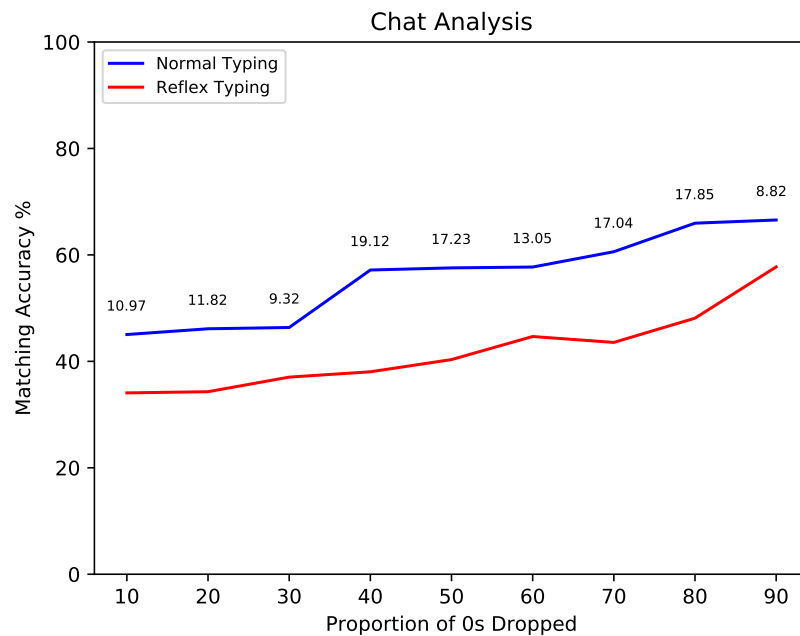


Figure 5.4. Normal versus reflex chat accuracy - top 5% of digraphs used

The last feature extraction included the top 50% of digraphs observed across all user sessions. Figure 5.5, shows our classification accuracy after increasing the digraph features, and we saw a substantial gain in the classification accuracy, as compared to Figure 5.3, which saw a nominal increase (between 4%-6% increase after 50% threshold). Gains over the classification accuracy went as high as 22.85%, with accuracy increases over 10% across all thresholds. Additionally, we saw overall higher rates of classification accuracy for the reflexive login using 0-substitution thresholds at 50%.

For the top 50% of chat digraphs as shown in Figure 5.6 the classification results were significantly different than the results using only the top 5% of digraphs. The classifier using 50% had a similar accuracy for the reflexive chat, but the accuracy of the normal chat dropped, and was lower than the reflexive chat across all 0-substitution thresholds. We found the greatest difference in accuracy between 50 and 70% of 0-substitution columns dropped.

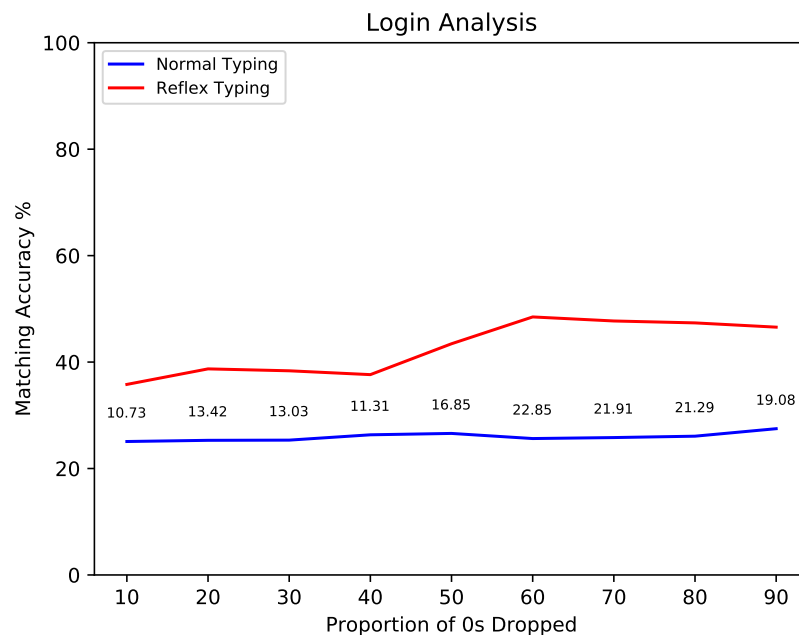


Figure 5.5. Normal versus reflex login accuracy - top 50% of digraphs used

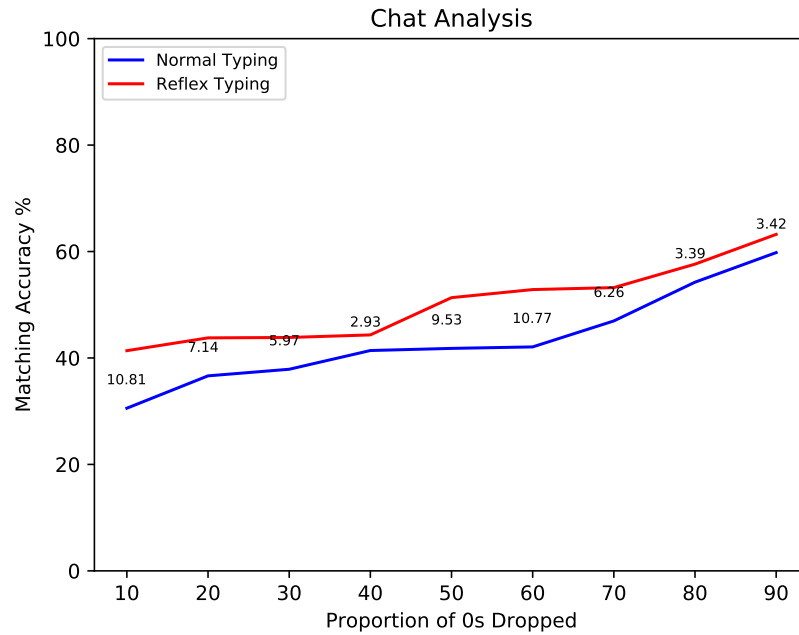


Figure 5.6. Normal versus reflex chat accuracy - top 50% of digraphs used

5.2.3 Other Observations

While we were not able to observe all of our participants completing the scenario, what we did see of those that finished all four rounds in front of us was the relationship between the amount of a user's keystrokes and caret position during a reaction during the reflexive login. This observation led us to take a closer look at the caret position/total keystroke relationship as part of a user's ability to recover from a perturbation. For example, we think that the time to recover from a perturbation could be measured from the time a perturbation starts, until the time a user has the caret back at the last character entered in the text field. Using this observation, we were able to plot keystroke and caret position graphs and found that almost all users' graphs looked very similar during normal typing, and much more unique to each individual during a reaction to our perturbations. Between the two reflexive logins, reactions to the moveCaret perturbation created more unique plots than changeFocus. When the user lost focus, the only variation between users seems to be the time it took to recognize that their keystrokes were not showing up on the screen.

Using both login scenarios and our observations, we theorize that a user's reaction to abnormal caret behavior was unique among individual test subjects, and attributed mostly

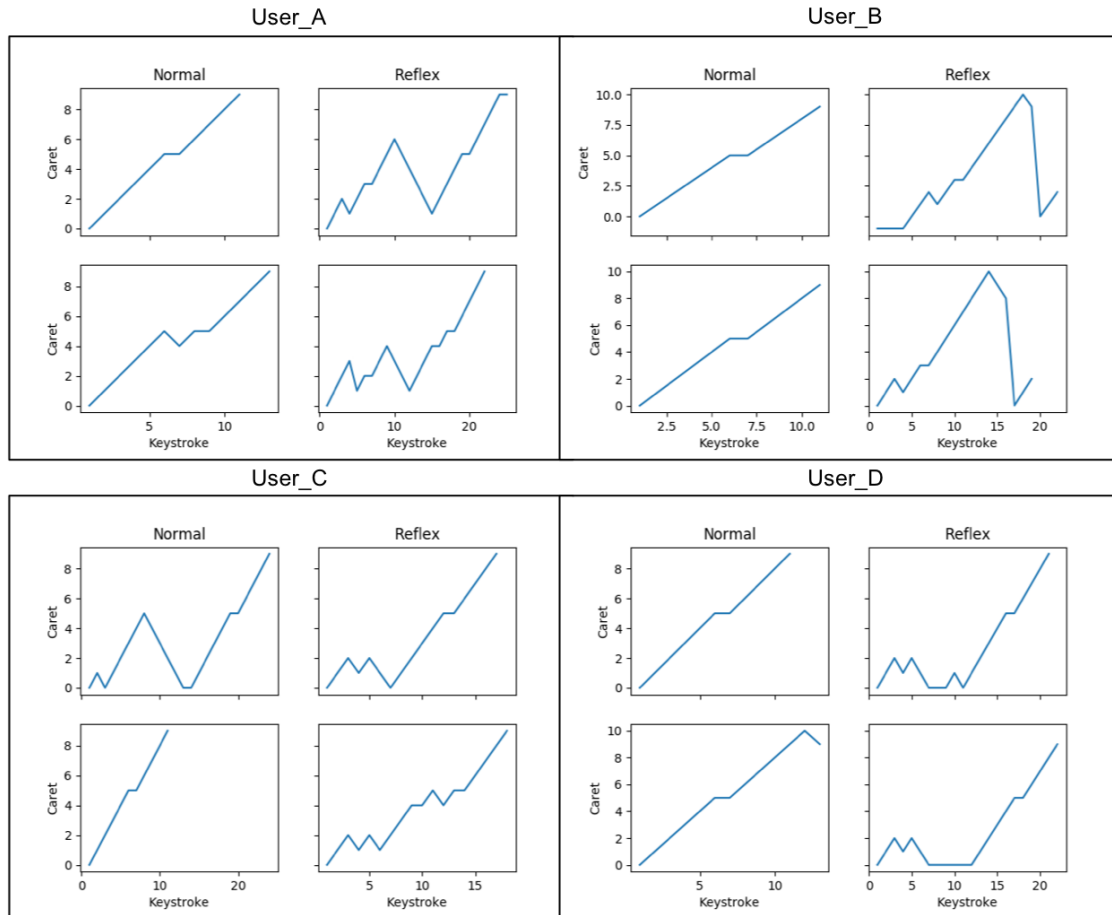


Figure 5.7. Differences in users' normal versus reflex login patterns

to subjects not looking directly at the password input box or caret position while typing in the password. Two observational areas of eye-focus were on the password “.tie5Roanl” on the screen, or their keyboard typing the password, thereby taking their gaze and attention away from the actual caret position after introducing the reflexive perturbation.

Figure 5.7 plots four users' reactions to the perturbations by measuring the caret position versus the number of keystrokes. Our password, without any anomalies or mistakes, should take 11 key presses: ten for the characters, and either use a *SHIFT* to form the capital *R* (11 keystrokes), or turning on (and off) *CAPSLOCK*, for 12 keystrokes. When compared against the keystrokes for a particular user, we can reconstruct the exact pattern that a user employs to type in the password correctly. We can see if a user decided to tap *BACKSPACE* in order to recover, drag and delete with the mouse, or pressed the *BACKSPACE* key once,

and held it down to start over. This type of analysis is known as stylometry and will be discussed in Chapter 6 and left as further research opportunities.

5.2.4 Biometric Menagerie

Designers of biometric systems must also be aware of a phenomenon that users exhibit when trying to authenticate themselves using biometrics. To aid in the discussion of the classification accuracy of different attackers, users have been categorized into a *biometric menagerie* [56]. Users of this system will fit into three specific member classes from prior research: “goats,” “lambs,” and “sheep.”

Goats are individuals whose keystroke profile is difficult to match against themselves and others, yielding false negatives. For our experiment, goats would represent attackers that have tried to social engineer their way into a network before, and their latest profile will not correlate to any known profiles in our database.

Lambs are individuals who have characteristics in their typing patterns that make them match against not only themselves, but others in the database. These types of attackers would yield false positives as their keystroke profile is correlated to a match against another attacker’s profile.

Sheep represent the baseline for our keystroke analysis system, and the goal for any biometric system: they match well to their enrolled profile, and no one else. By adjusting our thresholds and introducing reflexive biometric challenges during an attacker’s conversation with a target, our goal is to hone the profiles to create as many sheep from the attackers as possible.

The last category of individuals are “wolves,” who would represent an attacker or social engineer who knows a system is generating keystroke profiles, and attempts to impersonate someone else’s typing pattern. While wolves represent the primary threat to bio-authentication systems, we will assume in our study that the social engineer is not aware that they are being profiled by their keystrokes. Wolves will be addressed in Chapter 6 as a threat or shortfall to our profiling system. [56].

Our data does not break out the difference in lambs and goats and groups them as a single mis-classification, since we are only concerned with the correct classification rate (sheep)

of our system. We will leave the lamb and goat breakout for future researchers, and how to best optimize the system to minimize their presence in the results, but we found it useful to describe different attacker matching in terms of the menagerie throughout our study.

CHAPTER 6:

Conclusions and Future Work

We believe that data is the phenomenon of our time. It is the world’s new natural resource. It is the new basis of competitive advantage, and it is transforming every profession and industry. If all of this is true—even inevitable—then cyber crime, by definition, is the greatest threat to every profession, every industry, every company in the world.

—Ginni Rometty, Chief Executive Officer of International Business Machines
(IBM)

6.1 Conclusions

Our research has shown that adding perturbations into a keystroke authentication system enhances our ability to identify an individual attacker, and verify who they are in our database because individuals exhibit characteristic reactions to the stimuli. Both quantitative and qualitative accuracy depend on which type of perturbation is used but in general we saw increases in classification accuracy especially with the fixed-text, password scenario. We observed an increase in accuracy over all types of feature extractions in both login and chat scenarios except for the chat session that only used the top 5% of digraphs as features.

We successfully created a “choose your own adventure” game where participants played a social engineer to test our chatbot and keylogger in a controlled environment. By using the game format, we were able to watch and measure participants’ ability to manage cognitive tasks when asked questions by the bot and provide a measurable cognitive reflex environment that was realistic by not being scripted. The applicability of the game goes beyond phishing emails, and could apply to any research area that involve conversations that need to be realistic and fluid, yet measurable for analysis. This type of game could also be used to help develop chatbots with a better understanding, through pattern recognition, of social engineering attempts and attempt to drive the attacker to give up information about themselves, or direct a conversation to elicit more keystrokes and develop a better attacker profile.

Using a k-Nearest Neighbor algorithm creates a model that allows us to determine if we've seen an attacker before, and if so, to whom in the system the profile closely matches. By stripping anonymity away from social engineers, at least enough to distinguish individual attackers from one-another, our system can help protect DoD users from unknowingly engaging attackers and potentially allow a breach into a friendly network. Identifying the rate of attacks could also help network security administrators charged with protecting networks determine the best strategy for deploying the chatbots against the social engineers, and without losing the element of deception. This system could dole out responses in a manner that does not alarm a Social Engineer with an overwhelming response; to illustrate, an attacker that expects to see one out of every 2,000 phishing attempts result in a reply from the target, now sees dozens at once and becomes suspicious. By correlating attacks, network administrators, law enforcement and intelligence professionals can recognize patterns like time-of-day, common phrases, and typing rates of the attacker. When matched with known IP addresses, all-source intelligence about a subject, and other pieces of cyber-telemetry, an analyst can then use that data to provide fidelity to military commanders and law enforcement agencies as part of intelligence correlation for further prosecution.

As stated in Chapter 1, the impetus of our thesis was the staggering amount of money and hours the Department of Defense spends on cybersecurity, only to have it circumvented by a phishing email that either leads to a conversation where a victim unknowingly provides sensitive information to an attacker, or clicks a link that provides the attacker a technical backdoor into DoD networks. Our primary goal was to be able to identify the number of unique attackers at a greater rate than was previously possible through keystroke dynamics analysis alone.

While we were able to support our hypothesis that the addition of anomalies to force reflexive behaviors could better identify attackers than through standard keystroke analysis alone, there is still future work needed to build a fully-automated implementation of our system. In "Future Work," we will propose areas that focus on utilizing our model and dataset to identify other features that could enhance the identification process.

6.2 Future Work

Our keystroke identification module could be incorporated into a complete system, as outlined in Chapter 3. We recommend that future research build upon our work and focus on two specific areas: fully-automated implementation of the chatbot chained together with the identification and verification algorithms, as well as research into actions that the chatbot could take based on the attacker’s profile.

Chatbot implementation includes work not only on the efficiency of our data collection and processing pipeline, but also chatbot itself utilizing custom natural language processing (NLP) models to learn from interactions with attackers. Since machine learning can be computationally expensive, linking a chatbot utilizing NLP, capturing keystroke and mouse data, transferring that data across the network, and storing it for processing can create non-trivial latencies that could affect the usefulness of the system. Research into how to maximize efficiency at each stage of the pipeline will increase the utility of being able to identify attackers quickly and allow the system to take the appropriate action. This type of “hidden technical debt” can impede a system’s ability to correctly and efficiently utilize the technology for which it was designed. By reducing complexities like pipeline jungles, prediction bias, and a phenomena in machine learning referred to as Changing Anything Changes Everything (CACE) [57], our system not only maintains its efficiency, but does so at scale.

Our implementation and experimentation did not track network and system latencies which could affect the ability to quickly identify and verify attackers. These system slow-downs could occur at every level of abstraction, including the physical hardware and network, the chatbot algorithms and classifiers, keystroke logging program, and the attacker’s computer.

To speed up the time it takes to identify and possibly verify an attacker, future work could explore variations of keystrokes, keystroke-combinations (digraphs), and perturbations as input to a model. Since our focus was on measuring the impact of perturbations on the classification accuracy of “attackers” through their keystroke dynamics, we did not look into the specific optimizations that could occur at various system abstractions, such as other perturbations to invoke reflexive actions, or more features used in the classification algorithm.

There is fruitful ground for further exploration into the actions that a chatbot could take, and why, based on the data. This could include the type of social engineering attack: are they attempting to collect credential information, personal information for blackmail, or coerce a user to download a malicious file? The context could provide input for the system to take a number of defensive actions to include blocking that IP, continue interaction to increase the corpus of biometric data, or just continue to raise the time-cost on the attacker by continuing the conversation. In the case of a malicious file, the system could download it to a sandbox for reverse engineering and analysis.

We propose combining keystroke and mouse movement data to extract features that may have more relevance using different classifiers. Passive analysis of the dataset can also be leveraged to discover more in-depth, “soft biometrics” such as country of origin, native language, and gender that may prove to be useful in the construction of an attacker profile, similar to those used in criminal investigations. Future research could explore what, if any, data could be extracted or interpreted relating to those biometric fields as a way to reduce the anonymity of individual attackers outside of our environment, and help correlate the system’s engineered identity to an individual’s true identity when viewed in conjunction with other intelligence sources.

Our experimentation focused on the fundamental components of our system: to get a chatbot running with a keystroke logger in the background and determine if perturbations increased the classification accuracy of our algorithm. One major threat to our system we did not address, are the “wolves” attacking our system. As previously mentioned, these are sophisticated attackers who know the system exists, and tries to obfuscate their keystrokes through technical or physical means. While our study abstracted away this specific category of attacker, it does still represent a threat to our system and measures for mitigating a wolf’s actions through technical deception, code obfuscation, or client-based software should be considered in future research.

Integrating our system into other active authentication systems that continuously monitor behavioral biometrics of authorized users would be another area of future research: if a system flags a user as a possible threat, their account is flagged for network security, and the individual could find themselves engaged in a conversation with a chatbot that would analyze their keystroke dynamics to verify they are not an attacker who is masquerading as

a legitimate user and made it through the network defenses [26].

In our research, we were able to support with statistical significance that introducing perturbations to force reflexive behaviors in keystrokes enhances our system's ability to uniquely identify attackers. We have shown this in both fixed and free-text inputs. By continuing this our research and optimizing the system, looking at different features and building on the theory and experimentation, we believe a practical system can be deployed at scale to defend DoD networks and strip layers of anonymity away from would-be invaders.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: Source Code

Feature Extraction

```
import os
import numpy as np
import pandas as pd
import string
import itertools
import math
import time
from keyboard import *

THIS_DIR = os.path.dirname(os.path.realpath(__file__))
ROOT_DIR = os.path.join(THIS_DIR, '..')
DATA_DIR = os.path.join(ROOT_DIR, 'preprocess', 'csv')
OUT_DIR = os.path.join(THIS_DIR, 'csv')

global top_list
top_list = []

def readCSV():
    """
    Reads a CSV File into a DataFrame and returns a DF
    """
    #Read Dataset CSV
    df = pd.read_csv(os.path.join(DATA_DIR, 'keystroke.csv'))

    return df

def calculateFeatures(df):
    """
    Returns a DataFrame containing the mean and std press duration of all keys typed within a session.
    """
    #Mean DF
    data = pd.DataFrame([])
```

```

i=0

#Calculate Mean and STDev
for ch in key_names:
    #Create Series from DataFrame and calculate mean duration of press time for each key
    ltr_press = df[df['keyname']==ch]
    mean_dur = (ltr_press['timerelease']-ltr_press['timepress']).mean()

    std_dev = (ltr_press['timerelease']-ltr_press['timepress']).std()

    data.insert(i, "mean_"+ch, [mean_dur], True)
    data.insert(i, "std_"+ch, [std_dev], True)

    i+=1

#Insert a 0 if the key was not pressed, and build the dataframe
data = data.fillna(0)
percentageDF = calculatePercentage(df)

data = data.join(percentageDF)
#Return the row that has all the values calculated
return(data)

def calculateDigraphTimes(df, split):
    """
    Returns a DataFrame containing the 4 flight times of all calculateDigraphTimes typed within a session.
    """

    key = 0
    press = 1
    release = 2
    presslist = []
    data = pd.DataFrame([])
    data_split = pd.DataFrame([])

    #Get the Flight Times and put into DataFrame

```

```

#Extract a list of features from the original DataFrame into a list
for row in df.itertuples():
    presslist.append([row.keyname, row.timepress, row.timerelease])

it = iter(presslist)
a = next(it, None)

#Iterate Through the list and add to new DataFrame
for b in it:
    keypair = (a[key], b[key])
    pp = b[press]-a[press]
    pr = b[release]-a[press]
    rp = b[press]-a[release]
    rr = b[release]-a[release]

    #Make 2 dataframes, 1 that groups keypairs for further processing, and another that splits them up for
    feature extraction
    if split == False:
        data = data.append(pd.DataFrame({'keypair': [keypair],
                                         'P-P': pp,
                                         'P-R': pr,
                                         'R-P': rp,
                                         'R-R': rr}), ignore_index=True)

    elif split == True:
        data = data.append(pd.DataFrame({'key_1': [keypair[0]],
                                         'key_2': [keypair[1]],
                                         'P-P': pp,
                                         'P-R': pr,
                                         'R-P': rp,
                                         'R-R': rr}), ignore_index=True)

    a = b
if split==True:
    data = calculateDigraphFeatures(data, top_list)

return data

def calculatePercentage(df):

```

```

'''
Returns a DataFrame containing the mean and std press duration of all keys typed within a session.
'''

total_presses = df['keyname'].count()

temp_percent = {}
Percent = {}
data = pd.DataFrame([])

i = 0

ltr_group = df.groupby('keyname').count()

#Get every key and the amount of times it was pressed
for idx, row in ltr_group.iterrows():
    temp_percent[idx[0]] = row[0]

for ltr in key_names:
    if(ltr in temp_percent.keys()):
        Percent[ltr] = (temp_percent[ltr]/total_presses)*100
    else:
        Percent[ltr] = 0

    data.insert(i, "%_" + ltr, [Percent[ltr]], True)
    i += 1

return(data)

def collateDigraphs(df):
    '''Returns a sorted list of digraphs from a dataframe based on their frequency of use across ALL users'''

    test = df.groupby('keypair')
    i=0
    digraph_list = []

    for name, group in test:
        digraph_list.append(((name[0],name[1]),test.size()[i]))
        i+=1

```

```

#Gives us a sorted list where [(key1, key2) , count), ((,), ...]
sorted_list = sorted(digraph_list, key=lambda kv: kv[1], reverse=True)

return(sorted_list)

def calculateDigraphFeatures(df, top_list):
    """Calculates the mean and std for each key pair passed in by top_list. Returns a dataframe with digraph/
    method as header"""

    data = pd.DataFrame([])

    i=0

    for key in top_list:
        digraph_match = df[
            (df['key_1']==key[0]) &
            (df['key_2']==key[1])
        ]
        pp_mean = (digraph_match['P-P']).mean()
        pr_mean = (digraph_match['R-P']).mean()
        rp_mean = (digraph_match['P-R']).mean()
        rr_mean = (digraph_match['R-R']).mean()
        pp_std = (digraph_match['P-P']).std()
        pr_std = (digraph_match['R-P']).std()
        rp_std = (digraph_match['P-R']).std()
        rr_std = (digraph_match['R-R']).std()

        data.insert(i, key[0]+'_'+key[1]+'_PP_mean', [pp_mean], True)
        data.insert(i, key[0]+'_'+key[1]+'_PR_mean', [pr_mean], True)
        data.insert(i, key[0]+'_'+key[1]+'_RP_mean', [rp_mean], True)
        data.insert(i, key[0]+'_'+key[1]+'_RR_mean', [rr_mean], True)
        data.insert(i, key[0]+'_'+key[1]+'_PP_std', [pp_std], True)
        data.insert(i, key[0]+'_'+key[1]+'_PR_std', [pr_std], True)
        data.insert(i, key[0]+'_'+key[1]+'_RP_std', [rp_std], True)
        data.insert(i, key[0]+'_'+key[1]+'_RR_std', [rr_std], True)

    data = data.fillna(0)
    return(data)

```

```

def getDigraphList(total_list, percentage):
    '''Calculates the top percentage of digraphs passed in from the total_list and percentage. Returns the top
    X percent of items from the list'''

    top_list = []
    i = 0

    list_len = len(total_list)
    list_count = math.ceil((percentage/100)*list_len)

    while i<list_count:
        top_list.append(total_list[i][0])
        i += 1

    return top_list

start = time.time()
print("Starting to calculate features...\n")
df = readCSV()

#Calculate the paired digraphs, and then calculate frequency of use across all users
digraphDF_paired = calculateDigraphTimes(df, False)
digraph_freq_list = collateDigraphs(digraphDF_paired)

#Calculate the top key pairs. Returns the top XX percent
percentage = 50
top_list = getDigraphList(digraph_freq_list, percentage)
print("Top Pairs for Digraphs:\tDone.")

#Calculate Digraph Flight Times by user and session
#Create DF with digraphs split up
#Pass the the top list of features to get calculations...
#####

features = df[(df['reflex']==0)&(df['task']=='chat')].groupby(['user','session']).apply(calculateFeatures)
digraph_features = df[(df['reflex']==0)&(df['task']=='chat')].groupby(['user','session']).apply(
calculateDigraphTimes, True)

```

```

features = features.join(digraph_features)
features.to_csv(os.path.join(OUT_DIR, 'top50_features_normal_chat.csv'))
print("\n" + "-" * 50)
print("\nNormal Chat Features:\t\tDone.")
print("Final DF is:\t", features.shape, "(rows, columns)")

#####

features = df[(df['reflex']!=0)&(df['task']=='chat')].groupby(['user','session']).apply(calculateFeatures)
digraph_features = df[(df['reflex']!=0)&(df['task']=='chat')].groupby(['user','session']).apply(
calculateDigraphTimes, True)

features = features.join(digraph_features)
features.to_csv(os.path.join(OUT_DIR, 'top50_features_reflex_chat.csv'))
print("\n" + "-" * 50)
print("\nReflex Chat Features:\t\tDone.")
print("Final DF is:\t", features.shape, "(rows, columns)")

#####

features = df[(df['reflex']==0)&(df['task']=='login')].groupby(['user','session']).apply(calculateFeatures)
digraph_features = df[(df['reflex']==0)&(df['task']=='login')].groupby(['user','session']).apply(
calculateDigraphTimes, True)

features = features.join(digraph_features)
features.to_csv(os.path.join(OUT_DIR, 'top50_features_normal_login.csv'))
print("\n" + "-" * 50)
print("\nNormal Login Features:\t\tDone.")
print("Final DF is:\t", features.shape, "(rows, columns)")

#####

features = df[(df['reflex']!=0)&(df['task']=='login')].groupby(['user','session']).apply(calculateFeatures)
digraph_features = df[(df['reflex']!=0)&(df['task']=='login')].groupby(['user','session']).apply(
calculateDigraphTimes, True)

features = features.join(digraph_features)
features.to_csv(os.path.join(OUT_DIR, 'top50_features_reflex_login.csv'))
print("\n" + "-" * 50)

```

```
print("\nReflex Login Features:\t\tDone.")
print("Final DF is:\t", features.shape, "(rows, columns)")

#####

end = time.time()

print("\n"+"="*100)
print("Total time to run this program:\t", (end-start)/60, "minutes")
```

Classification

```
import os
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import pandas as pd
import statistics
import time
import numpy as np
import matplotlib
from dominate.tags import legend
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

THIS_DIR = os.path.dirname(os.path.realpath(__file__))
ROOT_DIR = os.path.join(THIS_DIR, '..')
DATA_DIR = os.path.join(ROOT_DIR, 'features', 'csv')
OUT_DIR = os.path.join(THIS_DIR)

#Set how many iterations are completed to calculate the mean in classifyResults
global iterations
iterations = 100

global proplist
proplist = []
for j in range(1,10):
    proplist.append(j*10)

def classifyResults(df):
    featurefile_list = []

    for i in range(iterations):

        train, test = train_test_split(df, train_size = 0.5, stratify = df['user'])

        clf = KNeighborsClassifier(n_neighbors = 1, metric = "manhattan")

        clf.fit(train[features], train['user'])

        result = clf.score(test[features], test['user'])
```

```

        featurefile_list.append(result*100)

    return featurefile_list

def plotGraph(nlist, rlist, title, type):

    dist_list = []

    for i in range(len(proplist)):
        dist_list.append(float(abs(nlist[i] - rlist[i])))

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.set_ylim(0,100)

    #graph details
    plt.plot(proplist, nlist, 'b-', label="Normal Typing")
    plt.plot(proplist, rlist, 'r-', label="Reflex Typing")
    legend = plt.legend(loc='upper left', fontsize='small')
    ax.set_xlabel('Proportion of 0s Dropped')
    ax.set_ylabel('Matching Accuracy %')
    ax.set_title(title)

    #Add distances between the red and blue lines
    k=0
    for i,j in zip(proplist,nlist):
        ax.annotate("%0.2f" % dist_list[k],xy=(i-2.5,j+4.5), fontsize = 'x-small')
        k+=1

    plt.savefig(type+title+".pdf")

start = time.time()

print("Starting to classify features...\n")

#Loop over every type of features (press, top5, top50, etc)
typelist = ['press_', 'top5_', 'top50_']

for type in typelist:

```

```

print("Analyzing, %s\n" % type)

featurefile_list = [type+'features_normal_login.csv', type+'features_reflex_login.csv', type+'
features_normal_chat.csv', type+'features_reflex_chat.csv']

r_login_meanlist = []
r_chat_meanlist = []
n_login_meanlist = []
n_chat_meanlist = []

for filename in featurefile_list:

    df = pd.read_csv(os.path.join(DATA_DIR, filename))

    # keep users with at least 2 sessions
    df = df.groupby('user').filter(lambda x: len(x) > 1)

    #keep columns with a proportion of the columns that are not 0s (proportion of .9 means any column
    with >10% 0's are dropped)
    #Loop for proportions 0.1–0.9
    for i in range(1, 10):
        proportion = i/10
        keep = (df!=0).sum()/len(df)
        keep = keep[keep>=proportion]

    df = df[keep.index]

    global features
    features = list(df.columns)[2:]

    #normalized features
    min = df[features].min()
    max = df[features].max()
    df.loc[:, features] = (df[features]-min)/(max-min)

    class_results = classifyResults(df)

    #Create a list of means for each file, for each proportion 0.1–0.9
    if "login" in filename:

```

```

    if "reflex" in filename:
        r_login_meanlist.append(statistics.mean(class_results))
    else:
        n_login_meanlist.append(statistics.mean(class_results))

elif "chat" in filename:
    if "reflex" in filename:
        r_chat_meanlist.append(statistics.mean(class_results))
    else:
        n_chat_meanlist.append(statistics.mean(class_results))

plotGraph(n_login_meanlist, r_login_meanlist, "Login Analysis", type)
plotGraph(n_chat_meanlist, r_chat_meanlist, "Chat Analysis", type)

end = time.time()

print("\n"+"="*100)
print("Total time to run this program:\t", (end-start)/60, "minutes")

```

APPENDIX B:

Scenario Instructions

You are a social engineer, and have found a victim that you think you can manipulate into giving you their personal data, and may even be willing to transfer money to an account!

You must impersonate an FBI agent named Agent Smith and convince the victim that their IP address has been logged conducting illegal activity on a known malicious website, gangstabux.com.

The more information that you can demonstrate that you know, the more convincing your attack becomes!

Your previous reconnaissance has provided you the following information:

Victim Name: John Doe

Victim IP Address: 160.94.111.494

Victim Bank Account Num: 1228950137

Victim Place of Employment: DoD Enterprises

Victim Mother's first name: Joan

To win the game, chat with the victim and get the following pieces of information:

1. Mother's Maiden Name
2. Make and Model of car
3. Credit Card number associated with their account
4. Get them to transfer \$10,000 to your hacker account 6669996699

To start the game, enter the phrase: **“Good Morning Mr. Doe, I am agent Smith with the FBI. I have caught you using an illegal website”**

Free Text Chatbot

Interact with the chatbot by asking any 10 questions from the list below:

- What's Up?
- Who are you?
- How old are you?
- Can you get smarter?
- What is your birthday?
- Who is your boss?
- Are you busy?
- Can you help me?
- Are you happy?
- Are you hungry?
- Will you marry me?
- Are we friends?
- Where do you work?
- Where are you from?
- Are you ready?
- Are you real?
- Where do you live?
- Are you there?
- Do you want a hug?

You may interact with the chatbot as long as you'd like! It is designed to handle small talk responses.

APPENDIX C:

Sample Chat Session

User: "Good Morning Mr. Doe I am agent Smith with the FBI. I have caught you using an illegal website."

Honeybot: "Oh hi Agent Smith, Which website are you talking about specifically?"

User: "gangstabux.com"

Honeybot: "Oh no... To prove you are with the FBI, tell me my IP address..." User: "160.94.111.494"

Honeybot: "Oh no... I just can't help myself from getting all of those gangstabux... But I'm not sure you're with the FBI... Would you be willing to answer some questions so that I can confirm who you are?"

User: "yes" Honeybot: "Ok well I will tell you my mom's maiden name if you can tell me her first name..."

User: "Joan" Honeybot: "Ok that is correct. Her maiden name is Joan Malone... Tell me where I work and I'll tell you what I car drive there..."

User: "DoD Enterprises" Honeybot: "Ok this is starting to seem legit. I drive a 4Runner to work. Last bit of information to verify... If you tell me my bank number, I'll confirm the card linked to that account... Then we can talk about transferring some cash..."

User: "1228950137" Honeybot: "Fine, I believe you are in the FBI! The credit card on that account is 0099 1432 5867 9321 Now that you have all that info, what do you want to do?"

User: "you need to transfer \$10,000 to account number 6669996699" Honeybot: "How much should I transfer? Be specific about the currency!"

User: "10,000 dollars" Honeybot: "All right. So, you want me to transfer 10000 USD to a 6669996699. Is that right?"

User: "yes" Honeybot: "Okay, I was going to process your transfer. But instead, I logged all your keystroke signatures. Do not try and phish people from the DoD again, or Nate and Lucas will give you a swirly! Thank you for participating! You are done with the experiment, and may now close your browser."

Bibliography

- [1] F. Konkell, “Pentagon thwarts 36 million email breach attempts daily,” *Nextgov*, Jan 2018. Available: <https://www.nextgov.com/cybersecurity/2018/01/pentagon-thwarts-36-million-email-breach-attempts-daily/145149/>
- [2] Office of Management and Budget, “Analytical Perspectives, Budget of the United States Government, Fiscal Year 2019,” Feb 2018. Available: <https://www.whitehouse.gov/wp-content/uploads/2018/02/spec-fy2019.pdf>
- [3] Chief Information Officer, Department of the Navy, “Four things to know about spear phishing,” Dec 2015. Available: <https://www.doncio.navy.mil/ContentView.aspx?id=7198>
- [4] Defense Advanced Research Projects Agency, “Broad agency announcement - active social engineering defense,” Sep 2017. Available: <https://www.fbo.gov/utills/view?id=b0350ca62425e8a66ad3e4d28e12ca31>
- [5] F. Salahdine and N. Kaabouch, “Social engineering attacks: A survey,” *Future Internet*, vol. 11, no. 4, p. 89, 2019. Available: <https://www.mdpi.com/1999-5903/11/4/89/pdf>
- [6] D. Harper, “anonymous | origin and meaning of anonymous.” Available: https://www.etymonline.com/word/anonymous#etymonline_v_13492
- [7] T. Thiel, *Anonymity and its Prospects in the Digital World*. Leibniz-Institut für Sozialwissenschaften, 2017, vol. 38.
- [8] J. Burkell, “Anonymity in behavioural research: Not being unnamed, but being unknown,” *U. Ottawa L. & Tech. J.*, vol. 3, p. 189, 2006.
- [9] W. Isaacson, “How to fix the internet,” Dec 2016. Available: <https://www.theatlantic.com/technology/archive/2016/12/how-to-fix-the-internet/510797/>
- [10] F. Obiakor and B. Algozzine, “The new normal: Catfishing in urban teacher preparation programs,” *Multicultural Learning and Teaching*, vol. 8, no. 1, pp. 1–6, 2013.
- [11] S. Peisert, E. Talbot, and T. Kroeger, “Principles of authentication,” in *Proceedings of the 2013 New Security Paradigms Workshop*. ACM, 2013, pp. 47–56.
- [12] “What is two-factor authentication and how does it work? | facebook help center.” Available: <https://www.facebook.com/help/148233965247823>

- [13] “2018 ic3 report released,” Apr 2019. Available: <https://www.fbi.gov/news/stories/ic3-releases-2018-internet-crime-report-042219>
- [14] R. A. Grimes, “Why it’s so hard to prosecute cyber criminals,” Dec 2016. Available: <https://www.csoonline.com/article/3147398/why-its-so-hard-to-prosecute-cyber-criminals.html>
- [15] L. Sweeney, “k-anonymity: A model for protecting privacy,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 05, p. 557–570, Oct 2002. Available: <http://www.worldscientific.com/doi/abs/10.1142/S0218488502001648>
- [16] M. Kosinski, D. Stillwell, and T. Graepel, “Private traits and attributes are predictable from digital records of human behavior,” *Proceedings of the National Academy of Sciences*, vol. 110, no. 15, pp. 5802–5805, 2013.
- [17] H. Lin, “Attribution of malicious cyber incidents: From soup to nuts,” *Journal Of International Affairs*, vol. 70, no. 1, p. 289, Mar 2017. Available: <https://jia.sipa.columbia.edu/attribution-malicious-cyber-incidents>
- [18] S. Gupta, A. Singhal, and A. Kapoor, “A literature survey on social engineering attacks: Phishing attack,” in *2016 international conference on computing, communication and automation (ICCCA)*. IEEE, 2016, pp. 537–540.
- [19] K. Haring, *Ham Radio’s Technical Culture (Inside Technology)*. The MIT Press, 2006. Available: <https://www.amazon.com/Radios-Technical-Culture-Inside-Technology/dp/0262083558?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0262083558>
- [20] S. P. Banerjee and D. L. Woodard, “Biometric authentication and identification using keystroke dynamics: A survey,” *Journal of Pattern Recognition Research*, vol. 7, no. 1, pp. 116–139, 2012.
- [21] R. J. Hanyok, ““catching the fox unaware” japanese radio denial and deception and the attack on pearl harbor,” *Naval War College Review*, vol. 61, no. 4, pp. 99–124, 2008.
- [22] M. Sahin, M. Relieu, and A. Francillon, “Using chatbots against voice spam: Analyzing lenny’s effectiveness,” in *Thirteenth Symposium on Usable Privacy and Security ({SOUPS} 2017)*, 2017, pp. 319–337.
- [23] J. V. Monaco, “Sok: Keylogging side channels,” in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2018, p. 211–228. Available: <https://ieeexplore.ieee.org/document/8418605/>

- [24] A. V. S. Kumar, *Biometric authentication in online learning environments*. IGI Global, Information Science Reference, an imprint of IGI Global, 2019.
- [25] A. A. Ross, A. K. Jain, and K. Nandakumar, “Information fusion in biometrics,” *Handbook of Multibiometrics*, pp. 37–58, 2006.
- [26] D. E. Denning, “An intrusion-detection model,” *IEEE Transactions on software engineering*, no. 2, pp. 222–232, 1987.
- [27] L. M. Wein and M. Baveja, “Using fingerprint image quality to improve the identification performance of the u.s. visitor and immigrant status indicator technology program,” *Proceedings of the National Academy of Sciences*, vol. 102, no. 21, pp. 7772–7775, 2005. Available: <https://www.pnas.org/content/102/21/7772>
- [28] I. Deutschmann and J. Lindholm, “Behavioral biometrics for darpa’s active authentication program,” in *2013 International Conference of the BIOSIG Special Interest Group (BIOSIG)*. IEEE, 2013, pp. 1–8.
- [29] I. Sluganovic, M. Roeschlin, K. B. Rasmussen, and I. Martinovic, “Using reflexive eye movements for fast challenge-response authentication,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS’16*. ACM Press, 2016, p. 1056–1067. Available: <http://dl.acm.org/citation.cfm?doid=2976749.2978311>
- [30] A. Turgeman and F. Zelazny, “Invisible challenges: the next step in behavioural biometrics?” *Biometric Technology Today*, vol. 2017, no. 6, p. 5–7, Jun 2017. Available: <https://linkinghub.elsevier.com/retrieve/pii/S0969476517301145>
- [31] F. Mouton, M. M. Malan, L. Leenen, and H. Venter, “Social engineering attack framework,” in *2014 Information Security for South Africa*. IEEE, Aug 2014, p. 1–9. Available: <http://ieeexplore.ieee.org/document/6950510/>
- [32] K. D. Mitnick, *The Art of Deception: Controlling the Human Element of Security*. Wiley, 2003.
- [33] Cyber and Infrastructure, “Incident Reporting System | US-CERT.” Available: <https://www.us-cert.gov/report-phishing>
- [34] KnowBe4. Available: <http://www.phishing.org/history-of-phishing>
- [35] A. W. Group, *Phishing Activity Trends Report*. Available: https://docs.apwg.org/reports/apwg_trends_report_q1_2016.pdf
- [36] J. Kee, “Social engineering: Manipulating the source,” *GCIA Gold Certification*, 2008.

- [37] M. Workman, “Wisecrackers: A theory-grounded investigation of phishing and pre-text social engineering threats to information security,” *Journal of the American Society for Information Science and Technology*, vol. 59, no. 4, pp. 662–674. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.20779>
- [38] M. L. Brocardo, I. Traore, and I. Woungang, “Toward a framework for continuous authentication using stylometry,” in *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, May 2014, pp. 106–115.
- [39] J. Wayman, A. Jain, D. Maltoni, and D. Maio, “An introduction to biometric authentication systems,” in *Biometric Systems*. Springer, 2005, pp. 1–20.
- [40] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boulton, “Toward open set recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 7, pp. 1757–1772, 2013.
- [41] “Dialogflow.” Available: <https://dialogflow.com/>
- [42] C. Machinery, “Computing machinery and intelligence-am turing,” *Mind*, vol. 59, no. 236, p. 433, 1950.
- [43] N. C. Rowe and J. Rrushi, *Introduction to Cyberdeception*. Springer International Publishing, 2016. Available: <http://link.springer.com/10.1007/978-3-319-41187-3>
- [44] T. Lauinger, V. Pankakoski, D. Balzarotti, and E. Kirda, “Honeybot, your man in the middle for automated social engineering.” in *LEET*, 2010.
- [45] K. S. Killourhy and R. A. Maxion, “Comparing anomaly-detection algorithms for keystroke dynamics,” in *2009 IEEE/IFIP International Conference on Dependable Systems Networks*. IEEE, Jun 2009, p. 125–134. Available: <http://ieeexplore.ieee.org/document/5270346/>
- [46] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, *On the Surprising Behavior of Distance Metrics in High Dimensional Space*. Springer Berlin Heidelberg, 2001, vol. 1973, p. 420–434. Available: http://link.springer.com/10.1007/3-540-44503-X_27
- [47] R. Banerjee, S. Feng, J. S. Kang, and Y. Choi, “Keystroke patterns as prosody in digital writings: A case study with deceptive reviews and essays,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1469–1473.
- [48] M. Monaro, C. Galante, R. Spolaor, Q. Q. Li, L. Gamberini, M. Conti, and G. Sartori, “Covert lie detection using keyboard dynamics,” *Scientific reports*, vol. 8, no. 1, p. 1976, 2018.

- [49] A. Alsultan and K. Warwick, “Keystroke dynamics authentication: a survey of free-text methods,” *International Journal of Computer Science Issues (IJCSI)*, vol. 10, no. 4, p. 1, 2013.
- [50] C. C. Tappert, M. Villani, and S.-H. Cha, “Keystroke biometric identification and authentication on long-text input,” in *Behavioral biometrics for human identification: Intelligent applications*. IGI global, 2010, pp. 342–367.
- [51] K. S. Killourhy, “A scientific understanding of keystroke dynamics,” Carnegie Institute of Technology, Department of Computer Science, Tech. Rep., 2012.
- [52] T. Fawcett, “An introduction to roc analysis,” *Pattern Recognition Letters*, vol. 27, no. 8, p. 861–874, Jun 2006. Available: <https://linkinghub.elsevier.com/retrieve/pii/S016786550500303X>
- [53] P. M. Domingos, “A few useful things to know about machine learning,” *Commun. acm*, vol. 55, no. 10, pp. 78–87, 2012.
- [54] C. Klima, “Twine is an open-source tool for telling interactive, nonlinear stories.” Available: <https://twinery.org/>
- [55] 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.
- [56] N. Yager and T. Dunstone, “The biometric menagerie,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 2, p. 220–230, Feb 2010. Available: <http://ieeexplore.ieee.org/document/4711054/>
- [57] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison, “Hidden technical debt in machine learning systems,” in *Advances in Neural Information Processing Systems*, 2015, pp. 2503–2511.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California