

Spring 2016

Determining Unique Agents by Evaluating Web Form Interaction

Ben Cooley

Follow this and additional works at: <https://digitalcommons.georgiasouthern.edu/etd>



Part of the [Digital Communications and Networking Commons](#), and the [Other Computer Engineering Commons](#)

Recommended Citation

Cooley, Ben. DETERMINING UNIQUE AGENTS BY EVALUATING WEB FORM INTERACTION

This thesis (open access) is brought to you for free and open access by the Graduate Studies, Jack N. Averitt College of at Digital Commons@Georgia Southern. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Digital Commons@Georgia Southern. For more information, please contact digitalcommons@georgiasouthern.edu.

DETERMINING UNIQUE AGENTS BY EVALUATING WEB FORM INTERACTION

by

BENJAMIN COOLEY

(Under the Direction of James Harris)

ABSTRACT

Because of the inherent risks in today's online activities, it becomes imperative to identify a malicious user masquerading as someone else. Incorporating biometric analysis enhances the confidence of authenticating valid users over the Internet while providing additional layers of security with no hindrance to the end user. Through the analysis of traffic patterns and HTTP Header analysis, the detection and early refusal of robot agents plays a great role in reducing fraudulent login attempts.

INDEX WORDS: Web, Security, Biometrics, Computer science, Robot, Machine learning, Keystroke dynamics, Network, Website, Automation

DETERMINING UNIQUE AGENTS BY EVALUATING WEB FORM INTERACTION

by

BENJAMIN COOLEY

B.S. Augusta State University, 2010

A Thesis submitted to the Graduate Faculty of Georgia Southern University in Partial fulfillment

of the requirements for the degree

MASTER OF COMPUTER SCIENCE

STATESBORO, GEORGIA

© 2016

BENJAMIN COOLEY

All Rights Reserved

DETERMINING UNIQUE AGENTS BY EVALUATING WEB FORM INTERACTION

by

BENJAMIN COOLEY

Major Professor: James Harris

Committee: Lixin Li
Youming Li

Electronic Version Approved

May 2016

ACKNOWLEDGEMENTS

I wish to thank my advisor James Harris for his interest and support of my work. For the help and support of my father Stuart Cooley and mother Heather Tew, I give my utmost gratitude. Additionally, I would like to thank my uncle Steve Campbell and his wife Rachael, Katie Hotchkiss, Brian Mykietka, and Josh Garwood for their help in proofing and their support throughout the completion of this thesis.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	2
LIST OF FIGURES	5
LIST OF TABLES.....	6
LIST OF EQUATIONS	7
CHAPTERS	
1 INTRODUCTION	8
2 LITERATURE REVIEW	9
WEB ROBOTS	9
DETERMINING HUMANNESS	12
BEHAVIORS OF AGENTS.....	13
DETERMINING UNIQUENESS BETWEEN HUMAN AGENTS	21
3 PURPOSE	28
4 METHODOLOGY	29
WEB SITE TO TRACK INPUTS	29
WEB ROBOT TRAFFIC GENERATION	34
COMPARISON METHODS.....	37
5 ANALYSIS OF RESULTS	42
TRAFFIC ANALYSIS	42
TIMING VECTOR ANALYSIS.....	49
SUMMARY	54
6 FUTURE WORK.....	56
7 CONCLUSION.....	57
REFERENCES	58
APPENDICES	61
A DISSIMILARITY ALGORITHM.....	61
B DATA CAPTURE LAYOUT TEMPLATE MARKUP	63
C LOGIN PAGE MARKUP.....	64
D NON-INTRUSIVE DATA CAPTURING JAVASCRIPT	67
E INTERACTION FORM CONTROLLER.....	69

F OBJECT MODEL ENTITIES.....	72
G BACKPROPAGATION CLASSIFIER.....	75
H SUPPORT VECTOR MACHINE CLASSIFIER	76
I DECISION TREE CLASSIFIER	77
J DISSIMILARITY COMPARISON CLASSIFIER	78
K COMMON EXPERIMENT VARIABLES AND FUNCTIONS	80
L BACKPROPAGATION TEST	83
M SUPPORT VECTOR MACHINE TESTS	84
N DECISION TREE TESTS	86
O DISSIMILARITY COMPARISON TESTS.....	89
P DATABASE SCRIPT	93

LIST OF FIGURES

Figure 1: Percentage of Robots by HTTP Requests	12
Figure 2: Visual representation of typed characters	25
Figure 3: Pattern of the transformed data that has been captured.	26
Figure 4: Image of the data capture application	31
Figure 5: Data Capture Database Schema	33
Figure 6: Key Hold Times with Fake Login	37
Figure 7: Key Press Times with Fake Login	37

LIST OF TABLES

Table 1: Human Headers during a session on the Data Capture Web Site	15
Table 2: "MJ12bot" Robot headers during a session on the Data Capture Web Site	16
Table 3: Human click path	16
Table 4: Password times between keystrokes	24
Table 5: Differences in time based on when a key is pressed and the previous key released	25
Table 6: Sample Request Headers	30
Table 7: Sample request information	30
Table 8: Sample input capture (Keyboard)	30
Table 9: Sample input capture (Mouse)	31
Table 10: Valid Login Attempts	36
Table 11: Additional Known Attempts	36
Table 12: A sample of randomly generated login attempts	36
Table 13: User-Agent Headers	43
Table 14: Resources Accessed by Robots	43
Table 15: Fast Resource Access Headers	44
Table 16: User submitted timing vector for Email Field	45
Table 17: Selenium submitted timing vector for Email Field	47
Table 18: Sahi submitted timing vector for Email Field	48
Table 19: Results of Support Vector Machine Training	51
Table 20: Decision Tree Results (values are per 10,000)	52

LIST OF EQUATIONS

Equation 1: Dissimilarity Equation	41
Equation 2: Valid login Function using Dissimilarity method	41

CHAPTER 1

INTRODUCTION

Identifying the humanness of a particular user of a Web page is crucial. Web robots consume valuable bandwidth and perform automated actions, such as repeated attempts to log in to user accounts. Studies suggest the Web robot to human ratio is as high as 10:1 regarding the number of sessions opened on a Web server; and as high as 4:1 concerning bandwidth consumed on a particular Web server [1].

This thesis explores the distinctions between humans and robots and the unique interactions between humans and Web forms. The exploration defines Web robots and their uses, the behaviors of different agents, traffic analysis between human and non-human robots, and various identification mechanisms. This research also evaluates the differences between humans and robots, and differences between multiple human actors.

CHAPTER 2

LITERATURE REVIEW

2.1 Web Robots

There are two classifications of Web robots: non-malicious and malicious. Most current research identifies Web robots based on Web-browsing patterns and resource requests. There has been little research concerning how robots interact with Web forms. A later section of this thesis analyzes collected data that details the interactions between Web robots and Web forms.

2.1.1 Non-Malicious Robots

Non-malicious Web robots normally take the form of a Web crawler; a robot that traverses Web pages searching for pieces of information. There are several sub-classifications of Web crawlers. These classifications include indexer, verifier, harvester, scraper, RSS crawler, experimental, and unknown types [2].

2.1.1.1 Indexer

Indexer robots create a map of the Internet. These robots follow objects defined in a Website's robots.txt and links to create the map.

2.1.1.2 Verifier

Verifier robots verify or validate a set of functionality or rules of Web pages. Several examples of verification are HTML verification, broken link detection, accessibility compliance, and quality assurance testing.

2.1.1.3 Harvester

Harvester robots do what their name implies; harvest content on Web pages. These types of crawlers request resources such as images and documents.

2.1.1.4 Scraper

Scraper robots save requested HTML files. Malicious and non-malicious users purpose the saved data for their own means. Malicious purposes include the gathering of email addresses for an attempt to break into a Web page. Non-malicious purposes include gathering data for language analysis and deep learning.

2.1.1.5 RSS Crawler

RSS crawlers keep up-to-date information from various RSS feeds. This crawler is often on a schedule and refreshes the RSS feeds periodically.

2.1.1.6 Experimental

Experimental crawlers use a set of combined techniques to map, verify, or harvest data on the Web. These are naturally used for research or experimental purposes, hence the name Experimental crawler.

2.1.1.7 Unknown

Other Web crawlers exist, however the function of these are outside of the scope of this thesis.

2.1.2 Malicious Robots

Various robot types exhibit malicious intent. Malicious users automate tasks that take significant time for a human to perform, increasing the efficacy of an attack. Some examples of malicious robots include those that destroy or disrupt an online service.

2.1.2.1 Destructive

Destructive robots can cause harm to a Web server by performing operations such as filling up the hard disk, registering many invalid users to take over the user-base, or exploit mechanisms of a business for financial gain.

2.1.2.2 Disruptive

Some disruptive robots prevent Web servers from serving pages to legitimate users. A Denial of Service attack prevents access to Web content by causing a Web server to execute too many I/O requests, or floods a network with requests.

Spam robots are another form of disruptive robot. These robots generate information on an online Web application, or send unwanted emails to an end user. The percentage of traffic that spam robots generate has decreased over the previous years [3].

2.1.3 Distribution of Robots

Web robots account for most Web traffic, accounting for up to 61.5% of all Web traffic on the Internet [3]. While Web crawlers are accountable for 20% to 31% of global Web traffic, independent Web sites may see 50% or more of their own traffic generated by Web crawlers [3] [4] [5]. Malicious robots account for as much as 5% of Internet traffic.

Figure 1 shows the distribution of traffic by robots. There are few resources that specify the distribution of robot categories, however, two papers by Doren provide details on potential

distributions of Web robots [2] [6]. Doren and his colleagues at the University of Connecticut collected the data between 2007 and 2009.

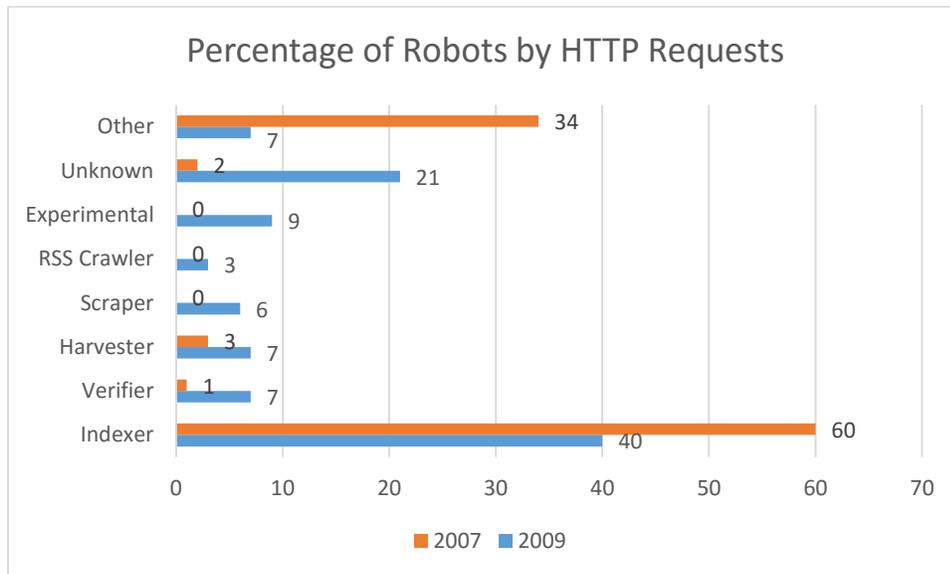


Figure 1 - Percentage of Robots by HTTP Requests

2.2 Determining Humanness

By assessing log files and form interactions, it is possible to identify non-human requests. Current research suggests that various measures of internet-browsing patterns can identify robots. For example, indexers often visit a robots.txt file on the Web server, where humans seldom visit this resource. It is possible to determine potential robots by weighing the time between requests and the time a robot interacts with a Web page. Evaluation of request headers and other common Web-browser behaviors to gauge whether a user is a Web robot is promising. Several attributes used as heuristic detection markers [7] are:

- 1) Did the user request the “robots.txt” file?
- 2) What is the time between requests?
- 3) What is the variance between request times?
- 4) Are there 404 response codes?

- 5) Are there 400 response codes?
- 6) Does the request have a bot user-agent header?
- 7) Does the request have a user-agent header that does not match a common browser?
- 8) Do requests have the same or similar referrer for each page visited?
- 9) Does the URL resolution has high number of unresolved requests?
- 10) Is the HTTP version number included?

Comparing the requesting IP address to a list of IP addresses known to generate traffic via autonomous agents can identify robot agents. For example, Google states their robots originate from a DNS entry whose domain is googlebot.com or google.com. The address returned by doing a hostname lookup for a potential robot is a value whose domain was as specified above. During this writing, IP Address: 66.249.66.1 gives a hostname of “crawl-66-249-66-1.googlebot.com” and is defined as a google crawler.

When machine learning algorithms are trained using the patterns of known Web robots, autonomous agents can be identified. Classification techniques such as neural networks, support vector machines, and decision trees are examples of these algorithms. This thesis later introduces these algorithms and the results of these classifiers.

2.3 Behaviors of Agents

2.3.1 Human Behavior

Several mechanisms identify human behavior on the Web. These mechanisms assess Web page access time, session costs, session length, and link following behavior [1] [7] [8] [9].

Humans seldom access multiple different Web pages within a single second. Human browsing requires visual or auditory processing which results in much slower request times than automated

agents. It is even rarer for humans to access different Web pages within a single half second. Web browsers used by humans load additional resources such as images, script files, and other multimedia content.

By requesting multimedia and script files, a typical human session costs more network bandwidth and memory than a robot agent does. Robot agents tend to forgo these additional resources, as they do not provide the resources that most robots require.

Session length, the number of Web page requests made by an agent, is an important criterion when determining if an agent is human. Humans typically request fewer Web pages than robots in any given session [1]. Humans regularly request fewer Web pages over a session because humans tend to follow links of interest, instead of trying to index, or harvest data as robots do. Because humans also process information more slowly than robots, the session times are longer for humans than robots. Humans represent fewer accesses per time interval than robots, which means that the number of requests per time interval is a good measure of humanness.

The way humans follow links is another criterion to consider. Human behavior regularly follows a depth-first approach in following links, whereas robots usually perform a breadth first search. Robots tend to queue links located on a Web page, which argues for why robots perform breadth-first searches. Moreover, robots follow hidden links more often than do humans.

2.3.1.1 Predictability and Standard Behavior

Humans demonstrate several predictable behaviors when it comes to interacting with Web pages. This section explores HTTP headers, network behavior patterns, and form interaction behaviors as they pertain to human agents.

HTTP headers that come from human agents are predictable, meaning they tend not to differentiate in the number or type of headers sent. Wild HTTP headers were captured using a data-capturing website. The methodology of this research explores the way that this Web site captures data and discusses how the data is analyzed. Data collected by this Web site shows that many Web crawlers do not submit a cookie header if prompted to send one back to the server; most human agents return this header. The tables below show this behavior for human and robot agents (cells are shaded to aid reading). Only the human responds with the required cookie. The Data Capture Web Site sends a response header “Set-Cookie”, which only human agents seem to respect when performing additional requests during a session. See Table 1 below.

IP Address	URL	Header Name	Date and Time
76.26.211.37	http://www.trytologin.com/	X-REWRITE-URL	2015-10-25 19:39:40.310
76.26.211.37	http://www.trytologin.com/	Upgrade-Insecure-Requests	2015-10-25 19:39:40.310
76.26.211.37	http://www.trytologin.com/	User-Agent	2015-10-25 19:39:40.310
76.26.211.37	http://www.trytologin.com/	Host	2015-10-25 19:39:40.310
76.26.211.37	http://www.trytologin.com/	Accept-Language	2015-10-25 19:39:40.310
76.26.211.37	http://www.trytologin.com/	Accept-Encoding	2015-10-25 19:39:40.310
76.26.211.37	http://www.trytologin.com/	Accept	2015-10-25 19:39:40.310
76.26.211.37	http://www.trytologin.com/	Connection	2015-10-25 19:39:40.310
76.26.211.37	http://www.trytologin.com/	X-REWRITE-URL	2015-10-25 19:39:53.640
76.26.211.37	http://www.trytologin.com/	Upgrade-Insecure-Requests	2015-10-25 19:39:53.640
76.26.211.37	http://www.trytologin.com/	User-Agent	2015-10-25 19:39:53.640
76.26.211.37	http://www.trytologin.com/	Referrer	2015-10-25 19:39:53.640
76.26.211.37	http://www.trytologin.com/	Host	2015-10-25 19:39:53.640
76.26.211.37	http://www.trytologin.com/	Cookie	2015-10-25 19:39:53.640
76.26.211.37	http://www.trytologin.com/	Accept-Language	2015-10-25 19:39:53.640
76.26.211.37	http://www.trytologin.com/	Accept-Encoding	2015-10-25 19:39:53.640
76.26.211.37	http://www.trytologin.com/	Accept	2015-10-25 19:39:53.640

Table 1 - Human Headers during a session on the Data Capture Web Site

Table 1 contrasts with Table 2 in that Table 1 displays the human returning a header, whereas Table 2 exhibits robot behavior that does not resend the header.

IP Address	Url	Name	Date and Time
91.121.169.194	http://www.trytologin.com/	Connection	2015-09-29 20:00:43.097
91.121.169.194	http://www.trytologin.com/	Accept	2015-09-29 20:00:43.097
91.121.169.194	http://www.trytologin.com/	Accept-Language	2015-09-29 20:00:43.097
91.121.169.194	http://www.trytologin.com/	Host	2015-09-29 20:00:43.097
91.121.169.194	http://www.trytologin.com/	User-Agent	2015-09-29 20:00:43.097
91.121.169.194	http://www.trytologin.com/	X-REWRITE-URL	2015-09-29 20:00:43.097
91.121.169.194	http://www.trytologin.com/	Connection	2015-09-29 20:00:46.537
91.121.169.194	http://www.trytologin.com/	Accept	2015-09-29 20:00:46.537
91.121.169.194	http://www.trytologin.com/	Accept-Encoding	2015-09-29 20:00:46.537
91.121.169.194	http://www.trytologin.com/	Accept-Language	2015-09-29 20:00:46.537
91.121.169.194	http://www.trytologin.com/	Host	2015-09-29 20:00:46.537
91.121.169.194	http://www.trytologin.com/	User-Agent	2015-09-29 20:00:46.537
91.121.169.194	http://www.trytologin.com/	X-REWRITE-URL	2015-09-29 20:00:46.537

Table 2 - "MJ12bot" Robot headers during a session on the Data Capture Web Site

Humans tend to request HTTP resources from the main Web page and subsequent visited pages.

Web page markup exposes resources meant for observation. Standard Web browser requests submit referrer headers, which create a logical click path as the user navigates a Web site. This is due to the agent following links readily available on the web page and by using normal browser controls, such as forward and back. The following table contains a subset of a captured human click path.

IP Address	Referrer	URL	Request Type	Date and Time
202.67.36.228	/	/Home/Contact	GET	2015-09-11 23:06:44.863
202.67.36.228	/Home/Contact	/	GET	2015-09-11 23:06:59.940
202.67.36.228	/	/	POST	2015-09-11 23:07:53.200
202.67.36.228	/	/	POST	2015-09-11 23:08:02.497
202.67.36.228	/	/Home/About	GET	2015-09-11 23:08:15.857
202.67.36.228	/	/Home/About	GET	2015-09-11 23:08:16.247
202.67.36.228	/Home/About	/	GET	2015-09-11 23:08:21.093

Table 3 - Human click path

Patterns have identifiable differences in human and robotic network behavior [6] [10] [11]. For example, when human agents access Web pages they tend to create a single HTTP request. If the server load is too high to handle requests in a timely manner, the user often does not continue to

make HTTP requests. Conversely, a robot does not care if response times are long and continues to attempt to access the resource from the Web server [10].

2.3.2 Robot behavior

2.3.2.1 Access Materials not generally seen by humans

Some crawlers attempt to access any resource available to them [11]. Resource availability is assessed through different mechanisms such as if the resource appearing with a direct link from the Web page, from another resource (such as robots.txt), through trying to access resources from a search engine or other site's links, or by randomly trying to access resources known to exist (login pages, configuration files, etc.) [6] [11].

Based on the logs from the Data Capture Web Site, it was determined that a robot attempts to access denied resources as defined in the robots.txt. Only one agent accessed denied resources; it was likely trying to enumerate resources, which could provide administrative access. Most ethical bots, such as the Google bot and Bing bot, did not attempt to access the resources marked as deny.

2.3.2.2 Resource Access Times vs Interaction Time

As presented earlier, Web page access times allow for the identification of robots. Robots tend to access several Web pages very quickly, usually within a fraction of a second. Human users, while they are able to access resources at such speeds, do not usually do so. Human browsing time tends to be over one-half second per page request, whereas robots would have made numerous requests for resources in this time [1].

Another item of interest concerning resource access times is the identification of robots via their access of Image files compared to HTML files. Most crawlers, excluding harvesters, tend not to grab images. Instead, these robots often prefer items of a smaller size such as text files and Web

pages [6]. Normal human browsing requests every image on the page, if images exist. Robots tend to exclude these images. AlNoamany, et al., consider the ratio of 10 Web page requests to < 1 image request to be robot-like, and the inverse of human behavior [1].

2.3.3 Current Measures to Prevent Invalid Agents

This section covers currently practiced methods that prevent invalid, or unauthorized, agents from accessing systems or resources. These methods include attempts to prevent robot access and attempts to prevent unauthorized accesses to resources and materials.

2.3.3.1 Primary authentication and verification methods

Most robot prevention takes place when an agent attempts to submit a Web form. The most common of these is CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart). Other methods for authentication and user verification include form authentication, Kerberos authentication, and external authentication methods.

2.3.3.1.1 CAPTCHA

CAPTCHA is the most widely used tool to prevent automatic entry of Web forms. This section provides a brief overview of CAPTCHA; however, the implementation details are beyond the scope of this thesis. The basis of its use is simple: create a scenario, which is difficult for robots to process correctly, but effortless for humans to do. While initially just asking the user to type the text in a distorted image, CAPTCHA has gone through several different iterations throughout the years. One subsequent iteration of CAPTCHA is reCAPTCHA. To improve on CAPTCHA, reCAPTCHA allows for additional character distortions, rotations, and spacing. Unfortunately, as robots have become more capable of reading text on an image, humans continue to struggle with reading the text. Ease-of-use methods for CAPTCHA and reCAPTCHA were explored, as well as an attempt to make use of the human generated data. Ease-of-use methods, such as asking the user

to click an image or widget, or choose a set of images that match a description are currently in use. CAPTCHA and reCAPTCHA use the data gathered from the initial methods to digitize text, aid in machine learning, and describe various images.

There are currently robots able to read most CAPTCHA images with a high success rate, some of which are commercially available. As the prevalence of robots increases, the demand for robot detection changes dramatically. Robots have bypassed even the new methods of CAPTCHA.

Next, several common authentication methods that validate a user are explored.

2.3.3.1.2 Form Authentication

Form authentication, and Internet basic authentication, requires the user to enter a username and password combination before allowing access to resources of the Web page.

The idea of password authentication has been in existence for millennia and is the most common form of user authentication. Security measures are beginning to include secondary-authentication methods. Secondary authentication helps prevent unauthorized access from different entities, when malicious users compromise usernames and passwords.

In the research conducted for this thesis, form authentication was the authentication measure utilized to authenticate users.

2.3.3.1.3 Kerberos Authentication

Kerberos is another popular authentication method used when accessing Web applications. This verification method uses a domain controller system to authenticate agents. This ensures authentication for the domain controller to the agent.

The basic process for Kerberos authentication is as follows:

1. A client requests credentials for a server from the domain controller.
2. The domain controller gives the client the credentials, which include a session password, encrypted with the client's password (held by the domain controller).
3. The client transmits the identity to the server using the session key provided by the domain controller.
4. The server then authenticates the information as valid, and allows the client to proceed.

As further protection, the use of a ticket-granting server to supply authentication tickets is used. Ticket-granting servers allow sessions to maintain associated tickets, which are valid for only a limited duration and are invalidated at the request of the ticket-granting server. [12]

NTLM-Authentication (Windows Authentication or challenge/response) is a similar technology, developed by Microsoft. Kerberos Authentication is considered to be an authentication protocol superior to NTLM-Authentication, partially due to its ability to provide mutual authentication and grant tickets for a limited duration.

2.3.3.1.4 External Authentication Methods

Additional authentication methods can use an external host to provide credentials to a system if that user has credentials with the external host. The most frequently used authentication method of this type is OAuth. Corporations such as Google, Facebook, and Twitter use OAuth.

OAuth authentication is similar to Kerberos Authentication. There is a trusted authentication provider, a client requesting access, and a resource server. The resource server contains the resource that the client is attempting to access. Using the same method as does Kerberos, the client requests credentials - known as tokens - from the authentication server and presents them to the resource server. The resource server, once presented with a token, can verify whether the

credentials were provided by the authentication server based on various cryptographic details provided by a pre-established set of cryptographic keys or other information [13].

2.3.3.2 Secondary Authentication

Since form authentication is the most common authentication type available on the Web, there is inherent potential for an unauthorized user using stolen credentials. To counter this, there are various secondary authentication measures, which are widely used on the internet today. These methods send a message to a known external communication device and ask the user to enter a challenge response. A challenge response value contains the content provided in the external communication message. To send these messages, SMS and email are often used. Some authentication tools include key-fobs that generate a password, or part of a password, on a timed interval.

While these secondary measures are highly effective in preventing unauthorized access to machines, they are still cumbersome for the end user. This thesis presents a case that allows human typing pattern analysis to perform automatic secondary authentication and unencumbered by secondary devices. The value of this is that typing patterns are unique to every human, as they rely on his or her biological movement characteristics. Robots, on the other hand, perform computed and rapid input patterns, which are easier to detect.

2.4 Determining Uniqueness between Human Agents

Numerous methods of authentication exist to enhance internet security and local security. High security enterprises need many layers of authentication. These enterprises include military, government, and companies that handle sensitive information. As a second line of defense, security divisions often introduce additional measures to increase the certainty of authentication. Some of these already mentioned are “something you know” (username/password) and “something

you own” (cellphone/email/smartcard). The following section looks at biological methods to identify an agent, known as “something you are.”

2.4.1 Biological Differences

There are several universal ways to identify humans using biometric information. These types of identifications include fingerprint uniqueness, iris (eye) uniqueness, hand dimensions, voice evaluation, and signature uniqueness [14] [15] [16] [17]. An active area of research is the evaluation of computer interaction signatures – namely typing patterns and mouse movement patterns. While this thesis addresses several topics, an in-depth evaluation is only included for computer interaction signatures.

Sophisticated security systems rely on the exploitation of aforementioned uniqueness inherent in human users. The distinctiveness and permanence of traits in an organism are the bases for biological identification. Although most literature suggests that behavioral traits are weak biometric traits, it is possible to enhance security significantly by introducing these traits as by-products of authentication. The Internet currently lacks broad support for the use of other identification methods, as those methods require specialized hardware and software. However, Web authentication may soon use face and voice recognition, as HTML5 and devices with multimedia components become commonplace.

2.4.1.1 Fingerprints

It is common knowledge that fingerprint patterns can accurately identify an individual. Effectively matching fingerprints to an individual can accurately identify that individual.

2.4.1.2 Signature

Day-to-day business transactions often use handwritten signatures. Due to the number of intra-class variants of characters in handwriting over time, it is difficult to classify a person by his or her signature. Specialized hardware components analyze speed, shape, and pressure during the act of signing to increase the effectiveness of determining the handwritten signature. However, the adoption of this method of analysis is rare, as it requires external hardware.

2.4.1.3 Computer Interaction Signature

While handwritten signature capturing requires specialized hardware, computer interaction signatures require only standard input mechanisms used during computer interaction, such as a keyboard or mouse. Since these hardware components are already part of most personal computing systems, the convenience of implementing this evaluation becomes more viable.

Suggested by Jain and Kumar [17], biometric evaluation should encompass a few inherent traits: user convenience, data acquisition environment, and data acquisition quality. Standard mechanisms available on the Internet are able to provide all of these traits. Utilizing standard input devices provides user convenience, while additional information is gathered using custom JavaScript behind the scenes. The data acquisition environment is also provided, due to the same reasons as user convenience. Most interactions with a Web page allows for tracking any input mechanic, therefore, data acquisition must quickly adjust to support various metrics.

Capturing keystroke dynamics is possible by embedding additional biometric controls on a standard login form. This allows for effective capture of biometrics as though an agent filled out parts of the login page, which captures usernames and passwords.

Keystroke dynamics are captured efficiently through mechanisms provided by modern Web browsers. JavaScript event listeners capture keystroke down events and keystroke release events. By capturing these two mechanics, significant information is gathered on how a user’s muscle memory and typing patterns are utilized for identification.

Identifying a user requires a model, which contains the quantum information captured:

- Time a key is pressed (Down Time)
- Time between the previous key is pressed and subsequent key is pressed
- Time the first key is released (Release Time)

These keystroke dynamics create a model using these criteria for every key pressed during interaction with the form.

In Table 4 below, a user types a simple password “HELLO”. The information gathered here is useful to evaluate the typing pattern of the particular agent.

Key	H	E	L	L	O
Down Time (ms)	0	84	140	251	334
Release Time (ms)	114	184	205	307	402

Table 4 - Password times between keystrokes

In order to visualize this information, Figure 2 is provided for convenience.

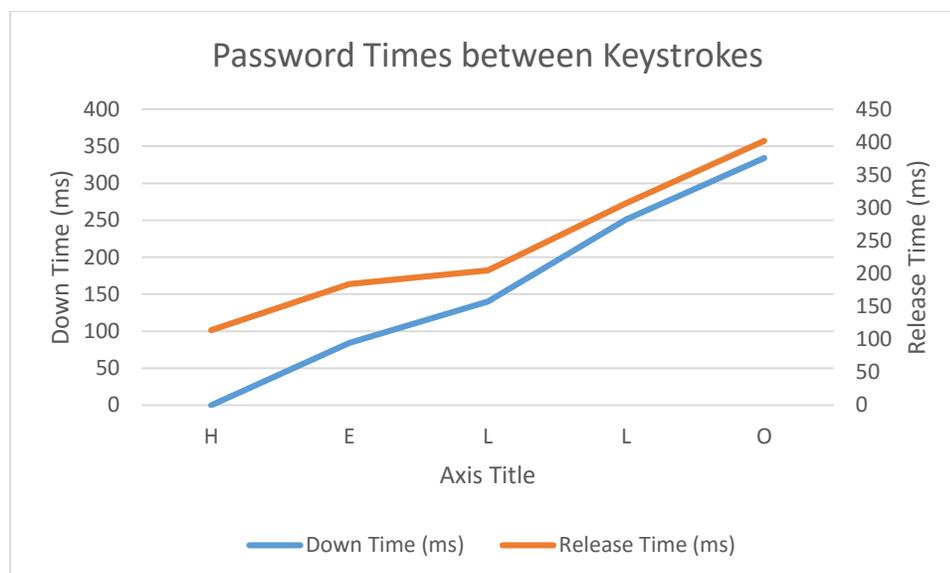


Figure 2 - Visual representation of typed characters

This information is used to create a series of metrics, which include the duration for which a key is held and the time until following key is pressed. Negative values in the time between press and release indicate that the subsequent key was pressed before the previous key was released; positive values specify a key was released before the succeeding key was pressed. This additional step aids in authentication, as the data now contains overlap information. Table 5 is shown below.

Key	H	E	L	L	O
Held Time (ms)	114	100	65	56	68
Time between press and previous key released (ms)	0	-30	-44	46	27

Table 5 - Differences in time based on when a key is pressed and the previous key released

There are two axis points displayed in Figure 3, which allow for classification later. The alternating values of key-down time and time between keypresses are used to generate a timing vector that serves as classification training data.

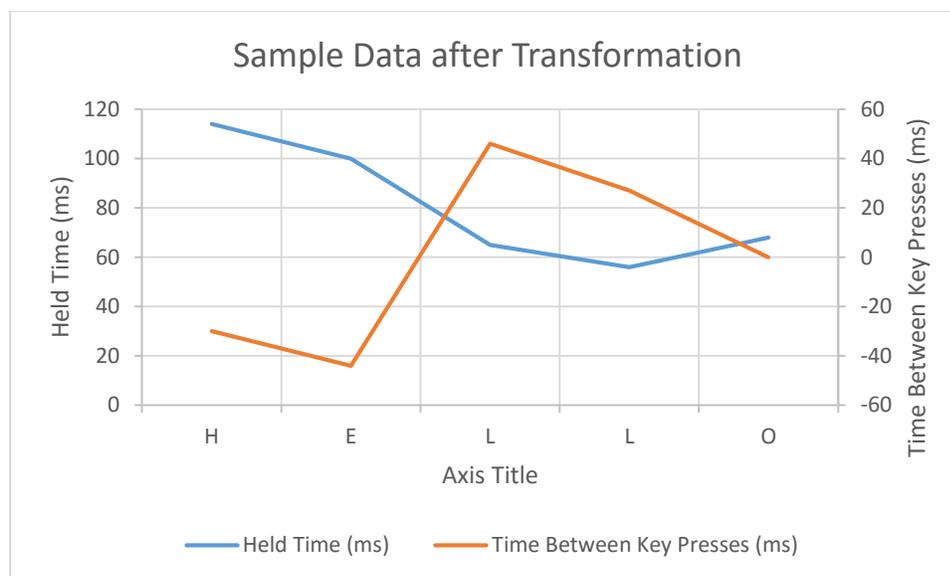


Figure 3 - Pattern of the transformed data that has been captured.

The timing vector of this sample is [114, 0, -30, 100, -44, 65, 46, 56, 27, 68]. Some researchers, such as Cho [18], add a keystroke to supplant additional values at the end of the timing vector by pressing the return key. The implementation provided in this research supplants zero as the second value of the dataset to indicate the start of initial data capture, although this value could be omitted.

Several mechanisms exist to classify a user's keystroke dynamics. Researchers capture and apply this data for use in backpropagation neural networks. Results as low as 2% false acceptance rate (FAR) and false rejection rates (FRR) have been achieved [19] [18]. Another user identification method is to create a master trajectory profile that categorizes a user based on the amount of dissimilarity encountered. Results have reached FAR and FRR as low as 4% using this method [20].

Including additional measures to increase the number of inputs is also achievable. Garg [21] advocates using meta-key information such as key type change (alpha to numeric) and the number of times the backspace key, shift key, and caps lock key are pressed during data entry.

Shanmugapriya [22] suggests the creation and use of a new measure called “Virtual Key Force” – a measure that takes into account the time between key presses and the distance between keys being pressed. Zhong, et al. [23] also use distances as additional metrics; however, their research suggests the typical use of Manhattan distance is insufficient, as it does not scale properly. Alternatively, Zhong suggests a hybrid approach of using Manhattan distance and Mahalanobis distance.

The initial presentation of the sample data in Table 4 only considers the key press and hold duration times. To create a superior model of an individual’s behavioral patterns, measures are added to increase the number of data points. Every keystroke is captured including shift, caps lock, and control keys as normal key inputs, in the model presented later in this thesis. These additional keys are used to model the user’s input.

While many researchers claim that backpropagation is a superb tool for determining a user’s validity, there remains the aspect of training the model appropriately [18] [19] [23] [24] [25] [26]. Real systems may find it difficult to train a neural network properly, as the network must be retrained for every new user [24]. Even with continuous integration of valid user authentication data, the neural network training requires valid and invalid login attempts to identify agents adequately. It is prohibitively expensive for production systems to retrain a model every time a user is created, due to the training needs of neural networks.

CHAPTER 3

PURPOSE

This thesis analyzes the behaviors present in both human and robot agents through Web form interaction. By comparing human and robot agents, the potential for reduction in internet traffic and fraudulent impersonation of human can be mitigated. An attempt to identify pragmatic measures using biometric measures as a secondary authentication is also tested.

If successful, the discrimination between humans and robots relieves humans of the need to perform secondary authentication measures such as CAPTCHA. Instead of having an active system to determine humanness, these measures should be evaluable automatically through biometric evaluation. The enhanced security mechanisms present in biometrics will enable web applications to become more secure against fraudulent login attempts.

CHAPTER 4

METHODOLOGY

To apply an automatic secondary authentication measure for Web applications that requires no additional user input, a Web application was created. This Web application captures a visiting agent's HTTP headers, session information, and meta-information relating to keyboard and mouse inputs. The information was subsequently used to identify robot agents and compare human agent biometric signatures. The next step is to perform formal research into these various behavioral patterns, as will be discussed in this section.

4.1 Web site to track inputs

A data capture Web application was created with the intent on capturing aspects of human and robot behavior. The Web site can be found on the public Internet at <http://www.trytologin.com>. All software development efforts were done in C#, JavaScript, CSS, and HTML for the application. This Web page captures HTTP headers, IP Addresses, and form interaction behaviors such as key presses and mouse clicks. A sample of each of these data points is provided in Tables 6, 7, 8, and 9.

Table 6 displays one agent's request headers that visited the Data Capture Web Site.

Name	Value	Date Created
Connection	keep-alive	9/28/2015 3:58 AM
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8	9/28/2015 3:58 AM
Accept-Encoding	gzip, deflate, sdch	9/28/2015 3:58 AM
Accept-Language	en-US,en;q=0.8	9/28/2015 3:58 AM
Host	localhost:20151	9/28/2015 3:58 AM
User-Agent	Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.42 Safari/537.36	9/28/2015 3:58 AM
Upgrade-Insecure-Requests	1	9/28/2015 3:58 AM

Table 6 - Sample Request Headers

Table 7 displays that same agent's IP address, referrer, and request type. This request shows that this particular agent submitted information to the Data Capture Web Site.

Date Saved	IP Address	Referrer	Url	Request Type
9/28/2015 3:58 AM	76.26.211.37	http://trytologin.com	http://trytologin.com	POST

Table 7 - Sample request information

Table 8 shows the keystroke dynamics by the agent.

Key Pressed	Element Name	Alt	Ctrl	Shift	Time Released	Time Pressed
72	Email	0	0	0	112	0
69	Email	0	0	0	144	-32
76	Email	0	0	0	408	312
76	Email	0	0	0	144	40
79	Email	0	0	0	192	88
72	Password	0	0	0	2647	2520
69	Password	0	0	0	2695	-63
76	Password	0	0	0	369	288
76	Password	0	0	0	152	56
79	Password	0	0	0	160	80

Table 8 - Sample input capture (Keyboard)

Similar to Table 8, Table 9 shows the mouse click interactions by the agent.

MouseX	MouseY	Button	Element Name	Time Pressed
581	240	0	Password	9/27/2015 11:58 PM
582	195	0	Email	9/27/2015 11:58 PM
664	265	0	Password	9/27/2015 11:58 PM
561	331	0	submit	9/27/2015 11:58 PM

Table 9 - Sample input capture (Mouse)

4.1.1 How information is collected and used

As stated above, many of the users' interactions are captured during the browsing session that the users perform. All information is stored and submitted in a database; the schema is shown in Figure 4.

The only information used to validate a user were the keystroke patterns that were entered. Since only keystrokes were used to validate users, this mechanism should be treated as a secondary authentication metric.

4.1.1.1 Form Input

The screenshot shows a web browser window with the address bar displaying 'www.trytologin.com'. The page has a dark navigation bar with links for 'Try To Login', 'Login', 'About', and 'Contact'. The main content area features the heading 'Try To Log in.' followed by the text 'Try to log in.' Below this, there are two input fields: 'Email' and 'Password'. A 'Log in' button is positioned below the password field.

Figure 4 - Image of the data capture application

The Web server receives captured data submitted from a normal HTML form. Keystroke and mouse dynamics are captured through user interaction, and are submitted with the form when the

user pressed the submit button. The user only needs to interact with normal login fields, while no additional interaction is necessary to capture keystroke and mouse data.

Interacting with email and password elements inserts hidden inputs into the form. The information captured by this form is comprised of mouse click metrics, key press metrics, and information about which form element is being acted on at the time. The agent submits data to the Web server that stores the information in a database. Sections of this thesis that follow use these hidden inputs to determine the uniqueness of human agents. Refer to *Table 8 - Sample input capture (Keyboard)* and *Table 9 - Sample input capture (Mouse)* for sample input data.

4.1.1.2 Database Schema

The database architecture used to capture and store data from the application is Microsoft SQL Server 2014. Figure 5 - Data Capture Database Schema provides the name, datatype, and nullable properties for each column. The ID column on each table represents that table's Primary Key (PK). Any field suffixed with “_id” represents a Foreign Key (FK) to the appropriate table.

The ApplicationRequestContexts table contains high-level information pertaining to each request. RequestHeaders has a many-to-one relation with ApplicationRequestContexts and tracks each HTTP header per request. MouseInteractions and KeyboardInteractions contain a many-to-one relationship with FormInteractions and track mouse and keyboard metrics, respectively. FormInteractions maintains a one-to-one relationship with ApplicationRequestContexts and acts as a parent table to MouseInteractions and KeyboardInteractions. The FormInfo table contains the raw input fields that the user submitted to the Web server and has a one-to-one relationship with ApplicationRequestContexts.

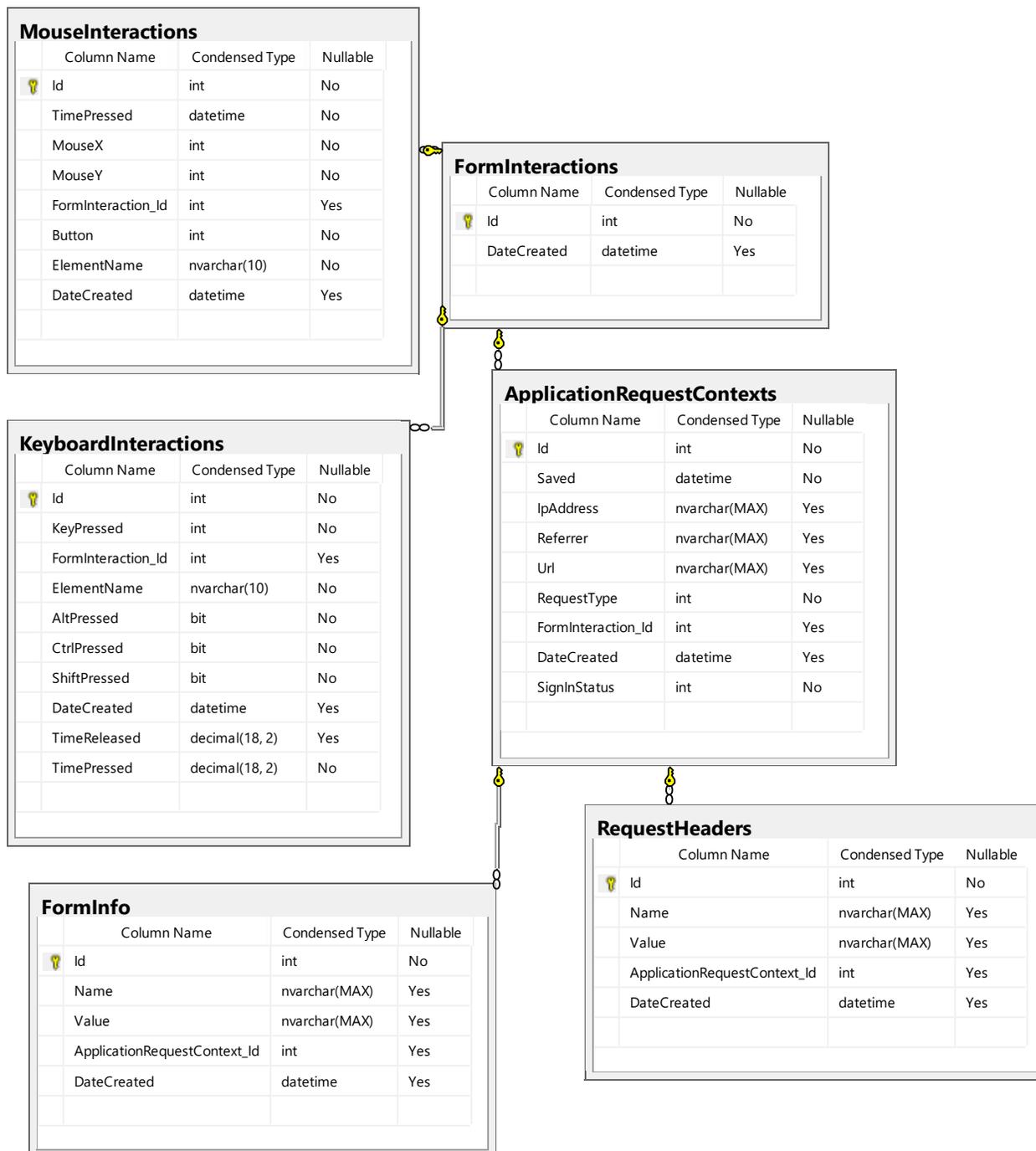


Figure 5 - Data Capture Database Schema

4.2 Web Robot Traffic Generation

Several commercial automatic applications were used to generate reproducible automated traffic. Many of these tools incorporate different methods to interact with Web forms, and try to mimic human behavior.

Besides the capture of commercially available Web robots, wild Web robots interacted with the Web site and their information was captured.

4.2.1 Commercial Automated Options

Several different commercially available programmable Web agents were used to generate traffic that submitted the Web form on the Data Capture Web Site. The agents used were Visual Studio Web Tests, Selenium, iMacros, and Sahi.

4.2.1.1 Visual Studio Web Tests

Visual Studio, a professional-grade development environment, provides methods to create automation tools. For the Web, however, Visual Studio does not allow for direct interaction with the Web page; instead, it opts to send HTTP POST requests with predefined parameters.

4.2.1.2 Selenium

Selenium is an open source Web testing framework used to automate tasks, and is widely used in the automation industry. It interacts with a Web browser to perform automated tasks. Since this runs commands through a browser, typical browser headers are shown.

4.2.1.3 iMacros

iMacros is a browser automation tool. This tool also uses a real Web browser to perform work and sends standard header information for each request.

4.2.1.4 Sahi

Sahi is another automation framework, which uses a Web browser to perform automated steps.

4.2.2 Human Biometrics Sample Data Points

It is crucial to collect a steady sample of data that is used to test the efficacy of different classification algorithms. For each of the experiments performed, the inclusion of six valid login attempts was used for training and an additional known login attempt was used for verification. Some of the algorithms used require additional training to classify the login attempt correctly; in such cases, an additional known invalid login attempt and several randomly generated login attempts were captured. For these tests, the simple password of “hello” was used. The various input parameters are shown below.

In Tables 10, 11, and 12, the columns prefixed with “H:” indicate the duration a particular key was held, and columns prefixed with “RT:” indicate the time after the previous key was released. If the “RT” columns are negative, the previous key was not released when the key was pressed. All times are represented in milliseconds.

Table 10, shows the set of six known valid login attempts used to train models. This data is used in the training of each classification algorithm to determine if the same user is attempting to login to the Web application.

Attempt	H: 'h'	RT: ''	H: 'e'	RT: 'h'	H: 'l'	RT: 'e'	H: 'l'	RT: 'l'	H: 'o'	RT: 'l'
Attempt 1	112	0	144	-32	408	312	144	40	192	88
Attempt 2	96	0	184	87	456	376	152	56	168	96
Attempt 3	152	0	191	-87	393	304	143	40	193	89
Attempt 4	87	0	337	225	353	272	151	46	183	64
Attempt 5	121	0	144	32	311	231	153	48	187	88
Attempt 6	112	0	144	-32	408	312	144	40	192	88

Table 10 - Valid Login Attempts

Table 11 shows additional captured logins that are used to help validate the model. Validation rules indicate the invalid login should be marked as invalid when validating and the valid login should be marked as valid.

Attempt	H: 'h'	RT: ''	H: 'e'	RT: 'h'	H: 'l'	RT: 'e'	H: 'l'	RT: 'l'	H: 'o'	RT: 'l'
Invalid Attempt	112	0	168	-48	104	24	168	55	136	56
Valid Attempt	152	0	184	-72	368	296	152	49	137	41

Table 11 - Additional Known Attempts

Many of the tests require additional training in order to build an accurate classification profile.

The randomly generated attempts shown in Table 12 are used to simulate invalid login attempts to the Web server.

Generated #	H: 'h'	RT: ''	H: 'e'	RT: 'h'	H: 'l'	RT: 'e'	H: 'l'	RT: 'l'	H: 'o'	RT: 'l'
1	197	0	32	-237	111	-112	223	-63	62	71
2	18	0	55	140	146	161	60	47	240	214
3	138	0	98	-81	153	-231	240	-166	183	-98
4	129	0	5	154	129	29	84	185	47	112
5	19	0	128	-83	234	177	147	14	233	160
6	74	0	134	115	138	149	48	-156	16	-167
7	219	0	29	-46	162	-4	40	-103	110	182
8	182	0	63	-99	240	80	219	-144	4	-47
9	176	0	177	62	221	123	66	147	16	-101
10	41	0	69	106	31	-46	206	-85	97	165

Table 12 - A sample of randomly generated login attempts

Figures 6 and 7 present the known valid and invalid records.

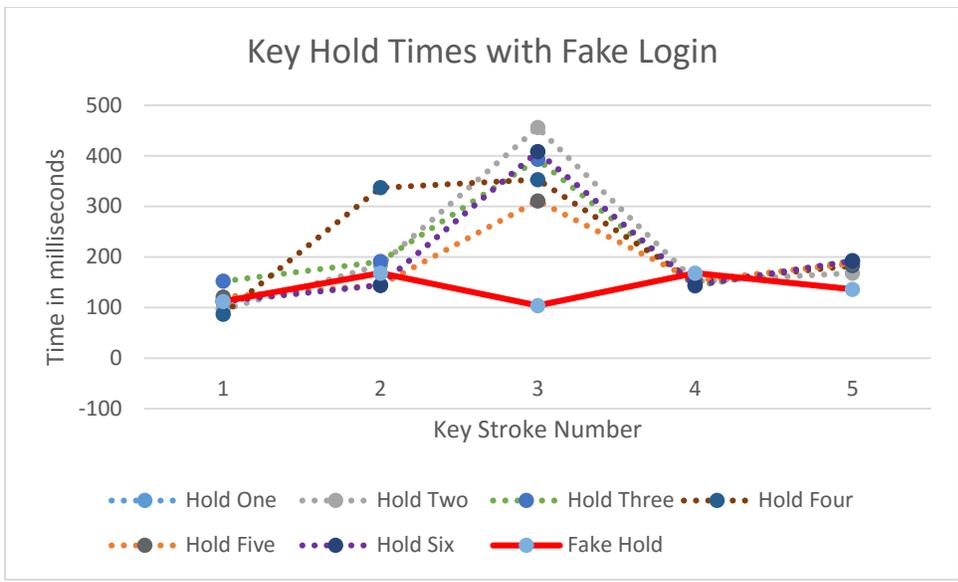


Figure 6 - Key Hold Times with Fake Login

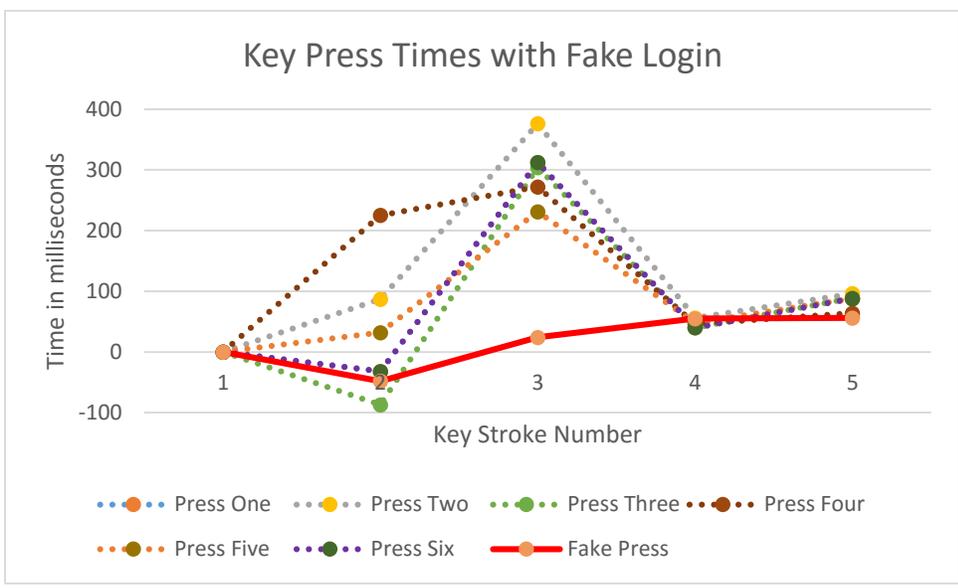


Figure 7 - Key Press Times with Fake Login

4.3 Comparison Methods

Once sufficient data was gathered, several different machine learning and classification algorithms were trained and tested to determine if low-volume training sufficed to determine login validity

based on key dynamic metrics. The methods used to create the validation models are backpropagation, support vector machines, decision trees, and dissimilarity comparison.

4.3.1 Backpropagation

Backpropagation has been a topic for many researchers who attempt to authenticate users via keyboard dynamics [19, 18, 20]. In some of the reports, backpropagation has the potential to provide accurate results regarding the amount of training provided.

In the research for this thesis, data was gathered by a Web application created to do so. User's typed password strokes were captured during several login attempts. In this way, a backpropagation neural network was trained using the C# programming language and AForge.Neuro neural network libraries [27].

Several potential models were created to train the model. The backpropagation model was trained using the valid training samples described above and 100 to 2000 generated invalid logins.

The backpropagation network was configured as follows:

- Number of inputs: 10
- Number of inner layers: between one and five
- Number of neurons per layer: between one and 1000
- Number of outputs: 1

This method required the use of 10 inputs, as per the number of features in the sample set, between one and five inner layers with between one and 1000 neurons in each layer, and one output. The classification of a result is valid if the neural network gave an output ≥ 0 . The classification of a result is invalid if the result were less than 0.

4.3.2 Support Vector Machines

Support vector machines add reliable classifications to a data model. While exploring various methodologies to perform classifications, this classification system seemed promising for many types of analysis. The implementation of a support vector machine was used to gauge the effectiveness of this model of input.

Using Accord.Neuro framework [28], the support vector machine evaluated was a kernel support vector machine, which had the following parameters:

- Number of inputs: 10
- Kernel: Gaussian with a sigma of 100
- Learning Algorithm: sequential minimal optimization

The tests run for the SVM ran 1000 times to gain an average baseline. These tests used the set of valid inputs and the known invalid input for training the SVM. Most tests used a set of randomly generated data to train as invalid login attempts. The tests included either 0, 10, 100, or 1000 randomly generated inputs to use during training. Once training finished, 10,000 login attempts were classified. The classification of a result is valid if the SVM gave an output ≥ 0 . The classification of a result is invalid if the result was less than 0.

4.3.3 Decision Trees

When training the decision trees, several different parameters regarding invalid testing data were evaluated. The decision tree engine used the C45 algorithm, ten inputs, and continuous variables ranging from -10000 to 10000. Training of the engine included six valid logins, one invalid login, and 10,000 generated invalid login attempts. Results obtained from this initial model encouraged the use of additional training using randomly generated invalid logins. Experiments were then

performed using training by 1, 10, 100, 1000, and 5000 randomly generated invalid logins. The classification of a result is valid if the SVM gave an output of one. The classification of a result is invalid if the result were equal to zero.

4.3.4 Dissimilarity Comparison

A dissimilarity comparison function compares known inputs to build a generalized model with various statistics and compares these values against an attempted validation.

The general approach of this comparison has its base in statistical modeling. Several valid login attempts were captured to create a master model, also known as a master trajectory. Five known valid inputs were used to create the initial model during the initial implementation of the algorithm. The average value for all known features was calculated and stored as if it was a single valid login when creating this master trajectory. Comparison of each valid login attempt to the master trajectory was used to calculate average dissimilarity and standard deviation. Comparison of average dissimilarity against any additional input to perform verification takes place once this model was created.

The set of inputs for an attempted login comprises the time a key is pressed, and the previous key is released. The values are stored in an array of numeric values such as [80, 0, 134, -31, 55, 210 ...].

These data points are plotted on a Cartesian map to create the known dissimilarity profile; for the sample just provided: [{X = 80, Y = 0}, {X = 134, Y = -31}, {X = 55, Y = 210} ...].

Equation 1 shows the general equation for calculating the dissimilarity between two profiles (M and A).

$$Dissimilarity_{(M,A)} = \sum_{i=1}^n \sqrt{(Mxi - Axi)^2 + (Myi - Ayi)^2}$$

Equation 1 - Dissimilarity Equation

The equations shown in Equation 2 show the evaluation criteria needed in order to validate if a user has typed in the same manner as his or her previous attempts.

*Evaluation Threshold = DissimilarityAverage + (Sigma * DissimilarityStandardDeviation)*

IsValidLogin(attempt) = Dissimilarity(master, attempt) ≤ Evaluation Threshold

Equation 2 – Valid login Function using Dissimilarity method

When a new login attempt takes place, the dissimilarity between the new attempt and the master trajectory are calculated. This calculation is then compared against an evaluation threshold. If the dissimilarity of the attempt is less than the evaluation threshold, the user is authenticated.

CHAPTER 5

ANALYSIS OF RESULTS

All data used for analysis can be found at <https://goo.gl/zkP3TE>. Note that this link requires a Georgia Southern University login to access the information.

5.1 Traffic Analysis

Clues exist to identify agents as human or robotic when evaluating the traffic generated from the Web application. Most of the indexer robots that accessed the application had some identifier in the HTTP request header. However, other factors should be evaluated, including the inter-page accesses and form information gathered by form submissions.

5.1.1 HTTP Headers and Access Times

It is possible to identify robot agents through the evaluation of HTTP headers. However, with more sophisticated robot agents, these headers appear as valid user requests. As described in the introduction, there is some disparity between what Web browsers and robot agents submit as HTTP headers. By utilizing the Data Capture Web Site, the exclusion or inclusion of certain field headers quickly identifies potential robotic agents. However, it is more difficult to identify human agents from one another if they were to use the same Web browser. Table 13 shows the most common User-Agent headers that were submitted to the Data Capture Web Site. Note the occurrence of values such as “bot”, “agent”, or “spider” (these rows are highlighted).

User-Agent Headers
Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/6.0)
Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Mozilla/5.0 (compatible; MJ12bot/v1.4.5; http://www.majestic12.co.uk/bot.php?+)
Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)
Mozilla/5.0 (Windows NT 5.1; rv:6.0.2) Gecko/20100101 Firefox/6.0.2
Mozilla/5.0 (compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm)
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.143 Safari/537.36
Mozilla/5.0 (iPhone; CPU iPhone OS 8_3 like Mac OS X) AppleWebKit/600.1.4 (KHTML, like Gecko) Version/8.0 Mobile/12F70 Safari/600.1.4 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Mozilla/5.0 (compatible; NetcraftSurveyAgent/1.0; +info@netcraft.com)
Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120 Safari/537.36
Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36

Table 13 - User-Agent Headers

Several autonomous agents saw sub-half-second resource access times. Tables 14 and 15 display the resources accessed and their headers. Resources accessed include “robots.txt”, which is generally a resource accessed by robots. The headers for the same records specify that the User-Agent is a robot.

IP Address	Url	Date
52.26.82.4	http://www.trytologin.com/	10/24/2015 7:05:17.677
52.26.82.4	http://www.trytologin.com/	10/24/2015 7:05:17.770
52.26.82.4	http://www.trytologin.com/Home/About	10/24/2015 7:05:18.113
66.249.79.217	http://www.trytologin.com/robots.txt	10/31/2015 1:42:35.257
66.249.79.217	http://www.trytologin.com/	10/31/2015 1:42:35.333
66.249.79.217	http://www.trytologin.com/robots.txt	11/7/2015 17:17:34.660
66.249.79.217	http://www.trytologin.com/	11/7/2015 17:17:34.767
158.69.225.37	http://www.trytologin.com/robots.txt	12/21/2015 9:31:31.087
158.69.225.37	http://www.trytologin.com/	12/21/2015 9:31:31.230
216.145.14.142	http://www.trytologin.com/	12/25/2015 1:49:41.457
216.145.14.142	http://www.trytologin.com/robots.txt	12/25/2015 1:49:38.847

Table 14 - Resources Accessed by Robots

IP Address	Date	User-Agent
52.26.82.4	10/24/2015 7:05:17.677	BusinessBot: Nathan@lead-caddy.com
52.26.82.4	10/24/2015 7:05:17.770	BusinessBot: Nathan@lead-caddy.com
52.26.82.4	10/24/2015 7:05:18.113	BusinessBot: Nathan@lead-caddy.com
66.249.79.217	10/31/2015 1:42:35.257	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
66.249.79.217	10/31/2015 1:42:35.333	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
66.249.79.217	11/7/2015 17:17:34.660	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
66.249.79.217	11/7/2015 17:17:34.767	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
158.69.225.37	12/21/2015 9:31:31.087	Mozilla/5.0 (compatible; Lipperhey-Kaus-Australis/5.0; +https://www.lipperhey.com/en/about/)
158.69.225.37	12/21/2015 9:31:31.230	Mozilla/5.0 (compatible; Lipperhey-Kaus-Australis/5.0; +https://www.lipperhey.com/en/about/)
216.145.14.142	12/25/2015 1:49:41.457	Mozilla/5.0 (Windows; U; Windows NT 5.1; en; rv:1.9.0.13) Gecko/2009073022 Firefox/3.5.2 (.NET CLR 3.5.30729) SurveyBot/2.3 (DomainTools)
216.145.14.142	12/25/2015 1:49:38.847	Mozilla/5.0 (Windows; U; Windows NT 5.1; en; rv:1.9.0.13) Gecko/2009073022 Firefox/3.5.2 (.NET CLR 3.5.30729) SurveyBot/2.3 (DomainTools)

Table 15 - Fast Resource Access Headers

These results agree with other researchers' results and report no new results.

5.1.2 Submitted form Information

When evaluating submitted form information from several different sources, it was apparent that wild robots did not submit the Web form. This is likely due to the robots being indexers. As a result, these agents were not assessed for biometric information. However, scripted agents and human agents submitted form information. One reason the form submission functionality is excluded is in consideration of the lost time and resources when a Web Site performs input validation.

5.1.3 Human Agents

Human agents are often identified primarily by slower request times and by submitting particular timing vectors. Table 16 displays timing vectors for one of the human agents that provided information on the Web form. Negative values in the Time Pressed column indicate that the key was pressed before the previous key was released.

Shift Pressed	Time Released	Time-Pressed
0	214	0
0	101	6
0	274	-79
0	374	-154
1	704	176
1	544	-208
0	561	441
0	664	-96
0	737	-87
0	175	80
0	263	-71
0	176	32
0	232	-104
0	225	104
0	64	2
0	239	40
0	278	-119
0	321	153

Table 16 - User submitted timing vector for Email Field

This agent is categorized as a human due to the variance in the timing vector. These results are provided for comparison of robotic behavior in the following section.

5.1.4 Visual Studio Web Tests

Visual Studio Web Tests do not allow for direct interaction with the Web page; instead, it opts to send predefined HTTP POST requests. Visual Studio tests are identified as robotic behavior by

evaluating the inter-request time between page access, the subsequent posting of data, and the posting of the identical information.

Successfully capturing and analyzing data has provided striking results. Request speed is already known to be a common robot detection marker. Nevertheless, repetitive data posting might indicate a new robot detection marker. However, to validate this new marker more research is needed.

5.1.5 Selenium

Selenium uses a Web browser to send commands, resulting in the identification of typical browser headers. However, it is determined that a robot is interacting with the form when the framework is used naively due to the rapid insertion of key presses. These inter-key press times are executed with sub-two millisecond timings, a speed that equates to superhuman typing speeds. Though there exist longer delays, it depicts atypical human typing speeds.

Table 17 shows the keyboard dynamics profile as captured from the Selenium agent.

Shift Pressed	Time Released (ms)	Time-Pressed (ms)
0	23	0
0	8	1
0	4	1
0	5	1
0	5	0
0	5	1
0	4	0
0	5	2
0	76	1
0	40	-35
0	39	34
0	6	2
0	4	1
0	5	1
0	5	2
0	6	1

Table 17 - Selenium submitted timing vector for Email Field

However, since a programming language drives Selenium, it is possible to create delays between key presses to modify this behavior.

As suspected, this agent is easily identifiable through its speed of typing. The speed of data input is not a well-researched topic, and this provides some validity into keystroke dynamics as a detection marker.

5.1.6 iMacros

iMacros posts standard header information, as it uses the actual browser to perform work. However, when inputting data into the login form, no key strokes are sent to the form. Since no key strokes are sent to the form, it becomes apparent that this was not a user typing login information. In this scenario, no timing vectors were captured. The submission of a form without any common input markers is submitted on behalf of an automation tool and is an additional input marker for detection of robots.

5.1.7 Sahi

Sahi uses a Web browser to perform automated steps. Request headers cannot determine if a robot is an agent performing actions, due to the agent using a Web browser. Determination of a robot agent was completed by evaluating the submitted key press interval and hold times; these were typically less than two milliseconds in duration.

Table 18 shows a captured timing vector from the Sahi agent. The input timing from this agent shows rapid keypress and key release timings, faster than humans can consistently do so. The shift pressed column shows a Boolean values (1 = true, 0 = false) if the shift key was pressed with another key. The Time Released and Time Pressed columns show timing in milliseconds.

Shift Pressed	Time Released	Time-Pressed
0	1	0
0	1	1
0	1	1
0	2	1
0	0	0
0	1	1
0	1	0
0	1	0
0	1	0
0	1	1
0	1	1
0	1	0
0	1	0
0	1	0
0	0	0
0	1	1

Table 18 - Sahi submitted timing vector for Email Field

Similarly to the Selenium experiment, results captured here also reflect that rapid input times are due to an automated agent entry. This further adds to the validity that rapid input timing is a marker for robot agents.

5.2 Timing Vector Analysis

The behavior patterns the automated agents and humans exhibit are vastly different. Often, HTTP headers differ and display information indicative of robot presence. Other times, interaction of the web page differs significantly. Evaluation of the form interaction patterns presents a clear indication of the differences in input patterns. Humans, for example, exhibit varying input patterns. Robots tend to maintain constant interaction patterns - if they are provided at all. In addition, robot page request time is often much faster than that of human agents; as previously shown in Table 14 - Resources Accessed by Robots.

5.2.1 Human v Human analysis

In the later sections, various algorithms including backpropagation, support vector machines, decision tree classification, and dissimilarity comparison are evaluated. For each of these algorithms, the data that was used is described in the methodology.

5.2.1.1 Backpropagation

While promising in individual research applications, the length of time required by backpropagation training prohibits its use in modern Web applications. Many researchers, who experimented with this method to provide a biometric signature for login, utilized several different human agents to provide negative training models in their research. Since this amount of training is not practical in the normal iteration of training, additional measures need to be evaluated that either extend or differ in methodology provided here. It is possible that when registering a new user account to capture a common phrase and use lazy learning via backpropagation to train the model using a single phrase or word. However, this method requires retraining for every individual during account registration. This does not seem like a practical architecture to use in production

ready systems since the training and comparison needs to compare thousands (or more) user inputs to provide invalid login data.

Initial testing of backpropagation yielded unexpected results. While creating valid training using six input samples, five yielded similar data input profiles as shown in Figure 6 - Key Hold Times with Fake Login and Figure 7 - Key Press Times with Fake Login.

Even after training the network with 2000 randomly generated invalid login attempts, the backpropagation algorithm calculated these attempts with the same certainty rate as valid login attempts. Because of the small sample of valid logins and training with the invalid data set, there is a high probability of accepting invalid user logins. When training the neural network without these randomly generated login attempts, there are only minor differences in classification certainty. These results clearly show that a more robust training system is required to identify the biometric markers of typing patterns correctly.

The experiments performed using backpropagation were ultimately unsuccessful. Most research on this topic supports backpropagation as a highly successful means of providing authentication. However, the results gathered with this research suggest otherwise. Since this method requires a substantial amount of training per potential user of the system, this method is not a good choice for determining invalid attempts using Web forms.

5.2.1.2 Support Vector Machines

As another promising method of classifying data, utilization of support vector machines provided successful results. With adequate random records trained as invalid logins, support vector machines accurately determine if a user is valid based on keyboard dynamics.

Results of this training are shown in Table 19. Each test performed 10,000 attempted logins.

Table 19 shows the results of training and classification, as well as performance metrics.

# Random Training Records	FRR	FAR	Memory Usage*	Processing Time*
5000	11.1%	0.0007%	684MB	1 Second
1000	1.3%	0.0013%	43MB	288 ms
100	0.1%	0.0018%	10MB	82 ms
10	0%	0.0069%	9.7MB	54 ms
0	100%	100%	9.7MB	51 ms

Table 19 - Results of Support Vector Machine Training

It is apparent that the inclusion of the invalid training records significantly increases the accuracy of the FAR. When utilizing 1000 or more random training records, memory and processor consumption explodes. Training with many generated logins causes performance issues if the system is in constant high demand.

Support vector machines are touted as highly accurate methods of clustering data, and the results above suggest that this holds true for these experiments. SVMs display accuracy, which allows for secondary authentication using a user's keystroke dynamics.

5.2.1.3 Decision Tree

Decision tree classification shows more promise than does backpropagation classification for this data. Since only a few inputs that classify as valid logins exist, using randomly generated invalid values contribute to a very high degree of accuracy towards classifying invalid logins.

When using the 6 known login attempts for the user input that was captured and one invalid attempt to train, the false acceptance rate (FAR) is consistently between 55 and 120 attempts per 10,000 attempts (.55% to 1.12% false acceptance, with an average of .7977%) when using 10,000 generated login attempts to test the model against. The false rejection rate (FRR) for this model yielded no errors in the number of falsely rejected attempts when using zero generated login attempts; these attempts include the initial valid model and an additional attempted login.

As stated in the methodology, attempts to decrease the false acceptance rate used additional invalid login training. While the decrease in false acceptance when trained with more invalid logins, the increase in false rejection rates may designate the need for true training rather than generated training.

FAR and FRR values for all tests are shown in Table 20. These values are per 10,000 generated logins. Max and min values are calculated from repeating testing iterations 1000 times.

Generated Invalid Login Count	False Acceptance Min	False Acceptance Max	False Acceptance Average	False Rejections
0	55	110	79.77	0
10	0	121	60.681	418
100	19	134	79.708	535
1000	0	82	27.736	826
5000	0	32	6.079	775

Table 20 - Decision Tree Results (values are per 10,000)

While it is apparent that the false acceptance rate is reduced once significant training is undertaken, the number of false rejections increases for most iterations.

This experiment was unsuccessful at classifying valid authentications using keyboard dynamics. This is due to the high false rejection rate seen when using more invalid training data.

5.2.1.4 Dissimilarity Comparison

Using the test data with this method produced promising results. Utilizing only six samples to train the model, this method has given an FAR of $< 0.04\%$ when tested against a known invalid login, and 1000000 generated login attempts with a sigma of 3.0 (explained in the methodology). This equates to less than 400 invalid login attempts per million.

This method is a highly accurate means of keystroke dynamics classification. To further test the accuracy of this algorithm, another known login attempt was added to the training set. This login attempt pattern differed significantly from the initial training set. A new master trajectory profile was created which accommodated the addition. Tests were then executed against a valid and invalid login attempt. With the initial sigma value of three, the dissimilarity comparison method lends itself to be more accepting of invalid attempts. The initial sigma of three classified the invalid attempt as valid. By reducing the sigma value to one, the known invalid login attempt was classified as invalid and the valid login attempt was classified as valid. The same test of comparing one million generated records against the verification algorithm using this model was executed and resulted in a significant reduction in accepted logins – an FAR of $< .03\%$ was consistently reported.

Once, the outlier record was removed from the training set and the sigma was set to one, in an attempt to reduce FAR. Valid and invalid tests were classified accurately. When compared with the one million randomly generated attempts, none of these attempts were successful. This result was unexpected and the test was performed one hundred additional times. The results conclude with an FAR of $3/100,000,000$ or $.000003\%$. All valid samples and the additional login attempt were classified as valid. However, by reducing the sigma value to 1, there are potential increases in false rejection rates.

As with SVM results, this method also successfully identifies keystroke dynamics by pattern analysis. This method does not require training against invalid login attempts to classify accurately.

5.3 Summary

It is feasible to identify robot agents and human actors with a high degree of accuracy using interaction patterns of a Web application. However, many of the behavioral patterns exhibited by robots are programmable to act like humans. Clever programming can mimic humans by slowing interactions, prioritizing click paths, changing the rate that input commands are sent, and making requests from known web browsers. Humans are also uniquely identifiable by their biometric signatures while interacting with Web pages.

Many Web robots self-identify using HTTP headers. Most of the robots whose data were captured via the Web application were typically found doing one of the following:

- Submitting an HTTP Agent header that identifies as a bot
- Accessing robots.txt
- Having fast request times impossible for a human user
- Performing abnormal click paths
- Accessing hidden resources
- Having exceptionally fast keystrokes

Early identification and session termination can reduce network and processor load of the Web server. This reduces the risk of DDoS attacks. By distinguishing robots with malicious intent, additional security risks are mitigated by disallowing these agents to attempt to login to the system.

In the event of humans logging into a Web application, it is viable to utilize layers of security to the login process. It has been shown that even with small amounts of valid training data, accurate identification models identify a particular user based on keystroke dynamic biometric signatures. Many researchers suggest the implementing neural networks to identify users; however, without

significant training of each user model, the applicability of neural networks and backpropagation is not feasible at this time. A superior set of choices is found in using support vector machines or through dissimilarity comparison, which had false acceptance rates of less than .01 percent of the randomly generated login attempts, and false rejection rates, which do not inhibit the user through most use cases.

CHAPTER 6

FUTURE WORK

While the use of keystroke dynamics to identify human agents is beneficial for security purposes, the behavioral changes humans exhibit over time may cause inaccuracies in long-term model generation. Temporal analysis of these behavioral changes is needed to bring keystroke dynamics to worldwide use. Models with higher accuracy may be developed by identifying a subset of valid records that best fits behavioral patterns demonstrated by a particular human.

CHAPTER 7

CONCLUSION

By evaluating the various information that a Web agent provides to a Web server, a promising assortment of performance and security optimizations can be implemented. Differentiating humans from Web robots and limiting the robot's resources will significantly reduce bandwidth consumption. Lastly, coupling of the capture of biometric information with its subsequent analysis increases security quite dramatically.

REFERENCES

- [1] Y. AlNoamany, M. C. Weigle and M. L. Nelson, "Access Patterns for Robots and Humans in Web Archives," Old Dominion University , Norfolk, 2013.
- [2] D. Doran and S. S. Gokhale, "A Classification Framework for Web Robots," *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE AND TECHNOLOGY*, 2012.
- [3] I. Zeifman, "Report: Bot traffic is up to 61.5% of all website traffic," Incapsula, 9 December 2013. [Online]. Available: <https://www.incapsula.com/blog/bot-traffic-report-2013.html>. [Accessed 23 August 2015].
- [4] I. H. Witten, M. Gori and T. Numerico, *Web Dragons: Inside the Myths of Search Engine Technology*, Elsevier, 2010.
- [5] M. H. M. J. H. X. Yuan, "An Efficient Scheme to Remove Crawler Traffic from the Internet," Department of Computing Science University of Alberta, Edmonton, Alberta, Canada.
- [6] D. Doran, *Detection, Classification, and Workload Analysis of Web Robots*, University of Connecticut, 2014.
- [7] D. Brewer, "Detecting Web Robots with Passive Behavioral Analysis and Forced Behavior," The University of Georgia, Athens, Georgia, 2010.
- [8] D. Nicholas and H. R. Jamali, "Web robot detection in the scholarly information environment," April 2008. [Online]. Available: <http://www.researchgate.net/publication/220195766>. [Accessed 4 July 2015].
- [9] D. Brewer and C. Pu, "A Link Obfuscation Service to Detect Webbots," University of Georgia, Athens.
- [10] A. Stassopoulou and M. D. Dikaiakos, "Web robot detection: A probabilistic reasoning approach," *Computer Networks*, pp. 265-278, 2009.
- [11] T. Fu, A. Abbasi and H. Chen, "A Focused Crawler for Dark Web Forums," *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE AND TECHNOLOGY*, no. June 2010, pp. 1213-1231, 2010.
- [12] C. Neuman, USC-ISI, T. Yu, S. Hartman and K. Raeburn, "The Kerberos Network Authentication Service (V5)," Network Working Group, 2005.

- [13] D. Hardt and Microsoft, "The OAuth 2.0 Authorization Framework," Internet Engineering Task Force (IETF), 2012.
- [14] N. K. Ratha and V. Govindaraju, *Advances in Biometrics*, London: Springer, 2008.
- [15] M. Stamp, *Information Security Principles and Practice*, Hoboken: Wiley-Interscience, 2006.
- [16] J. Trdine, *Advanced Biometric Technologies*, Rijeka: InTech, 2011.
- [17] A. K. Jain and A. Kumar, "Biometrics of Next Generation: An Overview," *Second Generation Biometrics*, 2010.
- [18] S. Cho, C. Han, D. H. Han and H.-I. Kim, "Web based Keystroke Dynamics Identity Verificaiton using Neural Network," *Journal of Organizational Computing and Electronic Commerce*, vol. 10, no. 4, pp. 295-307, 2000.
- [19] K. Revett, F. Gorunescu, M. Gorunescu and e. al, "AUTHENTICATING COMPUTER ACCESS BASED ON KEYSTROKE DYNAMICS USING A PROBABILISTIC NEURAL NETWORK".
- [20] K. Wangsuk and T. Anusas-amornkul, "Trajectory Mining for Keystroke Dynamics Authentication," Elsevier B.V., 2013.
- [21] U. Garg and Y. K. Meena, "User Authentication Using Keystroke Recognition," in *Proceedings of ICAdC, AISC*, 2013.
- [22] D. SHANMUGAPRIYA and D. G. PADMAVATHI, "VIRTUAL KEY FORCE – A NEW FEATURE FOR KEYSTROKE," *International Journal of Engineering Science and Technology*, vol. 3, no. 10, 2011.
- [23] Y. Zhong, Y. Deng and A. K. Jain, "Keystroke Dynamics for User Authentication," 2012.
- [24] F. Monrose and A. D. Rubin, "Keystroke Dyanmics as a biometric for authentication," *Future Generation Computer Systems*, no. 16, pp. 351-359, 2000.
- [25] R. A. Everitt and P. W. McOwan, "Java-Based Internet Biometric Authentication System," *IEEE Transactions On Pattern Analysis And Machine Intelligence*, vol. 25, no. 9, pp. 1166-1172, 2003.
- [26] B. Purgason and D. Hibler, "Security Through Behavioral Biometrics and Artificial Intelligence," *Procedia Computer Science 12*, pp. 398-403, 2012.
- [27] AForge Contributors, "AForge.NET :: Framework," [Online]. Available: <http://www.aforgenet.com/framework/>. [Accessed 18 July 2015].

- [28] Accord Framework Contributors, "Accord>NET Framework," [Online]. Available: <http://accord-framework.net/>. [Accessed 4 10 2015].
- [29] J. Jin, J. Offutt, N. Zheng, F. Mao, A. Koehl and H. Wang, "Evasive Bots Masquerading as Human Beings on the Web," IEEE, 2013.
- [30] A. Rappaport, "Robots & Spiders & Crawlers: How Web and intranet search engines follow links to build indexes," infoseek software.
- [31] K. Rivett, M. Gorunescu, F. Gorunescu, M. Ene, S. Tenreiro de Magalahaes and H. Santos, AUTHENTICATING COMPUTER ACCESS BASED ON KEYSTROKE DYNAMICS USING A PROBABILISTIC NEURAL NETWORK, London: University of Westminster.

APPENDICES

A Dissimilarity Algorithm

Code *I* - *Validate* *by* *Dissimilarity*

```

bool function verificationPattern (
    double[][] knownInputs,
    double[] inputToValidate,
    double sigma)
{
    var masterProfile = CreateMasterProfile(knownInputs);
    var dissimilarityProfile = CreateDissimilarityProfile(knownInputs, masterProfile);
    var inputToValidateDissimilarity = CalculateDissimilarity(masterProfile, inputToValidate);
    return inputToValidateDissimilarity <
        (dissimilarityProfile.Average() + (sigma*dissimilarityProfile.StandardDeviation()));
}

double[] function CreateMasterProfile(knownInputs)
{
    var masterProfile = new double[];
    for (var i = 0; i < length; i++)
    {
        for (var j = 0; j < knownInputCount; j++)
        {
            masterProfile[i] += knownInputs[j][i]/knownInputs.Count();
        }
    }
    return masterProfile;
}

double function CalculateDissimilarity(
    double[] masterProfile,
    double[] inputToValidate)
{
    var dissimilarity = 0d;
    for (var j = 0; j < lengthOfInput; j+=2)
    {
        dissimilarity += distance(
            masterProfile[j], //x1
            inputToValidate[j], //x2
            masterProfile[j+1], //y1
            inputToValidate[j+1] //y2
        )
    }
    return dissimilarity;
}

```

```
double[] CreateDissimilarityProfile(  
    double[][] knownInputs,  
    double[] masterProfile)  
{  
    var knownInputsDissimilarityForAverage = new double[lengthOfInput/2];  
    for (var i = 0; i < lengthOfInput; i+=2)  
    {  
        var dissimilarityLocal = 0d;  
        for (var j = 0; j < knownInputs.Length; j++)  
        {  
            knownInputsDissimilarityForAverage[i/2] += distance(  
                masterProfile[i], //x1  
                knownInputs[j][i], //x2  
                masterProfile[i+1], //y1  
                knownInputs[j][i + 1]//y2  
            )  
        }  
    }  
    return knownInputsDissimilarityForAverage;  
}
```

B Data Capture Layout Template Markup

```

<!DOCTYPE html>
<html>
<head>
  <meta name="description" content="Try To Login A Social Experiment. A human vs robot
  detection application. My graduate thesis research web page."/>
  <meta name="keywords" content="Try, To, Login, Try To Login, trytologin, tryto-
  login.com, bot, robot, webbot, human, detect, detection, identify, identification"/>
  <meta name="author" content="Ben Cooley"/>
  <meta name="language" content="English"/>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Try To Login.com</title>
  @Styles.Render("~/Content/css")
  @Scripts.Render("~/bundles/modernizr")
</head>
<body>
<div class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <button type="button"
        class="navbar-toggle"
        Data-toggle="collapse"
        Data-target=".navbar-collapse">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      @Html.ActionLink("Try To Login",
        "Index",
        "Home",
        new { area = "" },
        new { @class = "navbar-brand" })
    </div>
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav">
        <li>@Html.ActionLink("Login", "Index", "Home")</li>
        <li>@Html.ActionLink("About", "About", "Home")</li>
        <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
      </ul>
    </div>
  </div>
</div>
<div class="container body-content">
  @RenderBody()
  <hr/>
  <footer>
    <p>&copy; @DateTime.Now.Year - TryToLogin.com</p>
  </footer>
</div>

@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/bootstrap")
@RenderSection("scripts", false)
</body>
</html>

```

C Login Page Markup

```

@model Thesis.Models.LoginViewModel
@{
    ViewBag.Title = "Try To Log in";
}
<h1>@ViewBag.Title.</h1>
<div class="row">
    <div class="col-md-8">
        <section id="loginForm">
            @using (Html.BeginForm("Index",
                "Home",
                new { ViewBag.ReturnUrl },
                FormMethod.Post,
                new
                {
                    @class = "form-horizontal",
                    role = "form"
                }
            ))
        {
            @Html.AntiForgeryToken()
            <h4>Try to log in.</h4>
            <hr/>
            @Html.ValidationSummary(false, "", new { @class = "text-danger" })
            <div class="form-group">
                @Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })
                <div class="col-md-10">
                    @Html.TextBoxFor(m => m.Email, new
                    {
                        @class = "form-control",
                        v_on = "click: addClick, keydown: addKeyDown, keyup: addKeyUp"
                    })
                    @Html.ValidationMessageFor(m => m.Email,
                        "",
                        new { @class = "text-danger" })
                </div>
            </div>
            <div class="form-group">
                @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })
                <div class="col-md-10">
                    @Html.PasswordFor(m => m.Password, new
                    {
                        @class = "form-control",
                        v_on = "click: addClick, keydown: addKeyDown, keyup: addKeyUp"
                    })
                    @Html.ValidationMessageFor(m => m.Password, "", new
                    {
                        @class = "text-danger"
                    })
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-10 col-md-offset-2">
                    <input class="btn btn-default"
                        name="submit"
                        type="submit"
                        value="Log in"
                        v-on="click: addClick, keydown: addKeyDown, keyup: addKeyUp"/>
                </div>
            </div>
        }
    </div>
}

```

```

<div class="hidden"
  v-repeat="request.formInteraction.keyboardInteractions">
  <input class="form-control"
    name="KeyboardInteraction[{{$index}}].KeyPressed"
    type="hidden"
    value="{{keyPressed}}"/>
  <input class="form-control"
    name="KeyboardInteraction[{{$index}}].AltPressed"
    type="hidden"
    value="{{altPressed || 'false'}}"/>
  <input class="form-control"
    name="KeyboardInteraction[{{$index}}].CtrlPressed"
    type="hidden"
    value="{{ctrlPressed || 'false'}}"/>
  <input class="form-control"
    name="KeyboardInteraction[{{$index}}].ShiftPressed"
    type="hidden"
    value="{{shiftPressed || 'false'}}"/>
  <input class="form-control"
    name="KeyboardInteraction[{{$index}}].TimePressed"
    type="hidden"
    value="{{timePressed}}"/>
  <input class="form-control"
    name="KeyboardInteraction[{{$index}}].TimeReleased"
    type="hidden"
    value="{{timeReleased}}"/>
  <input class="form-control"
    name="KeyboardInteraction[{{$index}}].ElementName"
    type="hidden"
    value="{{elementName}}"/>
</div>
<div class="hidden"
  v-repeat="request.formInteraction.mouseInteractions">
  <input class="form-control"
    name="MouseInteraction[{{$index}}].MouseX"
    type="hidden"
    value="{{mouseX}}"/>
  <input class="form-control"
    name="MouseInteraction[{{$index}}].MouseY"
    type="hidden"
    value="{{mouseY}}"/>
  <input class="form-control"
    name="MouseInteraction[{{$index}}].Button"
    type="hidden"
    value="{{button}}"/>
  <input class="form-control"
    name="MouseInteraction[{{$index}}].TimePressed"
    type="hidden"
    value="{{timePressed}}"/>
  <input class="form-control"
    name="MouseInteraction[{{$index}}].ElementName"
    type="hidden"
    value="{{elementName}}"/>
</div>
}
</section>
</div>
</div>

```

```
@section Scripts {  
    @Scripts.Render("~/bundles/jqueryval")  
    @Scripts.Render("~/bundles/ui")  
}
```

D Non-Intrusive Data Capturing JavaScript

```

$(function () {
  var ui = new Vue({
    el: "#loginForm",
    data: function () {
      return {
        request: {
          formInteraction: {
            keyboardInteractions: [],
            mouseInteractions: []
          }
        },
        keysPressed: {},
        startTime: undefined,
        keysDown: []
      };
    },
    methods: {
      addClick: function (e) {
        var interaction = {
          mouseX: parseInt(e.clientX || e.x),
          mouseY: parseInt(e.clientY || e.y),
          button: e.button,
          timePressed: new Date().toISOString(),
          elementName: e.target.name
        };
        this.request.formInteraction.mouseInteractions.push(interaction);
      },
      addKeyDown: function (e) {
        if (!this.keysPressed[e.keyCode]) {
          var time = new Date();
          this.startTime = this.startTime || time.getTime();
          var interaction = {
            altPressed: e.altKey,
            ctrlPressed: e.ctrlKey,
            shiftPressed: e.shiftKey,
            keyPressed: e.keyCode,
            time: time,
            timeReleased: "",
            elementName: e.target.name
          };
          this.keysPressed[e.keyCode] = interaction;
          interaction.timePressed = this.keysDown.length ?
            -1 * (time.getTime() - this.keysDown[0].time.getTime()) :
            time.getTime() - this.startTime;
          this.keysDown.push(interaction);
          this.request.formInteraction.keyboardInteractions.push(interaction);
        }
      },
    },
  });
}

```

```
addKeyUp: function (e) {  
  var pressedKey = this.keysPressed[e.keyCode];  
  if (pressedKey) {  
    var time = new Date();  
    pressedKey.timeReleased = time.getTime() - this.startTime;  
    var index = this.keysDown.indexOf(pressedKey);  
    index > -1 && this.keysDown.splice(index, 1);  
    this.keysDown.length || (this.startTime = time.getTime());  
    delete this.keysPressed[e.keyCode];  
  }  
}  
});  
}  
);
```

E Interaction Form Controller

```

using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;
using Microsoft.AspNet.Identity.Owin;
using Thesis.Data;
using Thesis.Data.FormIInteraction;
using Thesis.Data.Request;
using Thesis.Models;
using Thesis.Models.Services;

namespace Thesis.Controllers
{
    public class HomeController : Controller
    {
        private readonly IApplicationDbContext _db;
        private ApplicationSignInManager _signInManager;

        public HomeController(IApplicationDbContext db)
        {
            _db = db;
        }

        public ApplicationSignInManager SignInManager
            =>
                _signInManager ??
                (_signInManager =
                    HttpContext.GetOwinContext().Get<ApplicationSignInManager>())
            ;

        public async Task<ActionResult> Index()
        {
            var request = CreateRequestObject(new LoginViewModel(),
                RequestType.Get);
            _db.Requests.Add(request);
            await _db.SaveChangesAsync();
            return View();
        }

        public async Task<ActionResult> About()
        {
            ViewBag.Message = "For a graduate degree thesis";
            var request = CreateRequestObject(new LoginViewModel(),
                RequestType.Get);
            _db.Requests.Add(request);
            await _db.SaveChangesAsync();
            return View();
        }
    }
}

```

```
[ActionName("Index")]
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Login(
    LoginViewModel model,
    string returnUrl)
{
    var request = CreateRequestObject(model);
    request.SignInStatus = SignInStatus.Failure;

    if (!ModelState.IsValid)
    {
        return View("Index");
    }

    var result =
        await
            SignInManager.PasswordSignInAsync(model.Email,
                model.Password,
                model.RememberMe,
                false);

    request.SignInStatus = result;

    _db.Requests.Add(request);
    await _db.SaveChangesAsync();

    switch (result)
    {
        case SignInStatus.Success:
            return View("Success");

        default:
            ModelState.AddModelError("", "Invalid login attempt.");
            return View("Index");
    }
}
```

```

private ApplicationRequestContext CreateRequestObject(
    LoginViewModel model,
    RequestType requestType = RequestType.Post)
{
    var request = new ApplicationRequestContext
    {
        Headers = new List<ApplicationRequestHeader>(),
        Form =
            requestType == RequestType.Post
                ? new List<ApplicationRequestForm>()
                : null
    };

    if (requestType == RequestType.Post)
    {
        foreach (
            var key in
                Request.Form.AllKeys.Where(
                    key =>
                        !key.ToLower().Contains("keyboardinteraction") &&
                        !key.ToLower().Contains("mouseinteraction")))
        {
            request.Form.Add(new ApplicationRequestForm
            {
                Name = key,
                Value = Request.Form[key]
            });
        }
        request.FormInteraction = new FormInteraction
        {
            KeyboardInteractions =
                (model.KeyboardInteraction ??
                 new KeyboardInteraction[] { }).ToList(),
            MouseInteractions =
                (model.MouseInteraction ?? new MouseInteraction[] { })
                .ToList()
        };
    }
    foreach (var key in Request.Headers.AllKeys)
    {
        request.Headers.Add(new ApplicationRequestHeader
        {
            Name = key,
            Value = Request.Headers[key]
        });
    }
    request.IpAddress = Request.UserHostAddress;
    if (Request.UrlReferrer != null)
        request.Referrer = Request.UrlReferrer.AbsoluteUri;
    request.RequestType = requestType;
    if (Request.Url != null) request.Url = Request.Url.AbsoluteUri;
    return request;
}
}
}

```

F Object Model Entities

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Thesis.Data
{
    public class IdentityModel
    {
        public IdentityModel()
        {
            DateCreated = DateTime.UtcNow;
        }

        [Key]
        public int Id { get; set; }

        public DateTime? DateCreated { get; set; }
    }
}

namespace Thesis.Data.Request
{
    public enum RequestType
    {
        Get = 0,
        Post = 1
    }

    public class DictionaryEntity : IdentityModel
    {
        public string Name { get; set; }
        public string Value { get; set; }
    }

    [Table("RequestHeaders")]
    public class ApplicationRequestHeader : DictionaryEntity
    {
    }

    [Table("FormInfo")]
    public class ApplicationRequestForm : DictionaryEntity
    {
    }
}
```

```

using System.Data.Entity;
using Microsoft.AspNet.Identity.EntityFramework;
using Thesis.Data.Authentication;
using Thesis.Data.Request;

namespace Thesis.Data
{
    public class ApplicationDbContext : IdentityDbContext<ApplicationUser>,
        IApplicationDbContext
    {
        public ApplicationDbContext()
            : base("DefaultConnection", false)
        {
        }

        public DbSet<ApplicationRequestContext> Requests { get; set; }

        public static ApplicationDbContext Create()
        {
            return new ApplicationDbContext();
        }
    }
}

using System;
using System.Collections.Generic;
using Microsoft.AspNet.Identity.Owin;
using Thesis.Data.FormInteraction;

namespace Thesis.Data.Request
{
    public class ApplicationRequestContext : IdentityModel
    {
        public ApplicationRequestContext()
        {
            Saved = DateTime.UtcNow;
        }

        public DateTime Saved { get; set; }
        public string IPAddress { get; set; }
        public string Referrer { get; set; }
        public virtual ICollection<ApplicationRequestHeader> Headers { get; set; }
        public virtual ICollection<ApplicationRequestForm> Form { get; set; }
        public string Url { get; set; }
        public RequestType RequestType { get; set; }
        public virtual FormInteraction FormInteraction { get; set; }
        public SignInStatus SignInStatus { get; set; }
    }
}

```

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace Thesis.Data.FormInteraction
{
    public enum MouseButton
    {
        Left = 0,
        Middle = 1,
        Right = 2
    }

    public class FormInteraction : IdentityModel
    {
        public ICollection<KeyboardInteraction> KeyboardInteractions { get; set; }
        public ICollection<MouseInteraction> MouseInteractions { get; set; }
    }

    public class MouseInteraction : IdentityModel
    {
        [Required]
        public DateTime? TimePressed { get; set; }

        [Required]
        public int? MouseX { get; set; }

        [Required]
        public int? MouseY { get; set; }

        [Required]
        public MouseButton? Button { get; set; }

        [StringLength(10), Required]
        public string ElementName { get; set; }
    }

    public class KeyboardInteraction : IdentityModel
    {
        [Required]
        public decimal? TimePressed { get; set; }

        public decimal? TimeReleased { get; set; }

        [Required]
        public int? KeyPressed { get; set; }

        [StringLength(10), Required]
        public string ElementName { get; set; }

        public bool AltPressed { get; set; }

        public bool CtrlPressed { get; set; }

        public bool ShiftPressed { get; set; }
    }
}

```

G Backpropagation Classifier

```

using System;
using AForge.Neuro;
using AForge.Neuro.Learning;

namespace Thesis.Data.Classifiers
{
    public class Backpropagation
    {
        public ActivationNetwork Network { get; }
        public BackPropagationLearning Teacher { get; }

        public Backpropagation()
        {
            Network = new ActivationNetwork(new BipolarSigmoidFunction(1), 10, 10, 1);
            Teacher = new BackPropagationLearning(Network);
        }

        public string[] BackPropogationTrain(double[][] inputData, double[][] outputData, int
iterations)
        {
            bool needStopping = false;
            int iterateCount = 0;
            double error = 0;
            string[] results = new string[2];
            double[][] input = inputData;
            double[][] output = outputData;

            Teacher.LearningRate = .1d;
            while (!needStopping)
            {
                error = Teacher.RunEpoch(input, output);

                if (error == 0)
                {
                    break;
                }
                else if (Math.Round(error, 2) == 0)
                {
                    break;
                }
                else if (iterateCount < iterations)
                {
                    iterateCount++;
                }
                else
                {
                    needStopping = true;
                }
                iterateCount++;
            }
            results[0] = error.ToString();
            results[1] = iterateCount.ToString();
            return results;
        }
    }
}

```

H Support Vector Machine Classifier

```

using System.Collections.Generic;
using Accord.MachineLearning.VectorMachines;
using Accord.MachineLearning.VectorMachines.Learning;
using Accord.Statistics.Kernels;

namespace Thesis.Data.Classifiers
{
    public class KernelSupportVectorMachines
    {
        private readonly KernelSupportVectorMachine _ksvm;

        public KernelSupportVectorMachines()
        {
            _ksvm = new KernelSupportVectorMachine(new Gaussian(100), 10);
        }

        public double Train(double[][] inputs, int[] outputs)
        {
            for (var i = 0; i < outputs.Length; i++)
            {
                outputs[i] = outputs[i] >= 1 ? 1 : -1;
            }

            var smo = new SequentialMinimalOptimization(_ksvm, inputs, outputs)
            {
                UseComplexityHeuristic = true
            };

            return smo.Run(true);
        }

        public IEnumerable<double> Classify(double[][] inputs)
        {
            foreach (var input in inputs)
            {
                yield return _ksvm.Compute(input);
            }
        }
    }
}

```

I Decision Tree Classifier

```

using System.Collections.Generic;
using Accord.MachineLearning.DecisionTrees;
using Accord.MachineLearning.DecisionTrees.Learning;
using Accord.Math;
using AForge;

namespace Thesis.Data.Classifiers
{
    public class DecisionTreeClassifier
    {
        public DecisionTreeClassifier()
        {
            Tree = new DecisionTree(new List<DecisionVariable>(
                new[]
                {
                    DecisionVariable.Continuous("0", new DoubleRange(-10000, 10000)),
                    DecisionVariable.Continuous("1", new DoubleRange(-10000, 10000)),
                    DecisionVariable.Continuous("2", new DoubleRange(-10000, 10000)),
                    DecisionVariable.Continuous("3", new DoubleRange(-10000, 10000)),
                    DecisionVariable.Continuous("4", new DoubleRange(-10000, 10000)),
                    DecisionVariable.Continuous("5", new DoubleRange(-10000, 10000)),
                    DecisionVariable.Continuous("6", new DoubleRange(-10000, 10000)),
                    DecisionVariable.Continuous("7", new DoubleRange(-10000, 10000)),
                    DecisionVariable.Continuous("8", new DoubleRange(-10000, 10000)),
                    DecisionVariable.Continuous("9", new DoubleRange(-10000, 10000))
                }, 2);

            Teacher = new C45Learning(Tree);
        }

        C45Learning Teacher { get; }
        public DecisionTree Tree { get; }

        public double Train(double[][] inputs, int[] outputs)
        {
            // The C4.5 algorithm expects the class labels to
            // range from 0 to k, so we convert -1 to be zero:
            outputs = outputs.Apply(x => x = (x <= 0 ? 0 : 1));
            var error = Teacher.Run(inputs, outputs);
            return error;
        }

        public IEnumerable<int> Classify(double[][] inputs)
        {
            return inputs.Apply(x => Tree.Compute(x));
        }
    }
}

```

J Dissimilarity Comparison Classifier

```

using System;
using System.Collections.Generic;
using System.Linq;
using Accord.Statistics;

namespace Thesis.Data.Classifiers
{
    public static class DissimilarityComparison
    {
        public static bool AuthenticateUser(
            double[][] knownInputs,
            double[] profileToValidate,
            double sigma)
        {
            var length = knownInputs[0].Count();
            if (knownInputs.Any(profile => length != profile.Count()))
            {
                throw new ArgumentOutOfRangeException(nameof(knownInputs));
            }
            if (profileToValidate.Count() != length)
                return false;
            var masterTrajectoryProfile =
                CreateMasterTrajectoryProfile(knownInputs, length);

            var dissimilarityProfile =
                GetDissimilarityProfile(knownInputs,
                    masterTrajectoryProfile,
                    length);
            var knownAverageDissimilarity = dissimilarityProfile.Average();
            var knownStandardDeviationDissimilarity =
                dissimilarityProfile.StandardDeviation();

            return IsProfileValid(
                masterTrajectoryProfile,
                profileToValidate,
                knownAverageDissimilarity,
                knownStandardDeviationDissimilarity,
                sigma);
        }

        public static bool IsProfileValid(
            double[] masterTrajectoryProfile,
            double[] profileToValidate,
            double knownAverageDissimilarity,
            double knownStandardDeviationDissimilarity,
            double sigma)
        {
            var profileToCheckDissimilarity =
                GetDissimilarity(masterTrajectoryProfile, profileToValidate);
            return profileToCheckDissimilarity <
                knownAverageDissimilarity +
                sigma*knownStandardDeviationDissimilarity;
        }
    }
}

```

```

public static double[] GetDissimilarityProfile(
    double[][] knownInputs,
    IReadOnlyList<double> masterTrajectoryProfile,
    int length)
{
    var knownInputsDissimilarityForAverage = new double[length/2];

    for (var i = 0; i < length; i += 2)
    {
        foreach (var input in knownInputs)
        {
            knownInputsDissimilarityForAverage[i/2] += Math.Sqrt(
                Math.Pow(masterTrajectoryProfile[i] - input[i], 2) +
                Math.Pow(masterTrajectoryProfile[i + 1] - input[i + 1],
                    2)
            );
        }
    }
    return knownInputsDissimilarityForAverage;
}

private static double GetDissimilarity(
    IReadOnlyList<double> masterTrajectoryProfile,
    double[] localInput)
{
    var dissimilarityLocal = 0d;
    var length = localInput.Count();
    for (var j = 0; j < length; j += 2)
    {
        dissimilarityLocal +=
            Math.Sqrt(
                Math.Pow(masterTrajectoryProfile[j] - localInput[j], 2) +
                Math.Pow(
                    masterTrajectoryProfile[j + 1] - localInput[j + 1],
                    2));
    }
    return dissimilarityLocal;
}

public static double[] CreateMasterTrajectoryProfile(
    double[][] knownInputs,
    int length)
{
    var masterTrajectoryProfile = new double[length];

    var knownInputCount = knownInputs.Count();
    for (var i = 0; i < length; i++)
    {
        var sum = 0d;
        for (var j = 0; j < knownInputCount; j++)
        {
            sum += knownInputs[j][i];
        }
        masterTrajectoryProfile[i] = sum/knownInputCount;
    }
    return masterTrajectoryProfile;
}
}}}

```

K Common Experiment Variables and Functions

```

using System;
using System.Collections.Generic;
using System.Linq;

namespace Thesis.Tests.Classifications
{
    public static class Constants
    {
        {
            public static readonly double[] One = { 112d, 0d, 144d, -32d, 408d, 312d, 144d, 40d,
192d, 88d };
            public static readonly double[] Two = { 96d, 0d, 184d, 87d, 456d, 376d, 152d, 56d,
168d, 96d };
            public static readonly double[] Three = { 152d, 0d, 191d, -87d, 393d, 304d, 143d, 40d,
193d, 89d };
            public static readonly double[] Four = { 87d, 0d, 337d, 225d, 353d, 272d, 151d, 46d,
183d, 64d };
            public static readonly double[] Five = { 121d, 0d, 144d, 32d, 311d, 231d, 153d, 48d,
187d, 88d };
            public static readonly double[] Six = { 112d, 0d, 144d, -32d, 408d, 312d, 144d, 40d,
192d, 88d };

            public static readonly double[] Fake = { 112d, 0d, 168d, -48d, 104d, 24d, 168d, 55d,
136d, 56d };
            public static readonly double[] Real = { 152.00d, 0.00d, 184.00d, -72.00d, 368.00d,
296.00d, 152.00d, 49.00d, 137.00d, 41.00d };
            public static readonly double[] No = { 0d, 0d, 0d, 0d, 0d, 0d, 0d, 0d, 0d, 0d };

            public static double[][] Inputs()
            {
                return new[]
                {
                    One,
                    Two,
                    Three,
                    Four,
                    Five,
                    Six,
                    Fake
                };
            }

            public static double[][] InputsNoFake()
            {
                return new[]
                {
                    One,
                    Two,
                    Three,
                    Four,
                    Five,
                    Six
                };
            }
        }
    }
}

```

```
public static double[][] GoodInputs()
{
    return new[]
    {
        One,
        Two,
        Three,
        Five,
        Six
    };
}
public static double[][] OutputsDouble()
{
    var output =
        new List<double[]>(new[]
        {
            new double[] { 1 },
            new double[] { 0 }
        });

    return output.ToArray();
}
public static double[][] OutputsDoubleNoFake()
{
    var output =
        new List<double[]>(new[]
        {
            new double[] { 1 },
            new double[] { 1 }
        });

    return output.ToArray();
}
public static double[][] GoodOutputs()
{
    var output =
        new List<double[]>(new[]
        {
            new double[] { 1 },
            new double[] { 1 }
        });

    return output.ToArray();
}
```

```

public static int[] GoodOutputsInt()
{
    return new[] { 1, 1, 1, 1, 1 };
}
public static int[] OutputsInt()
{
    return new[] { 1, 1, 1, 1, 1, 1, 0 };
}

public static int[] OutputsIntNoFake()
{
    return new[] { 1, 1, 1, 1, 1, 1 };
}

public static double[][] GenerateBadData(int cells, int rows, double max)
{
    var random = new Random();
    return
        Enumerable.Range(0, rows)
            .Select(
                x =>
                    Enumerable.Range(0, cells)
                        .Select(
                            y =>
                                (y == 1) ? 0 :
                                (y % 2 == 1
                                 ? (random.NextDouble() > .5d
                                  ? -1
                                  : 1)
                                : 1) *
                                random.NextDouble() * max)
                        .ToArray()
                    .ToArray();
}

public static int[][] GenerateBadData(int cells, int rows, int max)
{
    var random = new Random();
    return
        Enumerable.Range(0, rows)
            .Select(
                x =>
                    Enumerable.Range(0, cells)
                        .Select(
                            y =>
                                (y == 1) ? 0 :
                                (y % 2 == 1
                                 ? (random.NextDouble() > .5d
                                  ? -1
                                  : 1)
                                : 1) *
                                random.Next(max))
                        .ToArray()
                    .ToArray();
}
}
}
}

```

L Backpropagation Test

```

using System.Collections.Generic;
using System.Linq;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Thesis.Data.Classifiers;

namespace Thesis.Tests.Classifications
{
    [TestClass]
    public class BackPropogationClassifiers
    {
        [TestMethod]
        public void CanTrain()
        {
            var p = new Backpropagation();
            var arr = new List<double[]>
            {
                Constants.One,
                Constants.Two,
                Constants.Three,
                Constants.Four,
                Constants.Five,
                Constants.Six,
                Constants.Fake,
                Constants.Real
            };
            arr.AddRange(Constants.GenerateBadData(10, 1000, 500d));

            var output =
                new List<double[]>(new[]
                {
                    new double[] { 1 },
                    new double[] { 0 },
                    new double[] { 1 }
                });
            Enumerable.Range(0, 1000).ToList()
                .ForEach(x => output.Add(new double[] { 0 }));

            p.BackPropogationTrain(arr.ToArray(), output.ToArray(), 1000);
            var check1 = p.Network.Compute(Constants.One)[0];
            var check2 = p.Network.Compute(Constants.Two)[0];
            var check3 = p.Network.Compute(Constants.Three)[0];
            var check4 = p.Network.Compute(Constants.Four)[0];
            var check5 = p.Network.Compute(Constants.Five)[0];
            var check6 = p.Network.Compute(Constants.Six)[0];
            var realCheck = p.Network.Compute(Constants.Real)[0];
            var fakeCheck = p.Network.Compute(Constants.Fake)[0];
            var noCheck = p.Network.Compute(Constants.No)[0];
        }
    }
}

```

M Support Vector Machine Tests

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Thesis.Data.Classifiers;

namespace Thesis.Tests.Classifications
{
    [TestClass]
    public class SupportVectorMachineTest
    {
        private readonly KernelSupportVectorMachines _svm;
        private int _falseRejections;
        private int _max;
        private int _min;
        private readonly List<int> _numberOfFalseAcceptances;

        public SupportVectorMachineTest()
        {
            _svm = new KernelSupportVectorMachines();
            _numberOfFalseAcceptances = new List<int>();
        }

        public TestContext TestContext { get; set; }

        private void CanTrain(int numberOfRandoms = 0, int iterations = 1000)
        {
            for (var i = 0; i < iterations; i++)
            {
                Train(numberOfRandoms);
            }
            TestContext.WriteLine(
                "{0} - {1}. False Rejections: {2}. Avg False Acceptance: {3}. Total FAs: {4}",
                _min,
                _max,
                _falseRejections,
                _numberOfFalseAcceptances.Average(),
                _numberOfFalseAcceptances.Sum());
        }

        public void Train(int numberOfRandoms = 0)
        {
            var p = _svm;
            var inputs = Constants.Inputs().ToList();
            var randoms = Constants.GenerateBadData(10, numberOfRandoms, 500);

            var outputs = Constants.OutputsInt().ToList();
            outputs.AddRange(randoms.Select(x => 0));

            var toClassify = new List<double[]>
            {
                Constants.Real,
                Constants.Fake,
                Constants.No
            };
            toClassify.AddRange(inputs);
        }
    }
}

```

```

inputs.AddRange(randoms.Select(nums => nums.Select(y => (double) y).ToArray()));
var err = p.Train(inputs.ToArray(), outputs.ToArray());
var falseGen = Constants.GenerateBadData(10, 10000, 500)
    .Select(nums => nums.Select(y => (double) y).ToArray());
var errors = p.Classify(falseGen.ToArray()).ToList();
var errors2 = p.Classify(toClassify.ToArray()).ToList();

var errCount = errors.Count(x => x >= 0);

_max = Math.Max(_max, errCount);
_min = Math.Min(_min, errCount);

_numberOfFalseAcceptances.Add(errCount);

var falseRejections = errors2.Count(x => x >= 0);
if (falseRejections != 7)
{
    _falseRejections++;
}
}

[TestMethod]
public void CanTrainNoRandoms()
{
    CanTrain(0);
}

[TestMethod]
public void CanTrain10Randoms()
{
    CanTrain(10);
}

[TestMethod]
public void CanTrain100Randoms()
{
    CanTrain(100);
}

[TestMethod]
public void CanTrain1000Randoms()
{
    CanTrain(1000);
}

[TestMethod]
public void CanTrain5000Randoms()
{
    CanTrain(5000);
}
}
}

```

N Decision Tree Tests

```

using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Thesis.Data.Classifiers;

namespace Thesis.Tests.Classifications
{
    [TestClass]
    public class DecisionTreeTest
    {
        private readonly List<int> _numberOfFalseAcceptances;

        private int _falseRejections;
        private int totalComparisons;

        private int _max;

        private int _min = int.MaxValue;

        public DecisionTreeTest()
        {
            _numberOfFalseAcceptances = new List<int>();
        }

        public TestContext TestContext { get; set; }

        [TestMethod]
        public void CanTrain()
        {
            var p = new DecisionTreeClassifier();
            var inputs = Constants.Inputs();
            var outputs = Constants.OutputsInt();
            var toClassify = new List<double[]>
            {
                Constants.Real,
                Constants.Fake,
                Constants.No
            };
            toClassify.AddRange(inputs);
            p.Train(inputs, outputs);
            p.Classify(toClassify.ToArray());
        }

        [TestMethod]
        public void CanClassify()
        {
            var p = new DecisionTreeClassifier();
            var inputs = Constants.Inputs().ToList();
            var randoms = Constants.GenerateBadData(10, 10, 500);
            inputs.AddRange(
                randoms.Select(nums => nums.Select(y => (double) y).ToArray()));

            var outputs = Constants.OutputsInt().ToList();
            outputs.AddRange(randoms.Select(x => 0));
        }
    }
}

```

```

p.Train(inputs.ToArray(), outputs.ToArray());

var falseGen =
    Constants.GenerateBadData(10, 10000, 500)
        .Select(nums => nums.Select(y => (double) y).ToArray());
p.Classify(falseGen.ToArray());
}

[TestMethod]
public void CanClassifyWithRandoms(int numberOfRandoms)
{
    var p = new DecisionTreeClassifier();
    var inputs = Constants.Inputs().ToList();
    var randoms = Constants.GenerateBadData(10, numberOfRandoms, 500);

    var outputs = Constants.OutputsInt().ToList();
    outputs.AddRange(randoms.Select(x => 0));

    var toClassify = new List<double[]>
    {
        Constants.Real,
        Constants.Fake,
        Constants.No
    };
    toClassify.AddRange(inputs);

    inputs.AddRange(
        randoms.Select(nums => nums.Select(y => (double) y).ToArray()));
    p.Train(inputs.ToArray(), outputs.ToArray());

    var falseGen =
        Constants.GenerateBadData(10, 10000, 500)
            .Select(nums => nums.Select(y => (double) y).ToArray());
    var errors = p.Classify(falseGen.ToArray());
    var errors2 = p.Classify(toClassify.ToArray());

    var errCount = errors.Count(x => x == 1);

    _max = Math.Max(_max, errCount);
    _min = Math.Min(_min, errCount);

    _numberOfFalseAcceptances.Add(errCount);

    totalComparisons += falseGen.Count() + inputs.Count();

    var falseRejections = errors2.Count(x => x == 1);
    if (falseRejections != 7)
    {
        _falseRejections++;
    }
}

```

```

private void LoopGeneratedTests(int iterations, int numberOfRandoms)
{
    for (var i = 0; i < iterations; i++)
    {
        CanClassifyWithRandoms(numberOfRandoms);
    }
    TestContext.WriteLine(
        "{0} - {1} of {5}. False Rejections: {2}. Avg False Acceptance: {3}.
False Acceptances: {4}",
        _min,
        _max,
        _falseRejections,
        _numberOfFalseAcceptances.Average(),
        _numberOfFalseAcceptances.Sum(),
        totalComparisons);
}

[TestMethod]
public void LoopGeneratedTestsNoRandoms()
{
    LoopGeneratedTests(1000, 0);
}

[TestMethod]
public void LoopGeneratedTests10Randoms()
{
    LoopGeneratedTests(1000, 10);
}

[TestMethod]
public void LoopGeneratedTests100Randoms()
{
    LoopGeneratedTests(1000, 100);
}

[TestMethod]
public void LoopGeneratedTests1000Randoms()
{
    LoopGeneratedTests(1000, 1000);
}

[TestMethod]
public void LoopGeneratedTests5000Randoms()
{
    LoopGeneratedTests(1000, 5000);
}
}
}

```

O Dissimilarity Comparison Tests

```

using System;
using System.Linq;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Thesis.Data.Classifiers;
using Accord.Statistics;
using Thesis.Data.ReadFiles;

namespace Thesis.Tests.Classifications
{
    [TestClass]
    public class DissimilarityTest
    {
        int _numberOfFalseAcceptances;
        int _numberOfFalseRejections;

        public TestContext TestContext { get; set; }

        [TestMethod]
        public void CanDetermineDissimilarity()
        {
            var fake =
DissimilarityComparison.AuthenticateUser(Constants.InputsNoFake(), Constants.Fake, 1d);
            var real =
DissimilarityComparison.AuthenticateUser(Constants.InputsNoFake(), Constants.Real, 1d);
            var no =
DissimilarityComparison.AuthenticateUser(Constants.InputsNoFake(), Constants.No, 1d);
            Assert.IsTrue(real);
            Assert.IsFalse(fake);
            Assert.IsFalse(no);
        }

        [TestMethod]
        public void DissimilarityForRandoms()
        {
            var numberOfProfiles = 1000000;
            var knownInputs = Constants.GoodInputs();
            var length = knownInputs[0].Count();
            var masterTrajectoryProfile =
DissimilarityComparison.CreateMasterTrajectoryProfile(knownInputs, length);
            var dissimilarityProfile =
DissimilarityComparison.GetDissimilarityProfile(knownInputs, masterTrajectoryProfile,
length);
            var sigma = 1d;
            var knownAverageDissimilarity = dissimilarityProfile.Average();
            var knownStandardDeviationDissimilarity =
dissimilarityProfile.StandardDeviation();

            foreach (var known in knownInputs)
            {
                Assert.IsTrue(DissimilarityComparison.IsProfileValid(
                    masterTrajectoryProfile,
                    known,
                    knownAverageDissimilarity,
                    knownStandardDeviationDissimilarity,
                    sigma));
            }
        }
    }
}

```

```

Assert.IsTrue(
    DissimilarityComparison.IsProfileValid(
        masterTrajectoryProfile,
        Constants.Real,
        knownAverageDissimilarity,
        knownStandardDeviationDissimilarity,
        sigma));
Assert.IsFalse(
    DissimilarityComparison.IsProfileValid(
        masterTrajectoryProfile,
        Constants.Fake,
        knownAverageDissimilarity,
        knownStandardDeviationDissimilarity,
        sigma));
Assert.IsFalse(
    DissimilarityComparison.IsProfileValid(
        masterTrajectoryProfile,
        Constants.No,
        knownAverageDissimilarity,
        knownStandardDeviationDissimilarity,
        sigma));

RunAgainstInputs(masterTrajectoryProfile,
    knownAverageDissimilarity,
    knownStandardDeviationDissimilarity,
    sigma,
    numberOfProfiles);
TestContext.WriteLine("False Acceptances: {0}. FAR: {1}%",
    _numberOfFalseAcceptances,
    ((_numberOfFalseAcceptances / (double)numberOfProfiles) * 100));
}

private void RunAgainstInputs(
    double[] masterTrajectoryProfile,
    double knownAverageDissimilarity,
    double knownStandardDeviationDissimilarity,
    double sigma,
    int numberOfProfiles)
{
    var dataToTest = Constants.GenerateBadData(10, numberOfProfiles, 250d);
    var datas = dataToTest.Take(10).Select(x => string.Join(", ", x.Select(y =>
(y/1).ToString())));

    for (var i = 0; i < numberOfProfiles; i++)
    {
        if (
            DissimilarityComparison.IsProfileValid(
                masterTrajectoryProfile,
                dataToTest[i],
                knownAverageDissimilarity,
                knownStandardDeviationDissimilarity,
                sigma)
        )
        {
            _numberOfFalseAcceptances++;
        }
    }
}
}

```

```

[TestMethod]
public void DissimilarityForRandomsLoop()
{
    var numberOfProfiles = 1000000;
    var knownInputs = Constants.GoodInputs();
    var length = knownInputs[0].Count();
    var masterTrajectoryProfile =
DissimilarityComparison.CreateMasterTrajectoryProfile(knownInputs, length);
    var dissimilarityProfile =
DissimilarityComparison.GetDissimilarityProfile(knownInputs, masterTrajectoryProfile, length);
    var sigma = 1d;
    var knownAverageDissimilarity = dissimilarityProfile.Average();
    var knownStandardDeviationDissimilarity =
dissimilarityProfile.StandardDeviation();

    for (int i = 0; i < 100; i++)
    {
        RunAgainstInputs(masterTrajectoryProfile, knownAverageDissimilarity,
knownStandardDeviationDissimilarity, sigma, numberOfProfiles);
    }
    TestContext.WriteLine("False Acceptances: {0}. FAR: {1}%",
_numberOfFalseAcceptances, ((_numberOfFalseAcceptances / (double)numberOfProfiles) * 100));
}

[TestMethod]
public void TestFile()
{
    var items =
    CMUData.ReadFromFileData(
        CSVReader.ReadFile(@"DSL-StrongPasswordData.csv"));
    var inputGroups = items.GroupBy(x => new { x.Person, x.SessionIndex });
    var length = 20;
    TestContext.WriteLine("Person,Session,FalseAcceptances,FAR,FalseRejections,FRR");
    foreach (var inputGroup in inputGroups)
    {
        var localFalseAcceptance = 0;
        var localFalseRejection = 0;
        var knownInputs = inputGroup.Select(x => x.Data).Skip(45).ToArray();
        var userInputs = inputGroup.Select(x => x.Data).Take(45).ToArray();
        var masterTrajectoryProfile =
DissimilarityComparison.CreateMasterTrajectoryProfile(knownInputs, length);
        var dissimilarityProfile =
DissimilarityComparison.GetDissimilarityProfile(knownInputs, masterTrajectoryProfile, length);
        var sigma = 2d;
        var knownAverageDissimilarity = dissimilarityProfile.Average();
        var knownStandardDeviationDissimilarity =
dissimilarityProfile.StandardDeviation();

        var notCurrentPerson = inputGroups.Where(x => x.Key.Person !=
inputGroup.Key.Person);
        var dataToTest = notCurrentPerson.SelectMany(x => x).Select(x =>
x.Data).ToArray();
        var numberOfProfiles = dataToTest.Length;

```


P Database Script

```

USE [Thesis]
GO
/***** Object: Table [dbo].[ApplicationRequestContexts]    Script Date: 2/17/2016
10:11:45 PM *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[ApplicationRequestContexts](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Saved] [datetime] NOT NULL,
    [IpAddress] [nvarchar](max) NULL,
    [Referrer] [nvarchar](max) NULL,
    [Url] [nvarchar](max) NULL,
    [RequestType] [int] NOT NULL,
    [FormInteraction_Id] [int] NULL,
    [DateCreated] [datetime] NULL DEFAULT ('1900-01-01T00:00:00.000'),
    [SignInStatus] [int] NOT NULL DEFAULT ((0)),
    CONSTRAINT [PK_dbo.ApplicationRequestContexts] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
/***** Object: Table [dbo].[FormInfo]    Script Date: 2/17/2016 10:11:45 PM *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FormInfo](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](max) NULL,
    [Value] [nvarchar](max) NULL,
    [ApplicationRequestContext_Id] [int] NULL,
    [DateCreated] [datetime] NULL DEFAULT ('1900-01-01T00:00:00.000'),
    CONSTRAINT [PK_dbo.FormInfo] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO/***** Object: Table [dbo].[FormInteractions]    Script Date: 2/17/2016 10:11:45 PM
*****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FormInteractions](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [DateCreated] [datetime] NULL DEFAULT ('1900-01-01T00:00:00.000'),
    CONSTRAINT [PK_dbo.FormInteractions] PRIMARY KEY CLUSTERED
(

```

```

        [Id] ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
    ) ON [PRIMARY]

GO
/***** Object: Table [dbo].[KeyboardInteractions]    Script Date: 2/17/2016 10:11:45 PM
*****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[KeyboardInteractions](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [KeyPressed] [int] NOT NULL,
    [FormInteraction_Id] [int] NULL,
    [ElementName] [nvarchar](10) NOT NULL,
    [AltPressed] [bit] NOT NULL DEFAULT ((0)),
    [CtrlPressed] [bit] NOT NULL DEFAULT ((0)),
    [ShiftPressed] [bit] NOT NULL DEFAULT ((0)),
    [DateCreated] [datetime] NULL DEFAULT ('1900-01-01T00:00:00.000'),
    [TimeReleased] [decimal](18, 2) NULL DEFAULT ((0)),
    [TimePressed] [decimal](18, 2) NOT NULL DEFAULT ((0)),
    CONSTRAINT [PK_dbo.KeyboardInteractions] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
/***** Object: Table [dbo].[MouseInteractions]    Script Date: 2/17/2016 10:11:45 PM
*****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[MouseInteractions](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [TimePressed] [datetime] NOT NULL,
    [MouseX] [int] NOT NULL,
    [MouseY] [int] NOT NULL,
    [FormInteraction_Id] [int] NULL,
    [Button] [int] NOT NULL DEFAULT ((0)),
    [ElementName] [nvarchar](10) NOT NULL,
    [DateCreated] [datetime] NULL DEFAULT ('1900-01-01T00:00:00.000'),
    CONSTRAINT [PK_dbo.MouseInteractions] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
/***** Object: Table [dbo].[RequestHeaders]    Script Date: 2/17/2016 10:11:45 PM
*****/
SET ANSI_NULLS ON
GO

```

```

SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[RequestHeaders](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](max) NULL,
    [Value] [nvarchar](max) NULL,
    [ApplicationRequestContext_Id] [int] NULL,
    [DateCreated] [datetime] NULL DEFAULT ('1900-01-01T00:00:00.000'),
    CONSTRAINT [PK_dbo.RequestHeaders] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

GO
ALTER TABLE [dbo].[ApplicationRequestContexts] WITH CHECK ADD CONSTRAINT
[FK_dbo.ApplicationRequestContexts_dbo.FormInteractions_FormInteraction_Id] FOREIGN
KEY([FormInteraction_Id])
REFERENCES [dbo].[FormInteractions] ([Id])
GO
ALTER TABLE [dbo].[ApplicationRequestContexts] CHECK CONSTRAINT
[FK_dbo.ApplicationRequestContexts_dbo.FormInteractions_FormInteraction_Id]
GO
ALTER TABLE [dbo].[FormInfo] WITH CHECK ADD CONSTRAINT
[FK_dbo.FormInfo_dbo.ApplicationRequestContexts_ApplicationRequestContext_Id] FOREIGN
KEY([ApplicationRequestContext_Id])
REFERENCES [dbo].[ApplicationRequestContexts] ([Id])
GO
ALTER TABLE [dbo].[FormInfo] CHECK CONSTRAINT
[FK_dbo.FormInfo_dbo.ApplicationRequestContexts_ApplicationRequestContext_Id]
GO
ALTER TABLE [dbo].[KeyboardInteractions] WITH CHECK ADD CONSTRAINT
[FK_dbo.KeyboardInteractions_dbo.FormInteractions_FormInteraction_Id] FOREIGN
KEY([FormInteraction_Id])
REFERENCES [dbo].[FormInteractions] ([Id])
GO
ALTER TABLE [dbo].[KeyboardInteractions] CHECK CONSTRAINT
[FK_dbo.KeyboardInteractions_dbo.FormInteractions_FormInteraction_Id]
GO
ALTER TABLE [dbo].[MouseInteractions] WITH CHECK ADD CONSTRAINT
[FK_dbo.MouseInteractions_dbo.FormInteractions_FormInteraction_Id] FOREIGN
KEY([FormInteraction_Id])
REFERENCES [dbo].[FormInteractions] ([Id])
GO
ALTER TABLE [dbo].[MouseInteractions] CHECK CONSTRAINT
[FK_dbo.MouseInteractions_dbo.FormInteractions_FormInteraction_Id]
GO
ALTER TABLE [dbo].[RequestHeaders] WITH CHECK ADD CONSTRAINT
[FK_dbo.RequestHeaders_dbo.ApplicationRequestContexts_ApplicationRequestContext_Id]
FOREIGN KEY([ApplicationRequestContext_Id])
REFERENCES [dbo].[ApplicationRequestContexts] ([Id])
GO
ALTER TABLE [dbo].[RequestHeaders] CHECK CONSTRAINT
[FK_dbo.RequestHeaders_dbo.ApplicationRequestContexts_ApplicationRequestContext_Id]
GO

```