NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

CHARACTERIZING INFORMATION LEAKAGE FROM USER DEFINED INPUT EVENTS

by

John P. Burns

June 2020

Thesis Advisor:	Vinnie Monaco
Second Reader:	Theodore D. Huffmire

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2020	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE CHARACTERIZING INFORMATION LEAKAGE FROM USER DEFINED INPUT EVENTS	**5. FUNDING NUMBERS**
6. AUTHOR(S) John P. Burns	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.	**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

Computer operators cannot comprehend the amount of identifying information that a single machine regularly broadcasts over the internet. These seemingly benign bits of information have the ability to create unintended and irreparable security and privacy consequences, such as the ability to remotely profile and fingerprint devices. In this work, we explore the privacy implications of keyboard event timestamps recorded in a web browser. We use time controlled keyboard inputs to characterize information leakage from a host computer connected to the internet. The study takes the user out of the picture and focuses on the hardware. This area of focus is significant because keyboard timestamps are not guarded by permissions and remain ubiquitous across devices connected to the internet. Timestamp analysis betrays a non-trivial amount of information about the host machine. This study characterizes this passive leakage in the examination and testing of nine typical personal computers. The results yielded 100% accuracy in operating system profiling and finds that seemingly identical computers can be fingerprinted with almost 90% accuracy.

14. SUBJECT TERMS keyboard, clock skew, security, privacy	**15. NUMBER OF PAGES** 51
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

CHARACTERIZING INFORMATION LEAKAGE FROM USER DEFINED INPUT EVENTS

John P. Burns
Lieutenant, United States Navy
BS, Purdue University, 2013

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2020

Approved by: Vinnie Monaco
Advisor

Theodore D. Huffmire
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Computer operators cannot comprehend the amount of identifying information that a single machine regularly broadcasts over the internet. These seemingly benign bits of information have the ability to create unintended and irreparable security and privacy consequences, such as the ability to remotely profile and fingerprint devices. In this work, we explore the privacy implications of keyboard event timestamps recorded in a web browser. We use time controlled keyboard inputs to characterize information leakage from a host computer connected to the internet. The study takes the user out of the picture and focuses on the hardware. This area of focus is significant because keyboard timestamps are not guarded by permissions and remain ubiquitous across devices connected to the internet. Timestamp analysis betrays a non-trivial amount of information about the host machine. This study characterizes this passive leakage in the examination and testing of nine typical personal computers. The results yielded 100% accuracy in operating system profiling and finds that seemingly identical computers can be fingerprinted with almost 90% accuracy.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

Acknowledgments

I would like to thank Samantha Fuhrman Burns, without whose support this effort would not be possible.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Internet privacy is not a one-size-fits-all approach. It is a delicate balance of maintaining connectivity without revealing too much information. Sometimes the information that is broadcast out is of little consequence, used strictly to allow machines to reliably communicate together. Sometimes that information has the ability to reveal more information than was originally intended. This is the scope of many previous works, as well as this thesis: searching for hidden clues about a machine that can be used for both defending against botnets and fraud and for more malicious reasons, such as remotely tracking users over the internet.

Clock skew fingerprinting is a technique by which unique differences in hardware and manufacturing processes can be measured and used to identify individual machines. Several methods exist to gather information about machines based on their clocks. Information about the clock may be garnered remotely from timestamps generated from network packets. This information can be used to determine the clock skew, a measure of how much the clock shifts over a period of time. This skew is affected by ambient temperature and the amount of work the machine is doing. The way these timestamps are reported is also dependent largely on the host's Operating System (OS) and software environment. This thesis aims to characterize software-based sources of information leakage through event timestamps.

The keyboard is a unique vector for exploit: nearly every computer has one and since it is a primary interface between human and machine, its use is ubiquitous. Surprisingly, even though the keyboard use is so widespread it is rarely considered a privacy risk. The event timestamps generated by the keyboard can typically be collected with no special permission. Perhaps this is the case because they were largely thought to be inconsequential.

When a user presses a key on a keyboard, it raises a hardware interrupt on the Central Processing Unit (CPU). This interrupt is answered by the OS according to the scheduler. The scheduler operates according to an on-board oscillator, whose speed is dependent on several factors, including manufacturer and operating system settings. The goal is to use these user defined events to find more information about the computer's on-board clock and

demonstrate how this clock information can expose new threats.

One threat is remote user tracking that relies on generating network traffic that is highly correlated to the timing of the machine's input events. This is a passive attack, meaning the attacker doesn't have to act to gain information; the attacker merely has to listen. An example of this would be when a user types a query into a web search engine. This threat assumes that the host of the web search engine can see the timing of the packets sent from the host's machine. Such network traffic can be generated by auto fill features, where each keystroke sends a packet to the search engine. The host uses these timestamps to fingerprint the user, and in this way they are able to remember the user each time they visit the web page. After typing in their query, the host is able to compare this data to past collected data and know with a degree certainty who this user is. The web server can then use this information, combined with the user's previous search terms, to offer them focused ads.

1.1 Contributions

This thesis includes research and experimentation that is conducted on multiple identical Windows and Linux machines running the same OS and settings to explore the differences in system clocks as measured through keyboard event timestamps. Timestamps on each machine are recorded within a web browser, (e.g., Google Chrome). To control for differences in typing behavior, a system is developed to generate identical timing data. This is done by driving a keyboard independently and transparently to the machine being tested. This work also contributes multiple models to profile OS family, and fingerprint based on dominant frequencies.

1.2 Motivation

The techniques developed here have offensive and defensive applications. Offensively, potential vulnerabilities enumerated can be exploited by an actor seeking to undermine the online privacy of users. This is commonplace by businesses seeking to offer tailored content to the visitors of their websites, and could also be used by an attacker with more malevolent aspirations. This technique reduces the anonymity pool of users visiting a web page and can be combined with additional tracking tools to fingerprint users at finer granularity. If this is utilized by a company offering a third party add-on to a website, the company could

track users across a range of websites in order to create a digital profile and piece together sensitive information about individuals.

The defensive rational for interest in this subject is that the information gathered about a machine from its organic timing can be used to detect fraud. For instance, if an attacker is spoofing another machine's information (e.g., IP address spoofing), timestamps may not support the information he/she is advertising, which could raise concerns to security systems. This could also assist in aliasing a user attempting to appear like multiple different users. An example of this would be an individual attempting to log into multiple bank accounts from the same computer, even without fingerprinting, portraying a similar machine set up would look suspicious amongst the diversity of machines on the web.

1.3 Thesis Organization

Chapter 2 provides background for the reader to better understand the methodology and results that will be presented in later chapters. It includes an overview of the importance and impact of device fingerprinting as well as common approaches to achieving this. Background information on keyboard operations are explored to allow the reader to understand the experiment, as well as the importance of the CPU clock and how variations in the clock and its environment affects clock skew.

Chapter 3 outlines the specific methodology that is used to conduct research. The experiment is described, along with a detailed explanation of the techniques used to collect, process, and examine the data results from the experiment. Competing techniques are compared to give the reader an idea of the potential applications to future work.

Chapter 4 delves into the results of the experiments conducted. Each of the experiment outcomes are examined, including the accuracy and any findings that contribute to the results. This includes tables and confusion matrices.

Chapter 5 provides the conclusion of this work. A discussion of the implications of the findings is offered, as well as possible defenses. Future work that could further this topic is suggested.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Background

2.1 Keyboard Events

When a USB keyboard key is pressed or released, several things occur sequentially that add to the overall latency, or delay, of a character appearing on the user's screen. The micro-controller on board a commercial keyboard continually scans the matrix (i.e., an array of key circuits) at a set frequency looking for a key press. This is called matrix scanning. In turn, the keyboard's micro-controller is polled by Universal Host Controller Interface (UHCI), typically at a rate of 125Hz, which is the USB polling rate for low-speed devices OS [1], [2]. From there, the signal for the key is sent to the computer's OS. These two actions, the polling from the computer to the keyboard and scanning of the micro-controller introduce latency from the key press to even the time the signal is first seen by the computer that it is connected to. However, since the frequencies are fixed, the delay is a predictable maximum. Once the signal from the keyboard is in the computer, it causes a hardware interrupt to be raised. This interrupt is handled by the computer's OS based on the priority assigned to it. The delay seen by the computer's hardware is less predictable because of the variables that can increase delay, such as the number of interrupts the computer has to attend. Factors involved include the scheduling workload and the priority of the keyboard and other interrupts. The raised interrupt is eventually handled by the kernel, which handles one interrupt per scheduling clock cycle. This pattern is shown in Figure 2.1.

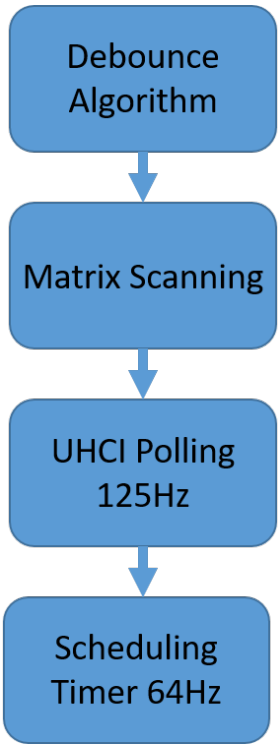


Figure 2.1. Sources of Keyboard Latency.

2.2 Clock Skew

Several methods exist to gather information about machines based on their clocks. Information about the clock is typically garnered from timestamps generated from network packets [3]. This information can be used to develop and determine the clock skew, a measure of the difference in the frequency of a clock and “true" clock [4]. Skew is affected by ambient temperature and the amount of work the machine is doing [5]. Temperature sensitivity can be attributed to the quartz oscillator, an integral part of the machine’s clock. A quartz crystal has a natural oscillating action when voltage is applied. The effect of temperature on the oscillation frequency depends on the cut of the crystal, causing it to slow down and speed up as the temperature changes [6]. This results in a change of the clock skew. Skew is a ubiquitous quality in clocks and if measured at a fine granularity, can be used to fingerprint a unique machine.

2.3 Fingerprinting

In order to fully understand this issue, the design of fingerprinting must be explained. Fingerprinting allows a service or individual to uniquely identify the machine or browser connected to the internet. This allows the identity to be tracked in a stateless manner when it revisits a web page, regardless if the user deletes the cookies stored on the machine. Fingerprinting is done for various reasons, primarily commercially to offer users tailored content on the web in an effort to sell products or to offer suggested search results.

2.3.1 Device Fingerprinting

Device fingerprinting focuses on tracking the individual machine, regardless of its user or what browser is being used. Some ways to fingerprint a device include the machine's Media Access Control (MAC) address, which remains unique over the lifetime of the machine and with the machine's clock skew. However, reading a machine's MAC address from within a sand-boxed environment (e.g., a web browser) is generally not feasible. Since the clock on a computer is a microchip containing a crystal on the computer's motherboard, it is unlikely to be changed over the machine's lifetime. Consequently, clock skew is a powerful way to fingerprint a machine as it doesn't rely on the web browser or other software installed on the machine.

2.3.2 Browser Fingerprinting

Browser fingerprinting is commonly used on web pages as a way to track users. Unlike device fingerprints, browser fingerprints generally depend on the browser type and version as well as any installed extensions. This is performed by making measurements on the host within the sand-boxed environment. For instance, the display characteristics and the fonts that are supported, and even whether or not the "do not track" feature is set, together form a fingerprint [7]. While these efforts do more to track web browsing than the standard approach of using cookies, its stability is affected by changes in the software, such as routine updates or modified settings.

As web browsers rely more heavily on the underlying hardware, namely to provide hardware-specific acceleration, they risk leaking information outside of the sandbox. This creates a window from the browser to the host machine that historically did not exist before, allowing

a web page to reach in and characterize device attributes from seemingly innocuous vectors. Since the text that is seen on the web has much more depth and detail than before, a website can use the way the text is rendered on the screen to preform fingerprinting. This is done by examining the differences in each pixel in the text. This attack can be done with pictures as well, offering information about the hosts graphics card [8].

2.4 Privacy Implications

Anonymity on the internet is a fleeting concept. As devices become increasingly connected and more information about the host is exposed through sand-boxed environments, it makes it more difficult to obscure identifying features. Furthermore, since many of these attacks do not necessarily reveal information that identifies a person, just their machine, some companies do not view fingerprinting as a moral dilemma, just a marketing tool [7]. This argument becomes moot because people use their devices to store and distribute personal information on social media and financial information on banking websites. This makes differentiating a person from a device almost irrelevant, especially since most individuals carry some or multiple types of connected device with themselves every day.

As the granularity of information exposed increases it also becomes more difficult to mask. For instance, to totally defeat rendered text fingerprinting computers would have to have built in randomness in the way text is displayed on the screen [8]. This randomness would probably not be detectable by the human eye, but it would be costly in computing power, causing the machine to run slower. A similar randomness could be applied to prevent clock fingerprinting, but since the clock is such a fundamental part of the computer's operation it would be difficult for a program to reliably change it without having negative side effects. If the trend continues, users will be forced to choose between risking their privacy or sacrificing features and usability to maintain some level of anonymity.

2.5 Related Work

A side channel attack leverages a system to find out information not intended to be revealed. One way of finding out information about a system is through the computer's clock. Kohno *et al.* [9] used clock data from timestamps sent over Transmission Control Protocol (TCP) packets and Internet Control Message Protocol (ICMP) requests and compared these times

to a known clock. Using this information, they were able to determine clock skew, which they speculated could be used to fingerprint devices over the internet. This exploit used clock information in a manner that the protocol designers did not intend. Murdoch [5] adjusted this attack to determine if the change of clock skew due to temperature could reveal hidden services, i.e., services running over Tor. In this model an attacker rapidly requests a file from a hidden server via Tor, where this server would normally be unknown to the attacker. The work done by that server creates heat which alters the clock skew, all the while a measuring computer polls potential servers and looks for the one with the corresponding clock skew shift, which reveals the hidden device.

Clock leakage was so prevalent that the original use of ICMP messages supporting time synchronization was superseded with Network Time Protocol (NTP). Once ICMP timestamps were obsoleted, hosts were largely allowed to treat this value however they wanted. This pivoted the problem and allowed different information to be divulged, including geolocation, OS, and clock skew in some cases [3]. Fingerprinting with clock skew relies on the fact that a machine had been encountered before and some ground truth data had been collected. Finding information that points to a particular OS can sometimes be done within the first samples of a machine seen for the first time. There are multiple examples of keyboard based side channel attacks dating back to when type writers outnumbered computers [10]. As one can imagine, these attacks focus on determining what the individual is typing. Present day attacks focus on using the microphones to detect unique sounds generated from typing, and even wearable technology. The latter attack involves using the accelerometers found within smart watches, which can pick up subtleties in wrist movements to determine keystrokes and personal identification numbers [11].

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 3:
Methodology

This experiment seeks to create data that is free from human variance. A device is created to ensure that input to the machines is predictable and identical, such that the measured timestamps only vary based on the behavior of the machine. From there, features are extracted to support multiple fingerprinting and profiling models based around numeric quantities and power spectrum approaches. This is done in order to determine if the machine's treatment of identical inputs lead to information leakage.

3.1 Data Collection

Experiments in this study centered around collecting keyboard timestamps that were generated at common intervals to examine the changes imposed by the machines. To create a repeatable keystroke string, an Arduino Leonardo drove a solid-state relay. When the program was started, a the relay would close for a random amount of time between 30 and 100 milliseconds (ms), simulating a key down and then open for between 40 and 300 ms, simulating the time between key presses; code seen in appendix A. A pseudo-random approach was taken to mimic typing by a human and to avoid introducing frequency patterns that could conceal anomalies. The same seed was used in each session such that the same sequence of delayed keystrokes are induced for each machine. The relay was connected to a INSIGNIA KN-PNK8001 keyboard matrix. This is a commonly available wired USB keyboard. The relay took the place of one of the key button switches to eliminate mechanical latency, which can be a major source of key latency [12]. Figures 3.1 and 3.2 show the device.

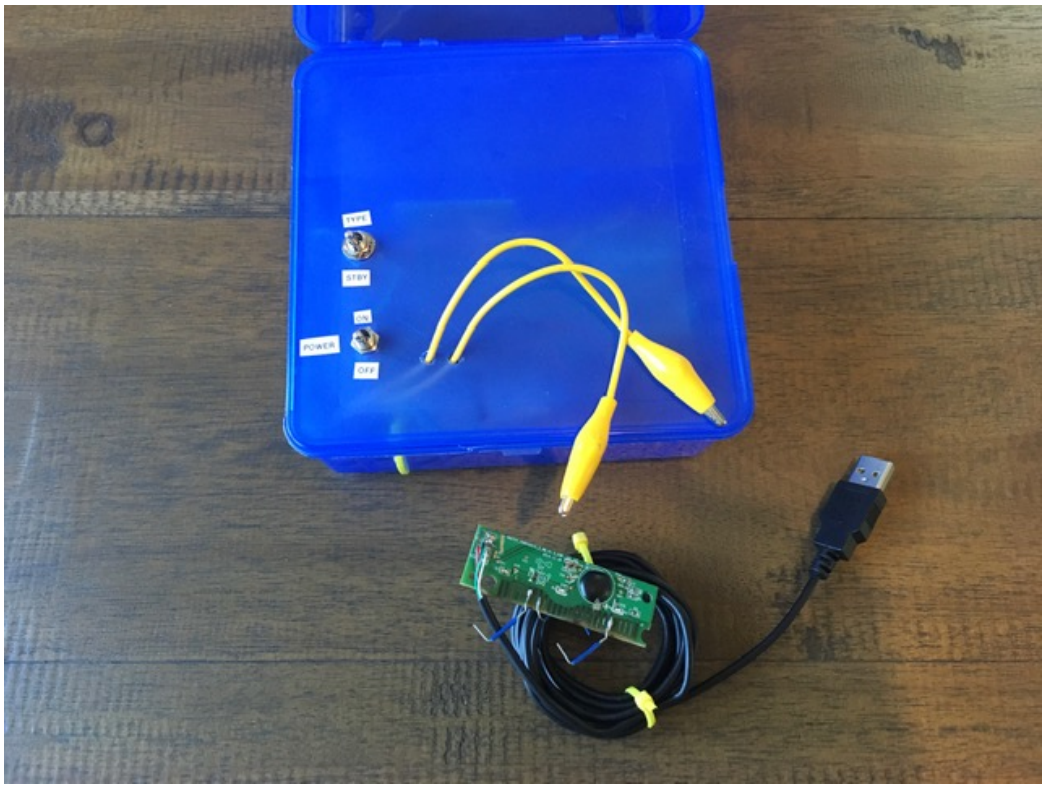


Figure 3.1. Key Generator with INSIGNIA Keyboard Circuit Below.

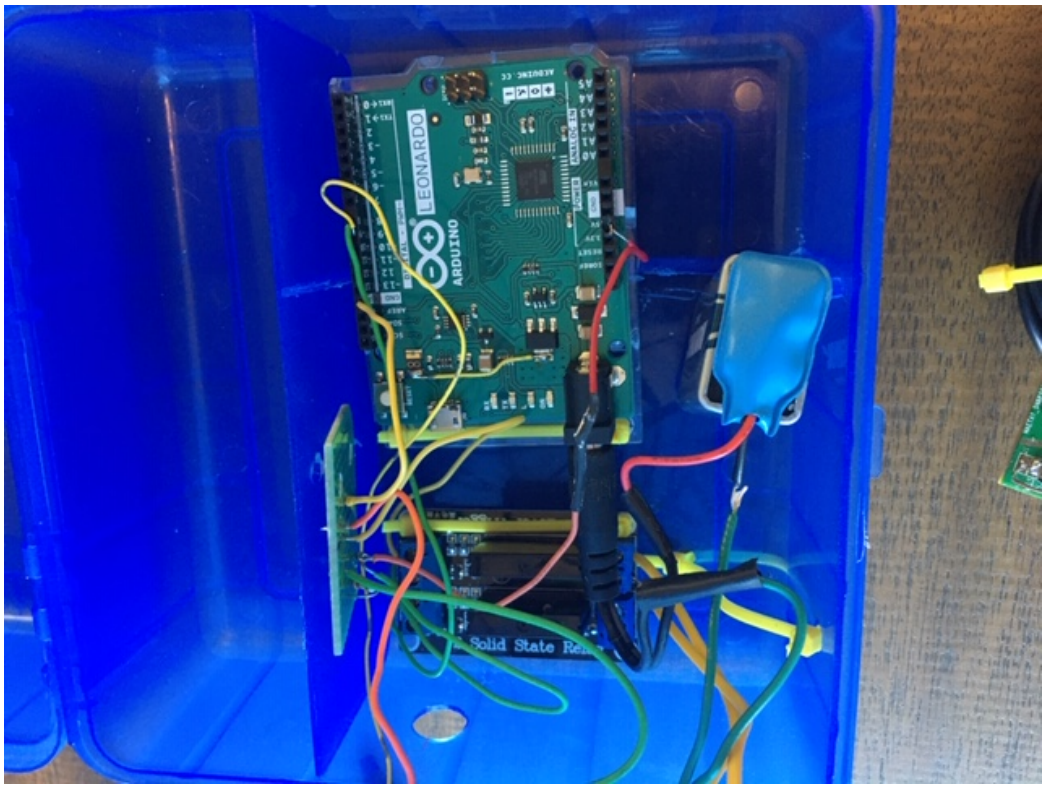


Figure 3.2. Key Generator Internal.

To collect the keyboard timestamps a logger written in JavaScript was hosted on a website. This program was constructed to collect the keyboard and mouse data as it is seen by a web page. This logger collected time in three ways: using the date.now and performance.now functions, as well as the event.time property attached to the DOM event. The functions date.now and performance.now generate a timestamp from within the browser. They differ in the fact that performance.now returns greater resolution. Arguably the most important time collected, and the one used in this study is the event.time source. This time is derived from the host machine OS when an event is generated, and is passed to the program in focus when the event was called; in this case, the logger web page. This timestamp is unique in that it is generated as the event is handled by the host machine and not at the mercy of the logger.

Collection was conducted in two rounds. The first being over six computers; three of which were loaded with a default Ubuntu 18.04 Linux image and three separate identical computers loaded with Windows 10. Identically timed keyboard strings were “typed" by

the relay driven keyboard into the web logger page on Google Chrome version 81 for two minutes, once a day for 4 days. This collection resulted in 1000 keyboard events per sample. A second collection was done on the six identical Windows 10 computers, also running Google Chrome version 81, for 3 minutes, twice a day for 5 days, resulting in 1280 keyboard events.

3.2 Feature Extraction

Two different techniques were compared to extract identifying features from the recorded timestamps. The first method of feature generation involved histograms of the number of occurrences of each number in each decimal place. This approach sought to take advantage of any rounding done to the timestamp.

The second method was to create power spectral densities of the intervals of the timestamps. This method would allow predominant frequencies imposed on the data by the various systems that affect data coming from the keyboard to be seen.

Two machine learning algorithms were used to perform both device profiling and fingerprinting. These include K Nearest Neighbor (KNN) and Random Forest (RF) classification models. The KNN works by comparing the distances of an unknown record to known records. The unknown values are plotted and classified according to a vote of known nearest neighbors [13], demonstrated in Figure 3.3.

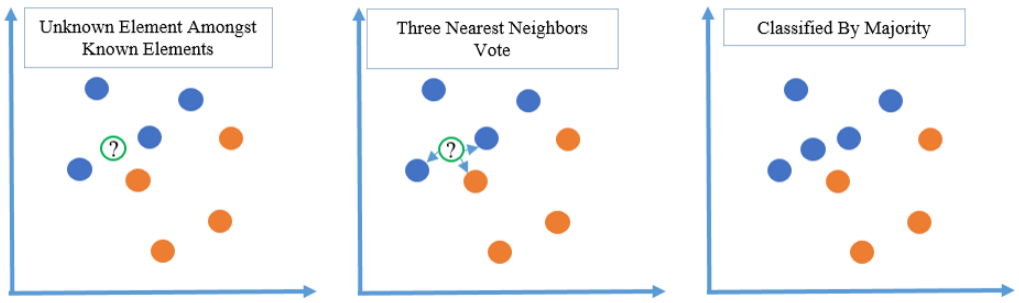


Figure 3.3. How an Unknown Element is Classified with KNN.

The RF classifier is comprised multiple decision trees. The decision trees are constructed by recursively splitting the training data by comparing feature values to a threshold in

order to minimize the impurity of each subset. This classifier consists of an ensemble of decision trees, where the vote of the trees determine the prediction [13], shown in Figure 3.4, reproduced from [14].

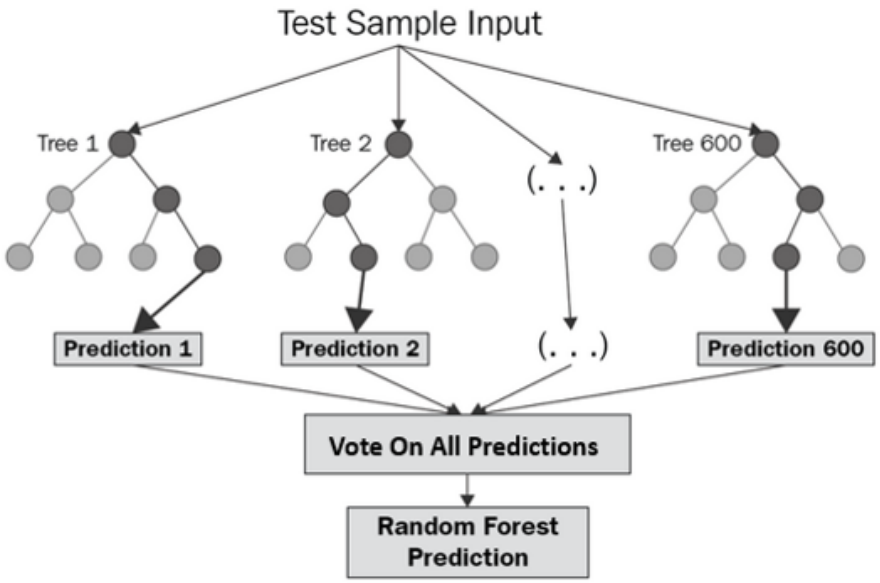


Figure 3.4. Example of a Random Forest Model. Source: [14].

3.2.1 Histogram Features

The first experiment run on the data was a histogram analysis on the distribution of the numeric value (0-9) of each decimal place (hundred to hundred billionth milliseconds) in the string of timestamps collected per machine per day. These counts were put into a Pandas DataFrame with a row belonging to each machine sample and the columns the digit frequency counts.

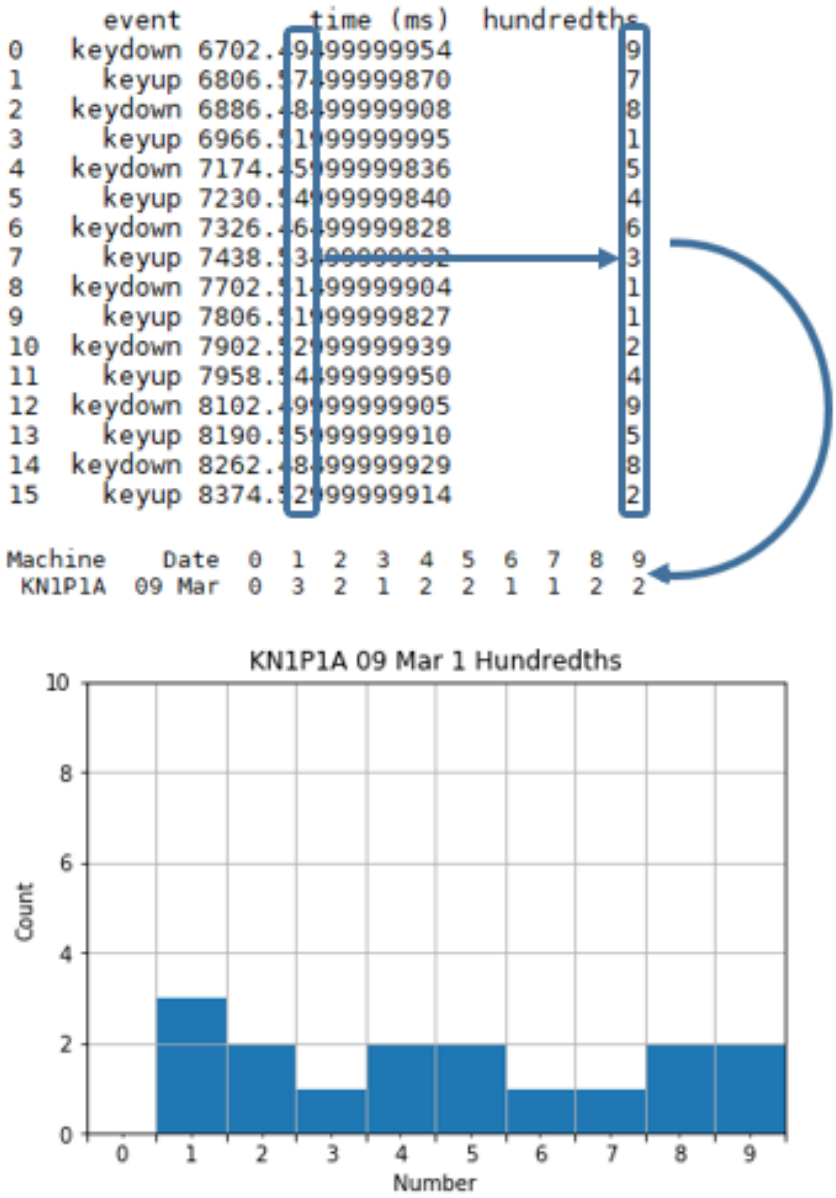


Figure 3.5. Example of Feature Generation.

Figure 3.5 shows a snippet of data and how the histogram experiment was constructed. In this, case the counts from the hundredths place was collected from one machine sampled on 09 March 2020. The use of digit frequency counts as identifying features is inspired by Benford's Law. This law declares that in naturally occurring numbers, smaller numbers appear more frequently than larger numbers, seen in Figure 3.6, reproduced from [15].

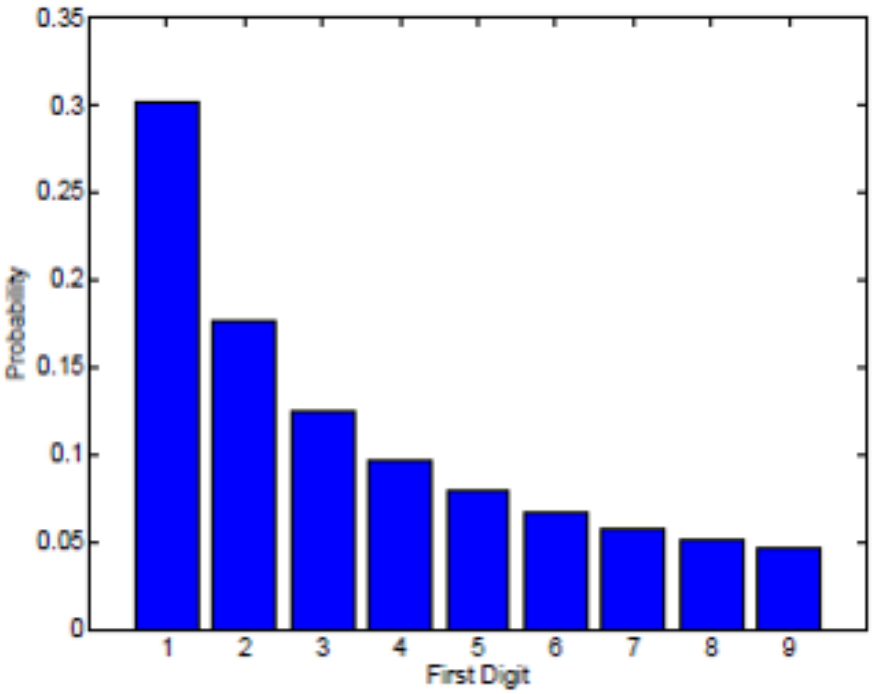


Figure 3.6. Typical Benford's Law Distribution. Source: [15].

There is work to suggest that human-generated keystrokes follow Benford's Law [15]. Due to this suggestion numerical distribution seemed like a possible avenue in distinguishing machines. However, while the histogram approach did have some success, the machine made distributions seen in this experiment were largely dissimilar to a Benford graph.

3.2.2 Power Spectral Density

The final experiment involved comparing the Power Spectral Density (PSD) of the recorded timestamps. This was done by using the scipy.signal library in an effort to explore predominant frequencies imposed on the timestamps by the machines. This was done via two methods.

The first technique to construct a PSD is a binning method. This method is represented by the equation in Figure 3.7.

$$P_s(f) = \frac{1}{N_p}\left|\sum_{n=0}^{N_p} s(n \cdot \Delta T) e^{2\pi i f n \Delta T}\right|^2$$

Figure 3.7. Binning PSD Equation [16].

This is done by turning the sparse array of collected timestamps into a dense array of length $n = t_{\max} - t_{\min}$ where $t_{\min}$ and $t_{\max}$ are the minimum and maximum timestamps represented in milliseconds, respectively. This dense array is comprised of zeros and ones, where the ones represent a time that an event occurred, or a filled bin. A discrete Fourier transform is conducted on the array to create the PSD. This is an accurate approach but does not offer very fine granularity because the timestamp is rounded according to the size of the bins. This approach also has a drawback in terms of frequency selection: the discrete Fourier transform lacks the ability to increase the resolution of the measured frequencies.

The second method used to construct PSD takes a shortcut since timestamps are point events and computes cumulative power for each frequency over the desired range. In a way, the power of each frequency is determined by overlaying wave plots over the array of timestamps, and summing the amplitudes from each point to find the total power at a given frequency.

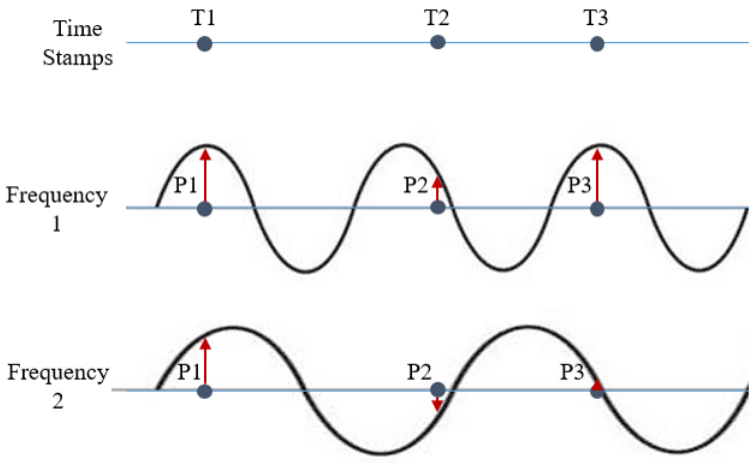


Figure 3.8. Example of Direct Power Computation.

Figure 3.8 illustrates how the power is calculated at two frequencies. Events are placed according to time of occurrence and the frequencies are plotted over them. The amplitude at a given time is demonstrated by the red arrow, with longer arrows corresponding to larger amplitude. The length of all the red arrows are summed to find the total power at a given frequency. In this case, one can see that with the same timestamps, frequency one has a greater power than frequency two. The equation for the directly computing the PSD is in Figure 3.9.

$$P_x(f) = \frac{1}{N}\left|\sum_{j=1}^{N} x(t_j)e^{2\pi i f t_j}\right|^2 = \frac{1}{N}\left|\sum_{j=1}^{N} e^{2\pi i f t_j}\right|^2$$

Figure 3.9. Direct PSD Equation [16].

The differences between the two computing methods do not end with their equations. Visualizing the graphs of both methods gives an idea of their robustness.

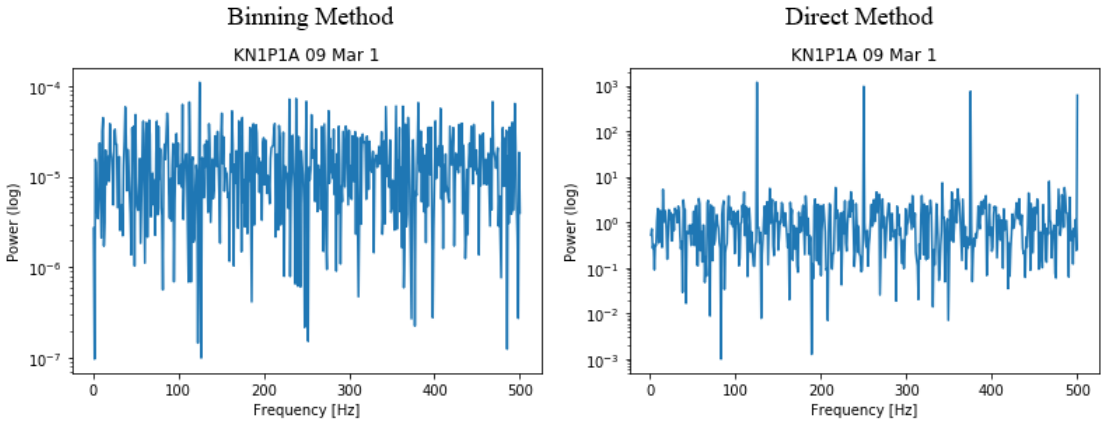


Figure 3.10. Plots Generated from Binning and Direct PSD Methods.

Figure 3.10 depicts both methods that were generated from the same data. While the peaks in the binning method graph do show the relative power at 125Hz since the power spread generated by the discrete Fourier transform is small, it is less obvious than the 125Hz power seen on the direct method graph.

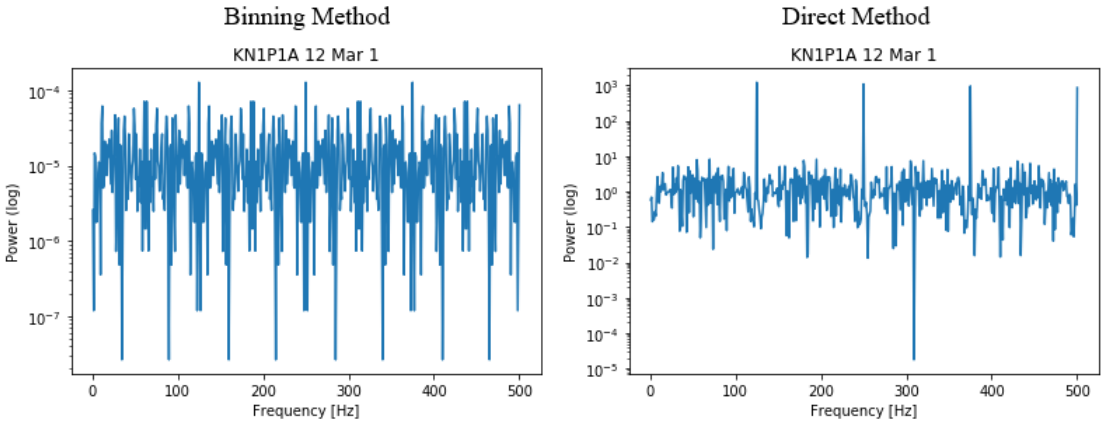
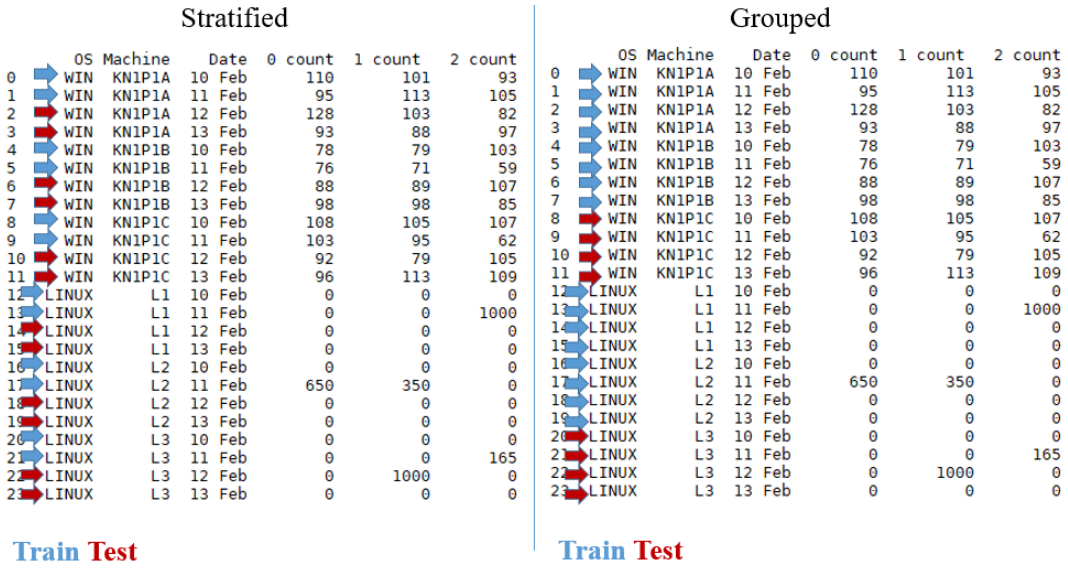


Figure 3.11. Plots Generated from Binning and Direct PSD Methods. (Note the Binning Anomaly.)

In Figure 3.11, comparing these graphs shows the anomalous behavior that is possible in the binning graphs. The powers seem to repeat themselves at 125Hz intervals. Unlike a typical harmonic response, these powers do not substantially decrease across the subsequent harmonics.

3.3 Device Profiling and Fingerprinting

Both profiling and fingerprinting required different approaches in validating model accuracy. Figure 3.12 demonstrates two different ways to split training and testing data.



Stratified

	OS	Machine	Date	0 count	1 count	2 count
0	WIN	KN1P1A	10 Feb	110	101	93
1	WIN	KN1P1A	11 Feb	95	113	105
2	WIN	KN1P1A	12 Feb	128	103	82
3	WIN	KN1P1A	13 Feb	93	88	97
4	WIN	KN1P1B	10 Feb	78	79	103
5	WIN	KN1P1B	11 Feb	76	71	59
6	WIN	KN1P1B	12 Feb	88	89	107
7	WIN	KN1P1B	13 Feb	98	98	85
8	WIN	KN1P1C	10 Feb	108	105	107
9	WIN	KN1P1C	11 Feb	103	95	62
10	WIN	KN1P1C	12 Feb	92	79	105
11	WIN	KN1P1C	13 Feb	96	113	109
12	LINUX	L1	10 Feb	0	0	0
13	LINUX	L1	11 Feb	0	0	1000
14	LINUX	L1	12 Feb	0	0	0
15	LINUX	L1	13 Feb	0	0	0
16	LINUX	L2	10 Feb	0	0	0
17	LINUX	L2	11 Feb	650	350	0
18	LINUX	L2	12 Feb	0	0	0
19	LINUX	L2	13 Feb	0	0	0
20	LINUX	L3	10 Feb	0	0	0
21	LINUX	L3	11 Feb	0	0	165
22	LINUX	L3	12 Feb	0	1000	0
23	LINUX	L3	13 Feb	0	0	0

Grouped

	OS	Machine	Date	0 count	1 count	2 count
0	WIN	KN1P1A	10 Feb	110	101	93
1	WIN	KN1P1A	11 Feb	95	113	105
2	WIN	KN1P1A	12 Feb	128	103	82
3	WIN	KN1P1A	13 Feb	93	88	97
4	WIN	KN1P1B	10 Feb	78	79	103
5	WIN	KN1P1B	11 Feb	76	71	59
6	WIN	KN1P1B	12 Feb	88	89	107
7	WIN	KN1P1B	13 Feb	98	98	85
8	WIN	KN1P1C	10 Feb	108	105	107
9	WIN	KN1P1C	11 Feb	103	95	62
10	WIN	KN1P1C	12 Feb	92	79	105
11	WIN	KN1P1C	13 Feb	96	113	109
12	LINUX	L1	10 Feb	0	0	0
13	LINUX	L1	11 Feb	0	0	1000
14	LINUX	L1	12 Feb	0	0	0
15	LINUX	L1	13 Feb	0	0	0
16	LINUX	L2	10 Feb	0	0	0
17	LINUX	L2	11 Feb	650	350	0
18	LINUX	L2	12 Feb	0	0	0
19	LINUX	L2	13 Feb	0	0	0
20	LINUX	L3	10 Feb	0	0	0
21	LINUX	L3	11 Feb	0	0	165
22	LINUX	L3	12 Feb	0	1000	0
23	LINUX	L3	13 Feb	0	0	0

Figure 3.12. Comparison of Grouped and Stratified Train Test Split.

The data in Figure 3.12 is an excerpt from the histogram method where the numbers in the hundredths of milliseconds are counted for each sample. The 50% stratified data on the left uses half of each machine's samples for testing. This is good for making sure every machine is represented in both the training and the testing sets. The grouped data on the right uses the information from four machines to train and then looks at two machines to test the model. The significant difference here is that the stratified approach ensures that all machines are represented in both training and testing, where grouping the data ensures the machines that appear in training and testing sets are mutually exclusive (i.e., each machine appears in either training or testing sets, but not both).

3.3.1 Device Profiling

Device profiling seeks to find out a specific detail of the device. This work seeks to build a model to determine the OS of a device, but profiling could be used to find out the device type, browser version, or model. This exploit assumes that the attacker has collected data on known machines, and then uses that information to profile a machine that has not been

seen before or is not otherwise known. The first round of data, collected over the Linux and Windows machines, was used to train and test a Random Forest classifier. The concatenated histograms of the counts of the fourteen decimal places were used as features to predict the OS of each machine.

3.3.2 Fingerprinting

Fingerprinting seeks to assign an identity to each device seen. This attack makes the assumption that if new data correlates to a device before, it is attributed the same identity. If data does not match up with a previously seen device, it is assigned a unique identity. This differs from profiling because to be done accurately, fine granularity needs to be used to identity every unique device seen, compared to determining broad characteristics in profiling. In the second round of data, the six identical Windows machines were put through two tests. The first was a series trained 14 Random Forest each using a count of one decimal place. The models were trained to predict machine identity. This experiment was run one hundred times and the results averaged. The goal of this experiment is to determine which, if any, decimal places carry machine-specific information. The second test was conducted using the PSD features. This was performed with a KNN classifier, where the power versus frequencies were features and the machine was the predicted class. Stratification was used for the second test, which preserves the percentage of each predicted class in both training and testing sets. That is, instead of randomly selecting training data at random, which could possibly exclude some of the classes, each class is split with the same ratio.

CHAPTER 4:
Results

Profiling results were very successful, resulting in 100% OS prediction accuracy. Fingerprinting accuracy was above a random-guessing baseline of 16% in every case, with some approaches revealing significant uniqueness in a particular machine.

4.1 Profiling

Using the concatenated histograms of the counts of the fourteen decimal places generated from the four days of Windows and Linux machines as features, a Random Forest Model was applied to judge profiling capability. Two machines of each OS were used to train and a unique machine was used to test. This was done to support the hypothesis that profiling could be accomplished and applied to machines that have not been collected on before. Real world consequence of this ability is that a website, having previously collected or using open source data, could determine the OS of a machine that had never before visited the site within seconds. This test was run 100 times and resulted in 100% accuracy in determining the OS of the machine not before seen. This test was repeated but this time only using the counts in the hundredths place, which yielded the same results of 100% classification accuracy. A hundredths histogram from each OS is shown in Figure 4.1.

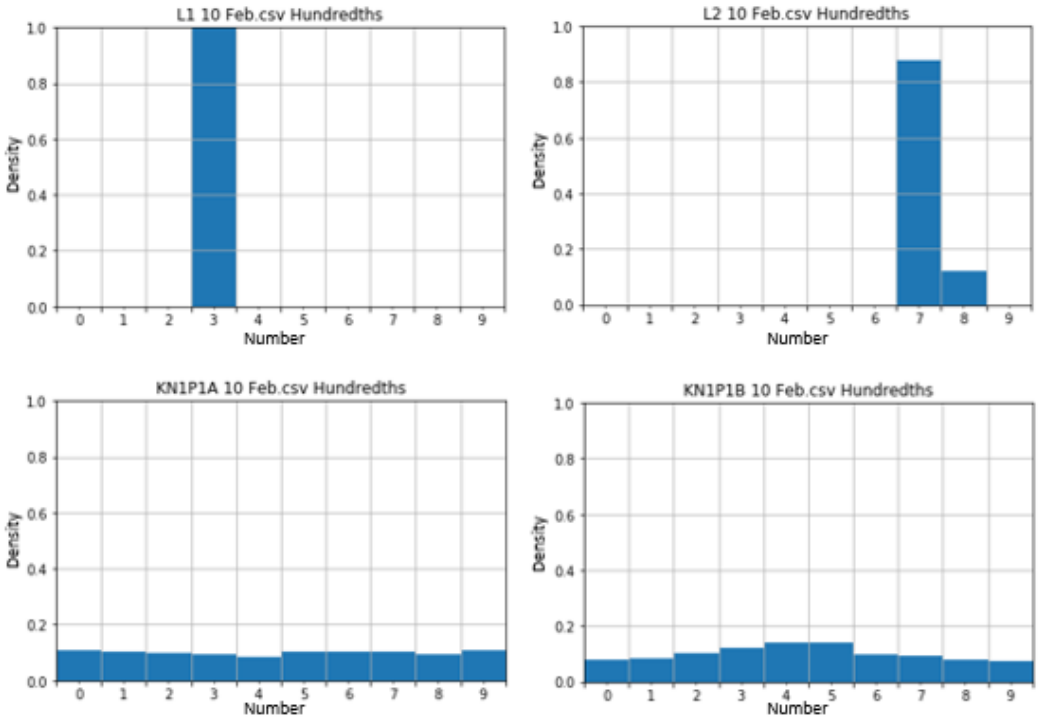


Figure 4.1. Hundredth of Milliseconds Place Histogram from Three Linux Machines (Top) and Three Windows Machines (Bottom).

Figure 4.1 demonstrates a huge disparity between the digit frequency distributions on Linux and Windows operating systems. Since the same keyboard, keyboard input timing, browser, and collection method were used, this difference can be attributed to the OS. In fact, the histograms of each of the decimal places demonstrates that Linux rounds the timestamp that is reported to the browser. An excerpt of event timestamps is shown in Figure 4.2.

KN1P1C 12 Feb.csv

	event	time
0	keydown	11265.37499999540
1	keyup	11369.42499999714
2	keydown	11441.33999999758
3	keyup	11529.40999999555
4	keydown	11737.43999999715
5	keyup	11792.32499999489
6	keydown	11889.38999999664
7	keyup	12001.42499999492
8	keydown	12265.50499999576
9	keyup	12360.61999999947
10	keydown	12457.42999999493

L3 12 Feb.csv

	event	time
0	keydown	2508.01500002854
1	keyup	2612.01500002062
2	keydown	2740.01500004670
3	keyup	2804.01500000153
4	keydown	2956.01500000339
5	keyup	3020.01500001643
6	keydown	3156.01500001503
7	keyup	3236.01500003133
8	keydown	3332.01500005089
9	keyup	3412.01500000898
10	keydown	3612.01500002062

Figure 4.2. Example of Event Time from a Windows Sample (Left) and Linux Sample (Right) in Milliseconds (Note the Hundredths Place).

Profiling was attempted using the directly computed PSD of the six devices. A KNN model using the five nearest neighbors was used; four machines were used to train and two, one of each OS, were used to test. The PSD was computed for frequencies ranging from from 1 to 500Hz in 1Hz increments (i.e., 500 features total). Accuracy was lower than the histogram method at 87.5%. Figure 4.3 shows a confusion matrix of the results.

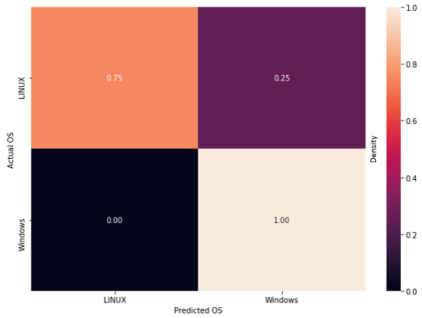


Figure 4.3. Normalized confusion Matrix of Direct PSD Classifying.

The confusion matrix showed that one of the four tested Linux machine samples was mistakenly classified as a Windows machine.

4.2 Fingerprinting

In the second round of data, the six identical Dell OptiPlex Windows machines were put through several tests in order to see if the keyboard timestamps would allow the machines

to be fingerprinted. These tests consisted of sixty samples; two samples for each of the six machines for five days. This was done both with the histogram method, as described in the previous section, and by investigating the power spectrum density of the machines.

4.2.1 Histogram Approach

This experiment was comprised of 14 experiments each using a Random Forest classifier with 80/20 stratified, train-test split. Each experiment used the digit frequency features for one decimal place in an attempt to predict the physical machine. On first glance it seemed as if the histogram approach may have promise as a fingerprinting tool. The model was run one hundred times and the results averaged.

Place	Average Score
Hundreds	13.47%
Tens	11.10%
Ones	12.70%
Tenths	18.87%
Hundredths	15.60%
Thousandths	13.67%
Ten Thousandths	19.33%
Hundred Thousandths	12.30%
Millionths	15.23%
Ten Millionths	19.43%
Hundred Millionths	13.17%
Billionths	17.87%
Ten Billionths	16.03%
Hundred Billionths	16.23%

Figure 4.4. Random Forest Accuracy Results for Each Decimal Place.

Figure 4.4 reveals that even the most unique decimal place scored approximately 3% better than the baseline accuracy of 16.6%. The model was run again with the concatenated histogram approach 100 times. The average score was 20.33%, merely a 4% improvement despite the increase in features and computation resources.

4.2.2 Power Spectral Approach

The second test was conducted on both the direct and binned PSD. This was done with a KNN approach using the vote from the five nearest neighbors. In this model, the power versus frequencies from 1 to 500 Hz were the features and the machine was the predicted class. The first model was trained using the binned PSD method. The train, test split was 80%, 20% stratified. It was run 100 times and the average score was 30.7%, or approximately 14% above the baseline accuracy. Next, the direct PSD was run with the direct method and its accuracy was 27.8%. A confusion matrix was developed for both tests, Figure 4.5.

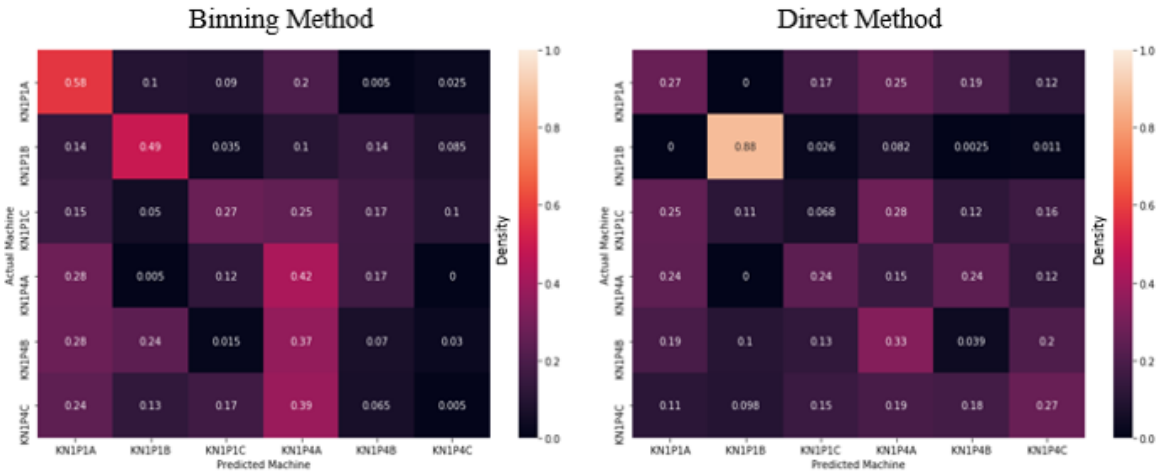


Figure 4.5. Results of Fingerprinting with Binning and Direct PSD.

While the PSD approach thus far had managed to increase accuracy over the histogram tests, there was an interesting finding with the direct method; it was able to classify machine KN1P1B with 88% accuracy.

As demonstrated in chapter three, the PSD had peaks around 125Hz, which corresponds with UHCI polling rate [1]. Since perhaps the variance of the polling rate might leak some information, another KNN was trained with frequency features ranging from 124.5Hz to 125.5Hz with 0.2mHz increments to increase granularity around this significant frequency. Figure 4.6 is a direct PSD.

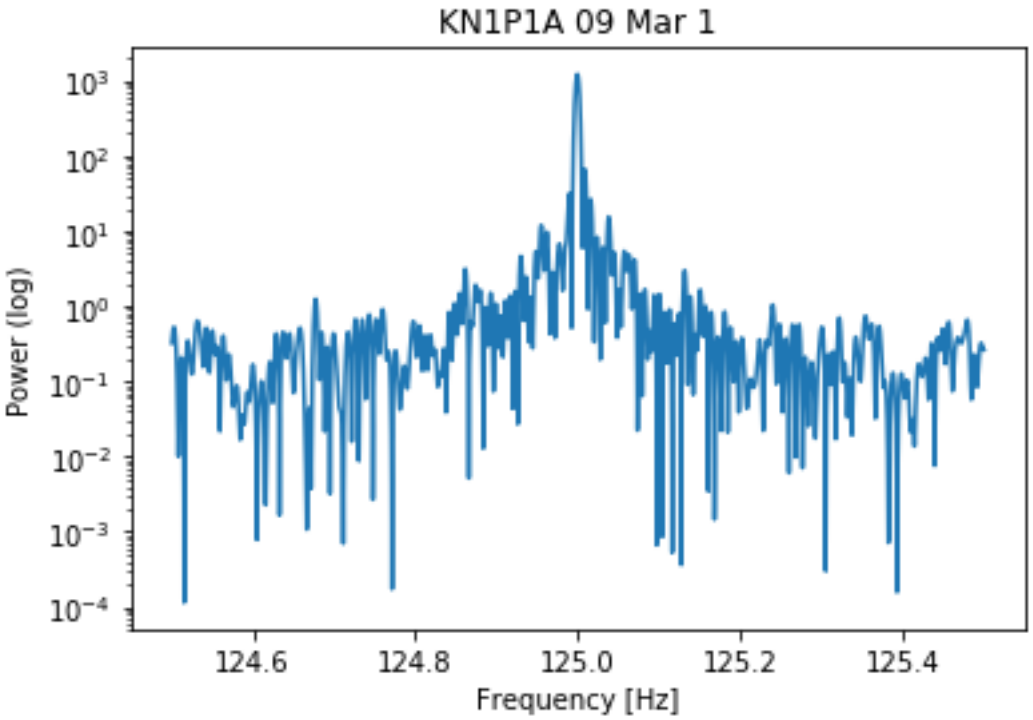


Figure 4.6. Example of Direct PSD from 124.5Hz to 125.5Hz.

The KNN was trained on the direct PSD with the 1Hz range and the resulting accuracy was 25.8%. This is slightly less accurate than the 500Hz spectrum direct test. A scatter plot was developed to take a closer look at the dominant frequency from each sample using the direct PSD method.

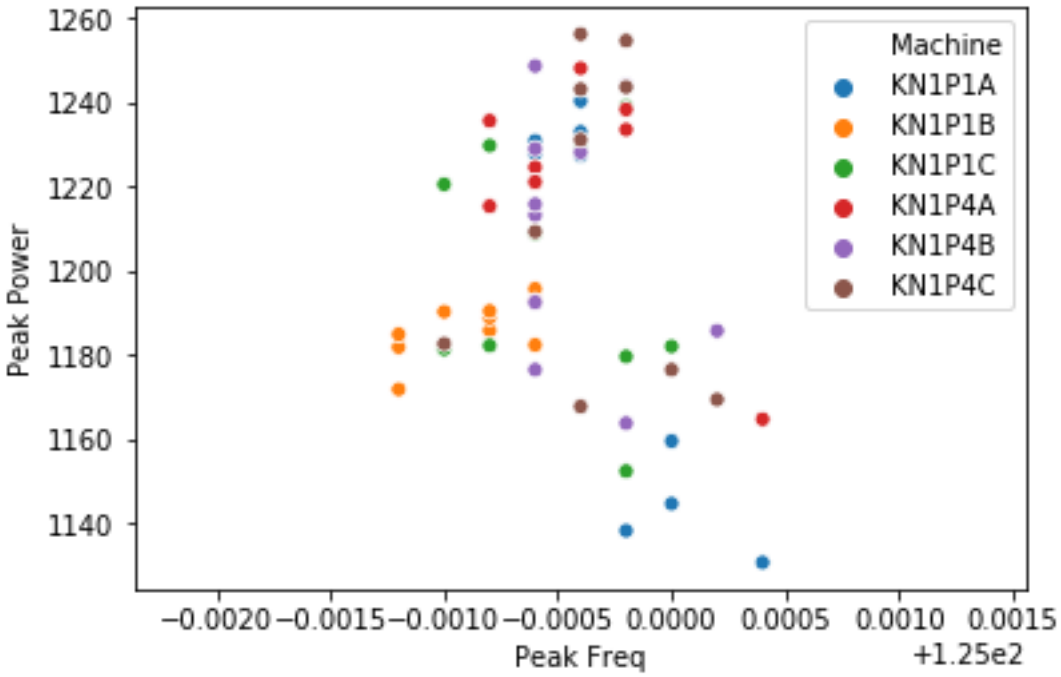


Figure 4.7. Dominant Frequency Plot Across all Samples.

Figure 4.7 shows the tendency of each clock to vary and demonstrates that while the optimum frequency may be 125Hz, there is a measurable amount of skew in most of the samples. With this information in mind, a final test was run just using the data from the scatter plot. Where only the dominant frequency and its corresponding power would be used as features. This eliminated all the noise and other signals present in the PSD. The average accuracy, again using an 80%, 20% stratified train-test split KNN model with five nearest neighbors, was 36.8%. That result is 20% above the baseline and a 5% increase from the next most accurate model, Figure 4.8.

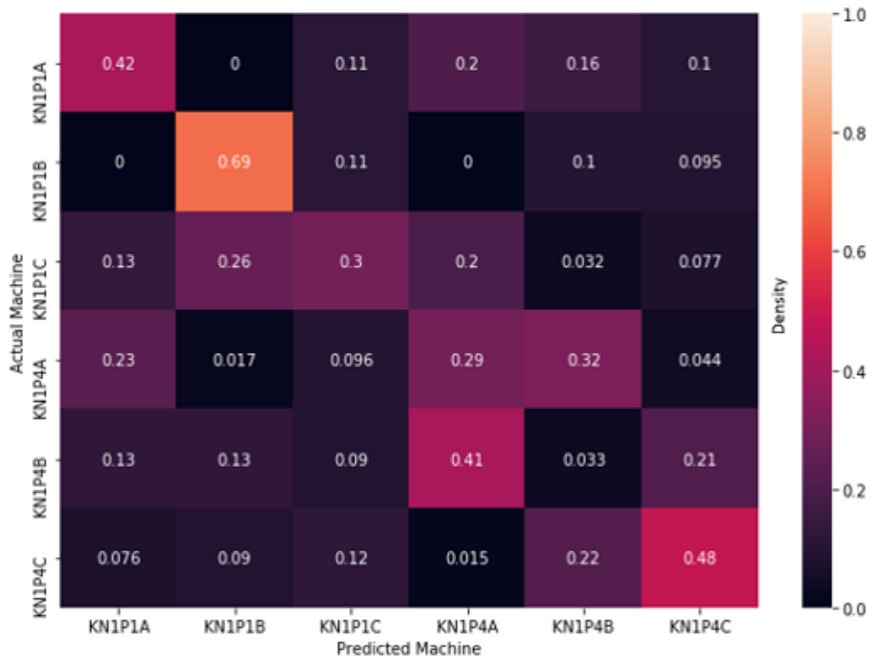


Figure 4.8. Confusion Matrix of KNN Using only Peak Frequencies.

While the 500Hz direct test is not able to predict any machine besides KN1P1B with more than 27% accuracy, this model is able to improve the prediction accuracy on four of the five remaining machines. Due the features being so few, just two per sample, this is also the quickest classification tool in terms of computation resources.

CHAPTER 5:
Conclusion

Experiments supported the hypothesis that information leakage form the keyboard was authentic. This seemingly innocuous peripheral contributed in allowing machines to be profiled and fingerprinted.

5.1 Discussion

These experiments showed there is information leakage from user defined input events. Even with controlled and machine generated keystrokes it is fully possible to passively determine the OS of a computer typing into a web page. This leakage also led to being able to pick out a particular machine out of a group of six with identical models and the same keyboard with almost 90% accuracy. This non-zero information leak is significant, especially when applied to the world of diverse machines that are connected to the web it lowers the ever shrinking anonymity pool.

5.1.1 Defenses

There are two defenses to the techniques presented in this work. The first is to mask the subtleties of machine clocks by introducing jitter in the timestamps [17]. By adding noise to the timestamps it makes it more difficult to capture the fine granularity needed to fingerprint the machine. This defense comes at a cost computationally because a random number needs to be generated for each event and applied to the timestamp. In addition to more computational work, it is possible this defense could become subject to a side channel exploit as well. If noise introduction is done in a non-standard manner it could reveal information about the implementation and allow the same type of profiling that was seen from the timestamp rounding in the histogram approach in Chapter 4.

Another defense method is preventing the timestamps from being affected by a user's actions. In this defense presented by Cao *et al.* [18], the clock is made deterministic by reporting a time that is derived independently to the host machine's hardware. This makes the attacker see a time that is in a different reference frame from the host.

5.2 Future Work

- This work focused on the keyboard. The mouse is likely another candidate source of information leakage. Mouse events come in much faster than typical keyboard inputs and there is a possibility that this can offer a new level of granularity that can be explored.
- In addition to the potential from investigating different devices, other browsers could be tested. The Tor browser takes steps to prevent fingerprinting and surveillance by providing only low-resolution time sources, but alternative methods to measure time with higher resolution have the potential of revealing unintended machine information.
- This study could be also be extended to different versions of the same OS. Profiling methods between Windows and Linux was 100% successful. It is possible that unique versions of the same OS may be able to be distinguished.

5.3 Closing Thoughts

This leakage is from the timestamps generated by a peripheral that nearly every computer uses: the keyboard. Keyboard timestamps are permissionless, therefore they can be collected without the user's knowledge and can reveal unintended information about the host machine that could be leveraged to mount a targeted attack. This study has shown that seemingly identical machines are prone to fingerprinting and has determined that the possibility exists that users may be tracked according to idiosyncratic device behavior.

APPENDIX A:
Keystroke Generation Code

```
#include <Wire.h>

const int relay = 8; // output to relay
const int rswitch = 9; // input from go/stop switch
int randDown;
int randUp;
int switchState = 0;
void setup ()
{
  pinMode(LED_BUILTIN, OUTPUT);
  randomSeed (123);// random numbers repeat each run
  pinMode(9,INPUT);
  pinMode(8,OUTPUT);
}

void loop ()
{
  switchState = digitalRead(rswitch);//read go/stop switch
  if (switchState == HIGH)
  {
      randUp = random(40, 300);//normal distro pseudo random number
   between 40 and 300
      delay(randUp); // random 40 to 300 ms for the delay before the key
    is pressed
      digitalWrite(relay,HIGH); // Pin 8 to SS relay, key is down
      digitalWrite(LED_BUILTIN, HIGH); //onboard LED
      randDown = random(30, 100); //pseudo random 30 to 100 ms for the
   pressed time
      delay(randDown);//delay for key press
      analogWrite(relay,LOW);// open the relay, key is up
      digitalWrite(LED_BUILTIN, LOW);//extinguish the light
  }
  else //standby flash onboard LED
  {
```

```
33          digitalWrite(LED_BUILTIN, HIGH); //onboard LED
            delay(500);
35          digitalWrite(LED_BUILTIN, LOW);//extinguish the light
            delay(500);
37      }
    }
```

List of References

[1] *Universal Host Controller Interface (UHCI) Design Guide*, Revision 1.1, Intel Co., March 1996. [Online]. Available: ftp.netbsd.org/pub/NetBSD/misc/blymn/uhci11d.pdf

[2] J. Atwood. Mouse DPI and USB Polling Rate. Blog Post, Coding Horror Blog, 2 April 2007. [Online]. Available: https://blog.codinghorror.com/mouse-dpi-and-usb-polling-rate/

[3] E. C. Rye and R. Beverly, "Sundials in the shade," in *Proceedings of the 20th International Conference on Passive and Active Measurement (PAM)*, Puerto Varas, Chile, Mar, 2019, pp. 82–98.

[4] S. Moon, P. Skelly, and D. Towsley, "Estimation and removal of clock skew from network delay measurements," in *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, New York, NY, USA, Mar. 1999, pp. 227–234.

[5] S. J. Murdoch, "Hot or not: Revealing hidden services by their clock skew," in *Proceedings of the 13th ACM Conference on Computer and Communications Security* (CCS '06), Alexandria, VA, USA, Oct. 2006, pp. 27–36.

[6] "HC49/4H SMX Crystals," Premier Farnell Ltd, Leeds, UK, Tech. Rep. Issue 13, 28 Sept. 2004. Available: https://www.farnell.com/datasheets/4547.pdf

[7] G. A. Fowler, "Think you're anonymous online? A third of popular websites are 'fingerprinting' you," *The Washington Post*, 31 October 2019.

[8] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in HTML5," in *Proceedings of W2SP 2012*, M. Fredrikson, Ed. IEEE Computer Society, May 2012.

[9] T. Kohno, A. Broido, and K. Claffy, "Remote physical device fingerprinting," in *2005 IEEE Symposium on Security and Privacy (SP'05)*, Oakland, CA, USA, May 2005, pp. 211–225.

[10] J. V. Monaco, "Sok: Keylogging side channels," in *2018 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, May 2018, pp. 211–228.

[11] R. Anderson, *Security engineering: A guide to building dependable distributed systems*, 2nd ed. Indianapolis, IN, USA: Wiley, 2008.

[12] D. Luu. Keyboard latency. Accessed Sept. 10, 2019. [Online]. Available: https://danluu.com/keyboard-latency/

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, no. null, Feb 2011, pp. 2825–2830.

[14] A. Chakure. Random Forest Regression. 29 June 2019. [Online]. Available: https://towardsdatascience.com/random-forest-and-its-implementation-71824ced454f.

[15] A. Iorliam, A. Ho, N. Poh, S. Tirunagari, and P. Bours, "Data forensic techniques using Benford's law and Zipf's law for keystroke dynamics," in *3rd International Workshop on Biometrics and Forensics (IWBF 2015)*, Mar 2015, pp. 1–6.

[16] B. Sadler and S. Casey, "On periodic pulse interval analysis with outliers and missing observations," *IEEE Transactions on Signal Processing*, vol. 46, no. 11, Nov 1998, pp. 2990-3002.

[17] J. V. Monaco and C. Tappert, "Obfuscating keystroke time intervals to avoid identification and impersonation," *arXiv preprint arXiv:1609.07612*, 2016.

[18] Y. Cao, Z. Chen, S. Li, and S. Wu, "Deterministic browser," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, Dallas, TX, USA, Oct 2017, pp. 163–78.

Initial Distribution List

1. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

2. Defense Technical Information Center
   Ft. Belvoir, Virginia