EXPLOITING USB POWER READINGS FOR HIGH-RESOLUTION HOST FINGERPRINTING

by

CAMERON JUAREZ

A THESIS

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the honors requirements
for the degree of
Bachelor of Science

June 2014

THESIS APPROVAL PAGE

Student: Cameron Juarez

Title: Exploiting USB Power Readings for High-Resolution Host Fingerprinting

This thesis has been accepted and approved in partial fulfillment of the requirements for the Bachelor of Science honors program in the Department of Computer and Information Science by:

Prof. Kevin Butler Advisor

Degree awarded June 2014

THESIS ABSTRACT

Cameron Juarez

Bachelor of Science

Department of Computer and Information Science

June 2014

Title: Exploiting USB Power Readings for High-Resolution Host Fingerprinting

Having confidence that a user can identify a machine they want to communicate with is critical for users who wish to keep their data safe. Being able to establish the identity of a machine is of critical importance for establishing user trust. In this paper, we outline a methodology to leverage the ubiquitous USB interface and power measurements consisting of voltage, wattage, and amperage to uniquely identify machines. We collect USB power samples from a corpus of 45 machines on the University of Oregon campus and through machine learning classifier techniques, we demonstrate that it is possible to use statistical metrics extracted from these samples to differentiate hosts based on class label. Using these statistical metrics, we are able to correctly differentiate sampled hosts by ID with an upwards of 94% accuracy. In addition we are able to identify characteristics about the machines such as model number and operating system with an accuracy of 98%. Using a Random Forest classifier were are able to generate fingerprints that are capable of consistently distinguishing hosts that are seemingly identical with an accuracy of 98% as well. Later in our analysis we show that our methods can be extended to determine other characteristics about target hosts as well such as what USB devices are connected. Our techniques are deployable in an accessible, low-cost fashion.

CURRICULUM VITAE

NAME OF AUTHOR: Cameron Juarez

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:
University of Oregon, Eugene

DEGREES AWARDED:
Bachelor of Science, Computer and Information Science, 2014, University of Oregon

AREAS OF SPECIAL INTEREST:
Machine Learning
Web Applications
Backend Infrastructure
Security
Cryptography

PROFESSIONAL EXPERIENCE:

Software Engineer, Google, Mountain View, California, Summer 2014 - Present

Developer, Venture Dept., Eugene, Oregon, Fall 2013 - Spring 2014

Software Engineer Intern, Google, Mountain View, California, Summer 2013

Software Development Engineer Intern, Amazon.com, Seattle, Washington, Summer 2012

ACKNOWLEDGEMENTS

I would like to thank Adam Bates for his time, guidance, and mentorship throughout the completion of this thesis. I would like to thank Kevin Butler and the OSIRIS lab for their input and feedback.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

1.1 Background 1

II. METHODOLOGY 5

2.1 Why Power Readings 5
2.2 Implementation 7
2.3 Collecting Samples 13
2.4 Data Corpus 17
2.5 Feature Extraction 19

III. EVALUTATION 22

3.1 Classification Algorithms 22
3.2 Results 23

IV. DISCUSSION 25

4.1 Analysis 25
4.2 Future Work 28

Chapter Page

V. RELATED WORK 30

5.1 USB Fingerprinting 30

5.2 Remote Fingerprinting 31

5.3 Learning Based Classifier Evasion 31

VI. CONCLUSION 33

REFERENCES CITED 34

LIST OF FIGURES

Figure Page

2.1 An image of our setup while the programmer can communicate with the power gauge. 8

2.2 Power gauge output before source modifications. 10

2.3 Unaltered printDotDecimal function used in place of the Arduino floating point library to print floats. 11

2.4 printDotDecimal with our additions to support 3 decimal points. 12

2.5 Main loop function loaded on the power gauge. While the device is on, this function is called in repetition indefinitely. 13

2.6 An image of the power gauge we used (provided by Adafruit). 14

2.7 Voltage over time from raw measurements of iMac 13.2's in our data corpus. 15

2.8 Cumulative distribution functions for amperage and wattage measurements across all iMac 13.2 hosts. 17

2.9 Cumulative distribution functions for voltage measurements across all iMac 13.2 hosts. 20

4.1 Voltage cumulative distribution functions of a single iMac 12.2 named Ortega. 26

4.2 Cumulative distribution functions for amperage and wattage measurements on Ortega host with three load variations. 27

LIST OF TABLES

Table Page

2.1 Description of data corpus. 18

2.2 The 5 highest ranked features by information gain. 21

3.1 OS Version accuracies by class label. 23

3.2 OS Model accuracies by class label. 23

4.1 Identification accuracy under varied loads of the machine, Ortega. . . . 27

4.2 The 5 highest ranked features by information gain for Ortega samples with varied load. 28

CHAPTER I

INTRODUCTION

Long has the topic of machine identification been a discussion in the computer security community. In today's world, the transfer of sensitive data is a commonplace transaction that people utilize every day (e.g. online banking, job applications [SSN], secure sign-on, etc) via a variety of different data transfer mediums such as flash drives, the Internet, and USB cables. Encryption exists to protect this data, but a more challenging problem remains posed: how can one determine if the machine they are communicating with is actually the one they think it is? Solving this problem has proven to be more challenging than it seems.

1.1 Background

Being able to establish identity is a core prerequisite for establishing trust. Consider a server in a data center, one would expect that verifying characteristics such as the model number, serial number, etc should be sufficient to confidently assume that the machine belongs to the data center and has not been tampered with. Despite this, even if the machine is physically present and can be observed to be as expected, there is little preventing the machine from surreptitiously relaying commands to an external host and providing responses from the same host in place of the machine's. Such a relay attack was posed by Parno who dubbed it a "cuckoo" attack [**?**].

How to alter and effectively fake a machine's serial number is common knowledge thanks to easily accessible resources on the Internet [**?**]. As a result, a machine can be swapped in place of another machine with the same hardware and serial number while

avoiding detection by the owner of the original machine. Methods of creating uniquely identifying "fingerprints" of machines have been posed in previous works with the potential to thwart such an attack but these methods suffer from low accuracy in practice [?].

In this thesis we propose that ubiquitous USB interface supported by USB ports found on nearly every modern computer and some mobile devices can be used to determine identity with higher accuracy than previous works. With the methods we propose, even seemingly identical machines composed of the same hardware can be uniquely differentiated. Minute differences in USB stack and hardware imperfections cause USB power readings taken from the same device to vary across different machines. With this we are able to create high-resolution fingerprints based on voltage, amperage, and wattage provided by a host measured using a device connected to one of their USB ports. In this work we demonstrate the practicality of this method and its strengths over previously proposed methods of USB Fingerprinting [?].

Making use of machine learning classifiers, we show that through the collection of power measurements we are also able to accurately differentiate hosts based on operating system, model, and manufacturer. Taking this further, we found that our classifier was able to uniquely identify host machines with a nearly equal level of accuracy. Through our machine identification trials we found that with our USB Fingerprinting method our classifier could construct a set of models that consistently identifies 94% of our corpus of 45 machines. Additionally we found that the classifier was able to identify 98% of a field of 10 seemingly identical machines from our corpus.

This thesis makes the following contributions:

– **A methodology for feature extraction of USB power measurements:** We collect measurements containing voltage, amperage, and wattage via a small,

inexpensive embedded device. From there we develop a feature extraction methodology that we apply to 22,500 measurements. We conduct a survey across several university computer labs of USB stack behavior yielding a data set of 45 different machines with a few different makes, models, and operating systems. We quantify and express the information gain of resulting feature vectors across our sampled machines.

– **Design and evaluation of USB-Power-based host classification techniques:** We use USB power output data to differentiate between different machine models with 98% accuracy and between different operating systems with 98% as well. Through the use of statistical techniques we refined classification results for individual machines and achieve a fingerprint that can uniquely identify 98% of a field of 10 seemingly identical machines. We analyze our methods used for classification along with the information gain of elements in our feature vectors in order to produce a more accurate result.

– **Development of collection tools suitable for commodity deployment:** Previous works were able to eliminate the need for expensive USB analyzers in favor of commodity smartphones [**?**]. We demonstrate an alternative approach that uses even less expensive hardware (around $10 worth). We develop a method for collecting power samples using the even less expensive power gauge mini-kit and a USBTinyISP AVR Programmer produced and sold by Adafruit. We use the fact that the power gauge is open source and explain how we used this to further refine our measurement collection process.

The rest of this thesis is structured as follows: Section **??** provides the details of how we collected power measurements and extracted feature vectors with statistical

tools for better classification results. Section **??** describes our classification techniques and results.

CHAPTER II

METHODOLOGY

We begin by observing that the USB interface is now nearly ubiquitous across both consumer and enterprise devices (e.g. desktop computers, laptops, mobile phones, etc). Hundreds of millions of people rely on devices which communicate via USB on a daily basis. Throughout this work it will become clear that the ubiquitous nature of USB devices works towards our advantage as we describe a method for host identification via USB fingerprints.

Previous works in USB fingerprinting relied extensively on custom hardware, expensive USB analyzers and Android smartphones for fingerprint collection [**?**]. These works were unable to consistently identify machines with greater than 90% accuracy. In this study we propose an even lower cost solution which has proven to provide more accurate results than previous works. To do this we examined variance in the power output of USB ports belonging to different hosts. In particular, we used voltage, wattage, and amperage to form unique USB fingerprints for individual hosts.

2.1 Why Power Readings

In order to create high-resolution host fingerprints we used a novel approach. By collecting power measurements commonplace to Electrical Engineers (EEs) and other EE related fields, we were able to form samples composed of power readings that were viable for host identification. After some pre-processing, we fed these samples into our machine learning classifier in order to form these high-resolution host fingerprints. This was possible in part because of the identifying information leaked through USB power readings.

With the findings of our work we assert that the information leaked by a host via power analysis is sufficient to be considered uniquely identifying. Although the USB specification calls for USB hosts to be provisioned with a downstream port that applies 5 volts to a connected device, it mentions that there is room for imperfections. According to the specification, an applied voltage anywhere between 4.75-5.25 is considered tolerable for a downstream port when a device is connected [**?**]. In practice, mechanical and other imperfections give rise to subtle variations in applied voltage and amperage from a connected source that can be observed from outside of the host as a side channel [**?**], [**?**], [**?**].

Each measurement we collected contained voltage, amperage, and wattage values read from the machine being fingerprinted via a low cost power gauge. Voltage was particularly useful for us during classification because different machines typically had consistent voltage output. In addition to this consistency, we found each sampled machine to have values that varied enough from the other machines to be accurately identified by our machine learning classifier with a high success rate. Moreover we found that many tuples of (voltage, amperage, wattage) measured from the same machine were unique to that machine.

Amperage tended to be much less consistent than voltage on a per machine basis. We measured the average variance per machine for voltage to be $1.06 * 10^{-4}$ volts versus the average variance for amperage, $4.25 * 10^{-2}$ amps, nearly 400x greater. Based on our preliminary tests, it seemed that voltage would be more useful for classification. Partly because of the lack of consistency of amperage values we measured, we measured power as well. By observing that power is directly related to both amperage and voltage via $P = IV$ (P is power measured in watts, I is current measured in amperes, and V is voltage measured in volts) we can see that we

have increased chances of each measurement taken with our power gauge being in a consistent ballpark with others taken from the same machine (power hovered around the measured voltage despite less consistent amperage readings).

2.2 Implementation

By using a $10.00 *power gauge mini-kit* developed by Adafruit [**?**] which includes a multipurpose Atmel ATtiny85 microcontroller we were able to take power readings across 45 different (but in some cases identically specified) machines. Before getting started with the power gauge we assembled and used a $22.00 *USBTinyISP AVR programmer kit* [**?**] also sold by Adafruit to change code on the gauge. The Adafruit power gauge was specifically purposed for the task of collecting measured voltage, amperage, and wattage and outputting these measurements via TTL serial output [**?**]. The power gauge arrived unassembled so it was necessary to solder the male and female USB connectors prior to taking measurements. In the next subsections we describe how we used the AVR programmer to load code onto the power gauge in further detail.

2.21 Using the Programmer

In this section we describe the Adafruit USBTinyISP AVR Programmer used in this study. With the programmer we were able to load altered source code onto the power gauge which allowed us to get more decimal points of precision from our measurements and to double the sampling rate. The Programmer was assembled from the *USBTinyISP AVR Programmer Kit* sold by the company, Adafruit (mentioned above). The kit arrived as a circuit board, case, series of resistors, micro controller, capacitor, buffer chip, and LEDs. From there we spent 3-4 hours soldering

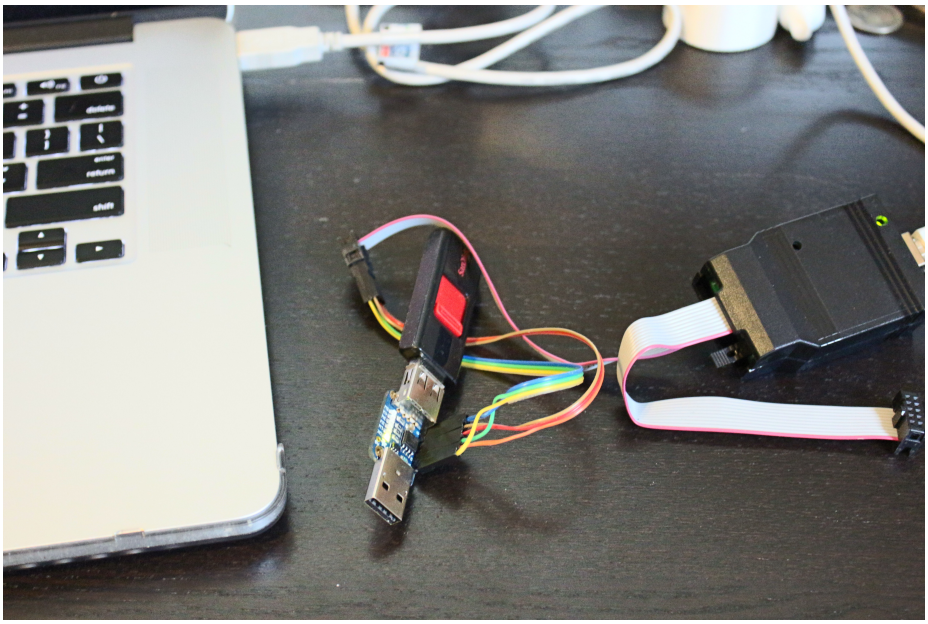

FIGURE 2.1: An image of our setup while the programmer can communicate with the power gauge.

the components to the circuit board as described in the instructions provided on Adafruit's support website. Once the Programmer was assembled we verified that it worked correctly by plugging it into a Linux machine with the supplied USB cable and observing that the green LED on the programmer was illuminated. We then loaded a sample blink sketch onto an Arduino UNO board using the programmer and a standard USB cable and saw that the specified LED blinked as intended.

Using the 6-pin connector included in the USBTinyISP Programmer Kit, we connected the programmer to the USB power gauge (as shown in Figure **??**) in order to load our modified source code onto it. To initiate the loading process we used Arduino IDE with a few custom settings to communicate with the power gauge. For starters we had to update the file `avrdude.conf` with the necessary settings for an ATtiny85 microcontroller. Then because we used Arduino IDE on OSX it was necessary to update a faulty linker included with the Arduino IDE software to allow sketches

larger than 4KB in size to be uploaded to the power gauge [?]. There wasn't actually documentation for completing this process specific to the power gauge mini-kit, but we found that the provided instructions for the nearly identical, Adafruit Trinket microcontroller worked fine. After making these alterations, we launched Arduino IDE and selected the correct settings to communicate with both the programmer and the power gauge. In our case, these were (from the Arduino IDE menu bar) `Tools -> Board -> Adafruit Trinket 16 MHz` and `Tools -> Programmer -> USBTinyISP`. From here we were able to successfully upload our custom source code with precision and timing modifications to the power gauge.

2.22 Modifying the Power Gauge

Out of the box the power gauge did not offer the precision or sampling rate that we desired. Luckily the power gauge was created for makers and fully reprogrammable. The 10 sample measurement in Figure **??** gathered by the power gauge demonstrates its initial lack of precision. A mere single decimal point of precision caused too many shared voltage and wattage values. In particular we found that many machines shared the voltage values: 4.8, 4.9, and 5.0 volts. After previously examining the USB 2.0 Power Delivery specifications, we found these numbers to be unsurprising [?]. However this shortage of possible values would have been detrimental to our accuracy when feeding those values into our machine learning classifier down the road. To remedy this problem, we were tasked with increasing the precision of the power gauge. We found that the power gauge was actually capable of up to three decimal points of precision with a few slight adjustments to the source code loaded on the device.

```
V: 4.9 I:  39 mA Watts: 0.2
V: 4.9 I:  15 mA Watts: 0.1
V: 4.9 I:  15 mA Watts: 0.1
V: 4.9 I:  25 mA Watts: 0.1
V: 4.9 I:  27 mA Watts: 0.1
V: 4.9 I:  31 mA Watts: 0.2
V: 4.9 I:  18 mA Watts: 0.1
V: 4.9 I:  15 mA Watts: 0.1
V: 4.9 I:  27 mA Watts: 0.1
V: 4.9 I:  27 mA Watts: 0.1
```

FIGURE 2.2: Power gauge output before source modifications.

Originally the device collected voltage measurements in millivolts via a method called `readVCC()` and subsequently measure the current in milliamperes via a method called `readCurrent()`. Finally it would calculate the wattage using the equation $P = IV$ in milliwatts and divide this value by 1000 to convert it to watts. The power gauge would then pass the measured values for voltage and wattage to another method called `printDotDecimal(value, numDecimalPoints)` which would print the values with precision specified by the second argument. The reason this method is necessary is because the power gauge does not have enough onboard memory for the Arduino floating point library. The floating point library is around 2,000 extra bytes in size while the power gauge source uses 5,302 bytes out of 5,310 available without the library but with our modifications. Because amperage was displayed by the power gauge in milliamperes (as an integer) the standard Arduino `print()` method was used to print it.

Before being altered, the code on the power gauge would call `printDotDecimal` using the result from `readVCC()` (the measured voltage in millivolts) as the first argument and using the integer value `1` as the second. This would only output the voltage value in volts to one decimal despite the fact that the code clearly supports

```
1     void printDotDecimal(uint16_t x, uint8_t d) {
2       ss.print(x/1000);
3       if (d > 0) {
4         ss.print('.');
5         x %= 1000;
6         ss.print(x / 100);
7       }
8       if (d > 1) {
9         x %= 100;
10        ss.print(x / 10);
11      }
12    }
```

FIGURE 2.3: Unaltered printDotDecimal function used in place of the Arduino floating point library to print floats.

two out of the box. Using the AVR programmer and Arduino IDE we changed the second argument to `2` and loaded the code onto the power gauge. After this change succeeded, we decided that two decimal points was an improvement but three would be even better. To accomplish this we made a few additions to the `printDotDecimal` method above. In the original method, the input value `x` (millivolts or milliwatts, both of which were typically greater than 100 but less than 6,000 in our case) and at most (when the second argument is `2`) printing the digit in the 10^0 column, 10^{-1} column, and 10^{-2} column while throwing out the digit in the 10^{-3} column. To add this 10^{-3} decimal digit to the output we appended our new code shown in Figure **??** after the closure of the last `if` statement in the original method.

After this minor change we successfully collected and displayed voltage and wattage measurements with three decimal points of precision using `printDotDecimal(vcc, 3)` and `printDotDecimal(watt, 3)` respectively in the main `loop`.

Despite having higher precision from our new additions to the code loaded onto the power gauge we encountered another issue. The original source code took samples at a painfully slow rate. Prior to making any modifications, the code loaded onto the

```
1         void printDotDecimal(uint16_t x, uint8_t d) {
2           ss.print(x/1000);
3           if (d > 0) {
4             ss.print('.');
5             x %= 1000;
6             ss.print(x / 100);
7           }
8           if (d > 1) {
9             x %= 100;
10            ss.print(x / 10);
11          }
12          // Print a third decimal point if d >= 3.
13          if (d > 2) {
14            x %= 10;
15            ss.print(x);
16          }
17        }
```

FIGURE 2.4: printDotDecimal with our additions to support 3 decimal points.

device took over 15 minutes to collect 50 samples from a machine. In order to reduce the amount of time necessary for data collection we found it necessary to increase the sampling rate of the device. To do this we took a look into the main `loop` of the original code described in Figure **??**.

We noted that the variable `printCounter` dictated how fast the loop would iterate over the block of code responsible for measuring and printing the power readings. We then effectively doubled the sample rate by replacing

```
1     printCounter %= 10;
```

with

```
1     printCounter %= 5;
```

after this adjustment, it took only around 7 minutes and 30 seconds to collect 50 samples from a machine. This meant we could take samples 2x faster, a considerable speedup. We decided not to increase the sampling rate any further in order to retain some accuracy during the measurement taking process. Despite this we note

```
1 void loop() {
2   ...
3
4   if (printCounter == 0) {
5     while (Lidx != 0) {}
6     //  TIMSK &= ~_BV(OCIE1A);
7     printStringDelay("\n\rV: "); //ss.println(vcc);
8     vcc += 50; // this is essentially a way to 'round up'
9     printDotDecimal(vcc, 1);
10    delay(50);
11    printStringDelay(" I: ");
12    if (icc < 100) ss.print(' ');
13    if (icc < 10) ss.print(' ');
14    ss.print(icc);
15    printStringDelay(" mA ");
16    delay(50);
17    printStringDelay("Watts: ");
18    watt += 50; // this is essentially a way to 'round up'
19    printDotDecimal(watt, 1);
20    delay(50);
21  //  TIMSK |= _BV(OCIE1A);
22  }
23  printCounter++;
24  printCounter %= 10;
25  ...
26 }
```

FIGURE 2.5: Main loop function loaded on the power gauge. While the device is on, this function is called in repetition indefinitely.

that accuracy was not our prime concern. As long as the power gauge produced precise measurements, the measured voltage, amperage, and wattage were not that important. This means to say that if the power gauge were to incorrectly offset something such as the measured voltage by +.2 volts, this was okay as long as this offset occurred for every measurement.

2.3 Collecting Samples

In order to get the power gauge to start enumerating measurements, it was necessary to plug a USB compatible device into the device side shown in Figure **??**. Our first attempt was to use a USB 2.0 to Micro B cable with an Android smartphone

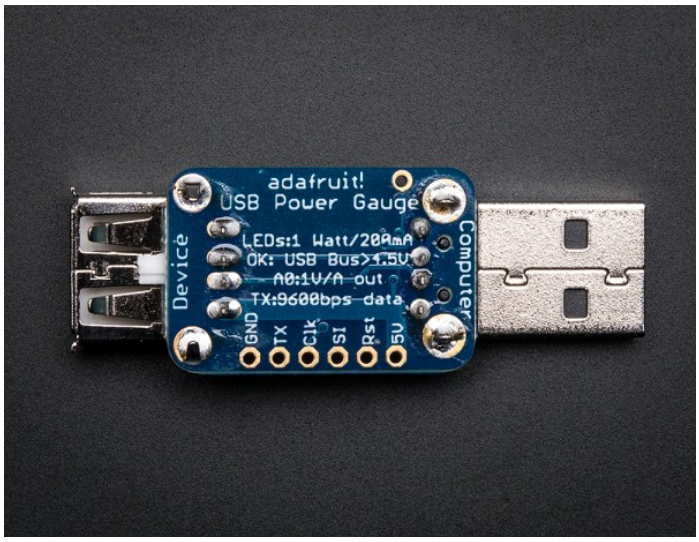

FIGURE 2.6: An image of the power gauge we used (provided by Adafruit).

(in this case, the Samsung Galaxy S4) connected to the Micro B side. We found that the smartphone would not always negotiate for the same amount of voltage and current during the enumeration step of the USB connection. As a result voltage readings jumped at times by over a volt during different fingerprinting instances of the same machine. To remedy this approach, we plugged a simpler flash drive (SanDisk Cruzer 16GB USB 2.0) into the device side instead. After making this modification, voltage measurements from the power gauge no longer displayed this behavior.

To be able to read and record output from the power gauge we used *Adafruit's USB to TTL serial cable* in conjunction with the command line tool, screen. After plugging in the power gauge to a host we wanted to sample, we would connect the `GND -> GND` and `RX -> TX` pins respectively from the cable to the power gauge. In order to get more consistent results, each of the samples we took was collected a few seconds after a fresh reboot before running any non-system related programs or logging in.

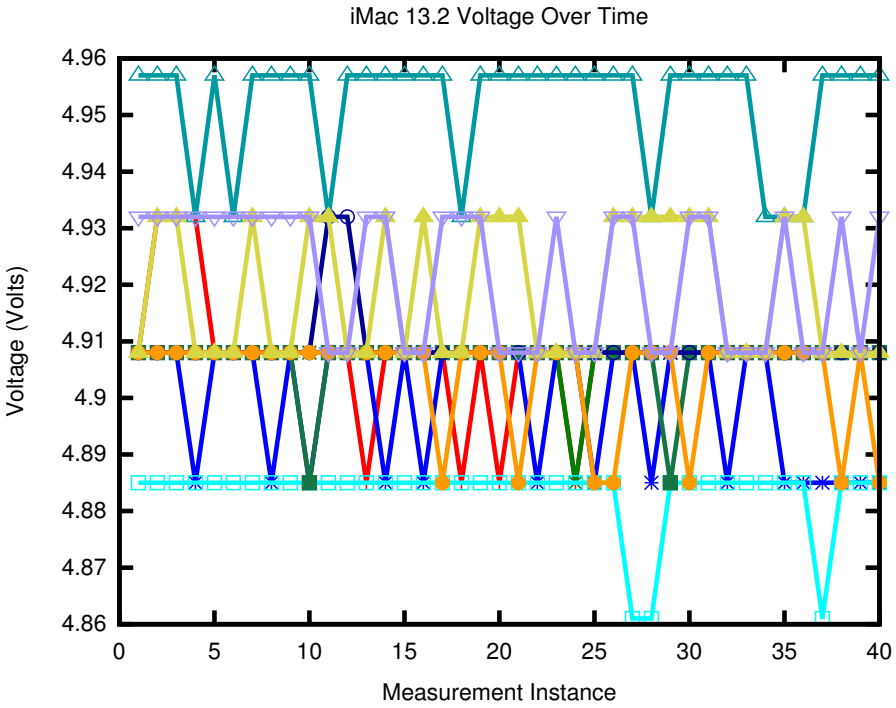

FIGURE 2.7: Voltage over time from raw measurements of iMac 13.2's in our data corpus.

To fetch the output we were interested in from screen, we used the command `screen -L /dev/tty.NoZAP-PL2303-0000X014 9600,cs7` where `X` varied from 1 to 2 depending on which USB port the serial cable was plugged in to. The command created a file called `screenlog.0` in the working directory which contained power measurements from the sampled host. After sampling a single host, we renamed this file to adhere to the following pattern: `location_MachineIDModel#OperationSystem` so that we could later conveniently process these files. For example, a raw sample file for a iMac 12.2 in Deschutes was called `des_WhoaiMac12,2OSX10,8,5`.

Each sample we took was comprised of raw 10 measurements. In preliminary testing we observed that amperage and wattage readings had high variance and would likely not be useful in classification (contrasting voltage as shown in Figure **??**, voltage

typically varied between 2-3 values per machine). In an attempt to reduce variance, we arranged to have each sample to be comprised of the maximum likelihood estimations (MLEs) of raw measurements. We collected 50 samples at a time from each distinct machine. Even with our custom increased sampling rate, taking 500 measurements with the power gauge took on average 7 minutes and 30 seconds per machine. In total we spent over 5 hours and 37.5 minutes collecting samples. Prior to sampling each of the hosts, we recorded a few identifying characteristics about the machine including `OS name` and `version number`, `USB Firmware version number` and `USB Controller Manufacturer` as well as `USB Driver Version` into an excel spreadsheet for use later with our machine learning classifier.

To recap, the process for data collection was as follows:

1. Record identifying attributes about the target machine.
   (a) Machine ID.
   (b) Machine Model (w/ Model Number).
   (c) Operating System (w/ OS Version).
   (d) USB Hardware Manufacturer (w/ Firmware Version).
   (e) USB Driver Version Number.
2. Reset the target machine.
3. Allow the target to reach the login screen.
4. Plug in the host side of the power gauge into the target.
5. Using the USB to TTL serial cable connected to the power gauge, plug in the USB cable into the sample collecting machine.

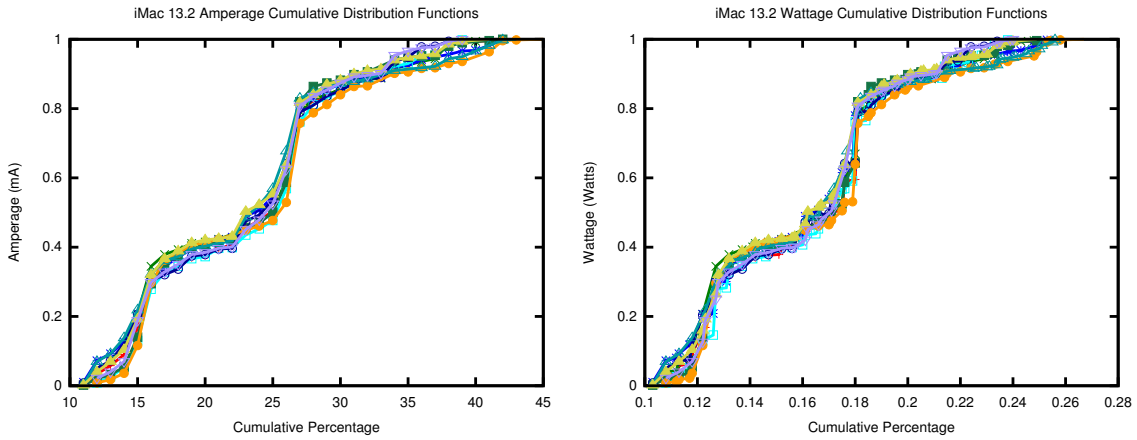

FIGURE 2.8: Cumulative distribution functions for amperage and wattage measurements across all iMac 13.2 hosts.

6. Run the screen command mentioned above.

7. Let the power gauge collect samples via screen for 7 minutes and 30 seconds.

8. End the screen process and rename the output file (screenlog.0) using the format mentioned above.

The procedure was used in order to eliminate potential outside variables that could possibly affect our results. Future work remains to determine whether this approach is robust when the machine is either under load or operating with more (or less) plugged-in USB devices than a mouse and keyboard. Future work is also required to determine whether the power gauge is precise enough to regularly gather consistent measurements from the same host.

2.4 Data Corpus

By using the procedure detailed in the section above, we collected samples from a variety of university owned machines. These machines were located across four different computer labs on the University of Oregon campus. The labs were by

in large homogenous typically consisting of either one or two groups of machines with identical specifications. Many of these nearly identical machines were also running the same operating system installed from a shared disk image. Our data set contained samples from 30 hosts running Microsoft Windows 7 Service Pack 1 and 15 hosts running Apple's OS X 10.8.5. Our corpus is described in more detail in Table **??**. In total we collected 500 measurements per machine for a total of 22,500 measurements taken which constituted 2,250 samples. We collected additional measurements from machines that we had already fingerprinted to ensure the measurements were consistent with ones we had already collected. We did not include these additional measurements in our data corpus. Our work suffers from a relatively small data set but the results we found proved to be promising for future work.

Class	**Label**	**Host Count**
OS		
	OSX 10.8.5	15
	Windows 7 SP1	30
Model		
	Apple iMac 12.2	5
	Apple iMac 13.2	10
	Dell Optiplex 980	13
	Dell Optiplex 990	13
	Dell Optiplex 9010	4
	TOTAL MACHINES INSPECTED	45
	TOTAL MEASUREMENTS COLLECTED	22,500

TABLE 2.1: Description of data corpus.

2.5 Feature Extraction

After collecting raw samples from our corpus of machines, we used a custom ruby script to parse out meaningful data. Our raw data consisted of a 3-tuple per measurement: the measured *voltage*, *amperage*, and *wattage* of the target host (see Figure **??** for an example raw measurement). Voltage and wattage were recorded in volts and watts respectively with three decimal points of precision. Amperage was recorded in milliamperes with three digits of precision. To get a better representation of the raw data, we constructed a per-sample feature vector of statistical metrics that we later used as input to our machine learning classifier. We repeated these metrics for each of voltage, amperage, and wattage. In total our feature vectors were formed of 9 distinct features that proved to be great for consistently accurate classification of our sampled hosts.

We chose our first metric included in the feature vector to be mean. Because measurements from a single machine typically jumped around a lot we found an average of each of the quantities being measured to be a useful tool. Our feature vector contained mode as well, this was especially useful when considering voltage because each machine typically varied over only 2-3 unique values for voltage during sample collection. Given this consistency, mode was a great feature for our classifier. Lastly we measured standard deviation of each voltage, amperage, and wattage to get a snapshot of how much values from a single machine varied. We found standard deviation to be beneficial when classifying machines whose values varied more from the average than other machines.

We utilized a custom ruby script to automate the process of computing and aggregating these feature vectors for each of our 2,250 samples and then exported them to a single `.csv` file. The format of the feature vector extracted from each

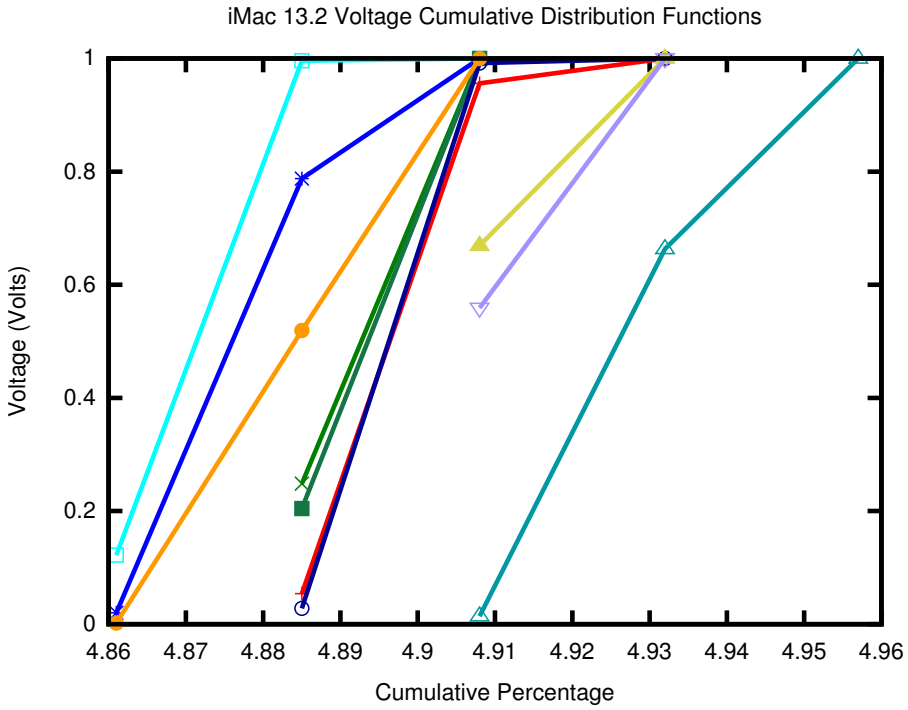

FIGURE 2.9: Cumulative distribution functions for voltage measurements across all iMac 13.2 hosts.

sample is as follows: `Machine ID,Voltage Mean,Voltage Mode,Voltage Standard Deviation,Amperage Mean,Amperage Mode,Amperage Standard Deviation,Wattage Mean,Wattage Mode,Wattage Standard Deviation`. For the purposes of creating a meaningful visual presentation of our data, we also wrote a bash script to parse the .csv output of our ruby script and generate several GNUPlot scripts. GNUPlot is a portable, command line graphing tool commonly used for data visualization in the academic community.

Figure **??** shows that by visual inspection, voltage was for the most part distinct across different machines. We expected voltage to be more useful during classification than amperage and wattage because of the variance shown in Figure **??**. Despite this, calculated information gain showed that this variance could actually be useful.

Feature	**Information Gain**
Voltage Mean	3.88 bits
Wattage Mode	3.02 bits
Voltage Standard Deviation	2.27 bits
Amperage Standard Deviation	2.25 bits
Voltage Mode	2.19 bits

TABLE 2.2: The 5 highest ranked features by information gain.

Voltage mean had the highest information gain as expected but wattage and amperage standard deviation made the top 5 as well as shown in Table **??**.

CHAPTER III

EVALUTATION

In this section we further explore the discriminating potential of USB power readings. We do so by employing several machine learning classifiers which attempt to identify where input samples originated after examining a training set. Using these classifiers, we took a look at different defining characteristics of the machines in our data corpus and whether the classifiers were able to accurately distinguish them. We used a subset of the stock set of classifiers provided by the well-respected WEKA libraries to accomplish this. Lastly we seek to develop a fingerprint that will uniquely identify individual hosts amidst other identically specified machines.

3.1 Classification Algorithms

Building off the findings of previous works [**?**] we found supervised learning algorithms such as decision tree classifiers to be particularly accurate and robust when paired with our data set. Supervised learning algorithms analyze training data instances (each containing a vector of attributes and a class label) to construct an inferred function which can be used for mapping new data instances. When talk about the *accuracy* of these supervised learning algorithms, we are referring to the percentage of unseen instances which whose class labels were correctly determined by the classifier.

We experimented with a number of different classifiers provided by the WEKA libraries in an ad hoc fashion. These classifiers included J48 Decision Trees, Random Forest, Random Tree, and Instance-Based Learners (using a nearest neighbor

OS Version	Accuracy
OSX 10.8.5	98%
Windows 7 SP1	99%

TABLE 3.1: OS Version accuracies by class label.

MNF	Model	Number	Accuracy
Apple	iMac 12.2	OSX 10.8.5	96%
Apple	iMac 13.2	OSX 10.8.5	97%
Dell	Optiplex 980	Windows 7 SP1	99%
Dell	Optiplex 990	Windows 7 SP1	98%
Dell	Optiplex 9010	Windows 7 SP1	99%

TABLE 3.2: OS Model accuracies by class label.

algorithm). Ultimately we found the Random Forest classifier to yield the highest accuracy although it only slightly edged out WEKA's IB1 classifier.

3.2 Results

We began by using our classifier to distinguish machine operating system. We found that the classifier generated a model which correctly identified the operating system of machines in our data corpus with 99% accuracy (as shown in Table **??**). Admittedly this number might be overly optimistic due to the homogeneity of OS versions of the machines in our data corpus.

We then turned our attention to classification using machine model as the class label and found some particularly interesting results. In addition to 98% accuracy when distinguishing host models as shown in Table **??**, we found our classifier only mistaken 9 out of 1,500 sample instances of Dell/Windows hosts for iMac/OSX hosts. The classifier did slightly worse when identifying iMac/OSX hosts mistaking 13 out of 750 instances for Dell/Windows machines. Overall the classifier was very effective in identification with machine model as the class label.

3.21 Machine Identification

Lastly we put our Random Forest classifier to the test using Machine ID as the class label. We found the model generated by the classifier to yield even better results than previous works [**?**] with 94% accuracy when distinguishing hosts in our data corpus. Such a high accuracy percentage attests to the viability of using our power measurement methods to form host fingerprints. Additionally, in a field of 10 identically specified machines our classifier achieved an accuracy of 98%.

CHAPTER IV

DISCUSSION

In order to promote and advance the techniques of USB Fingerprinting via power readings, we released our source code and instructions on how to load it to the power gauge we used (see Section **??**). Although our exact approach requires specific hardware, we believe our methodologies can be revised and deployed effectively on any device that is capable of measuring USB power output in a precise manner.

4.1 Analysis

In order to determine the robustness of our approach we found it necessary to take a closer look at host power output while under varying loads. It would seem natural that with more connected power delivery (PD) devices, a host would output less power through the power gauge than it would with no other connected USB devices. To test this assumption, we took 50 samples containing 10 measurements each from a machine in our data corpus with the following load variances (for each of these we used the procedure outlined in Section **??** to collect measurements):

- **No Load**: Before resetting the machine we disconnected all USB devices from the machine until sample collection was finished.
- **Regular Load**: We reset the machine with the mouse and keyboard connected and collected samples as usual.
- **Load**: Before resetting the machine we connected an uncharged Android smartphone to the target host along with the mouse and keyboard to the target host until sample collection was finished.

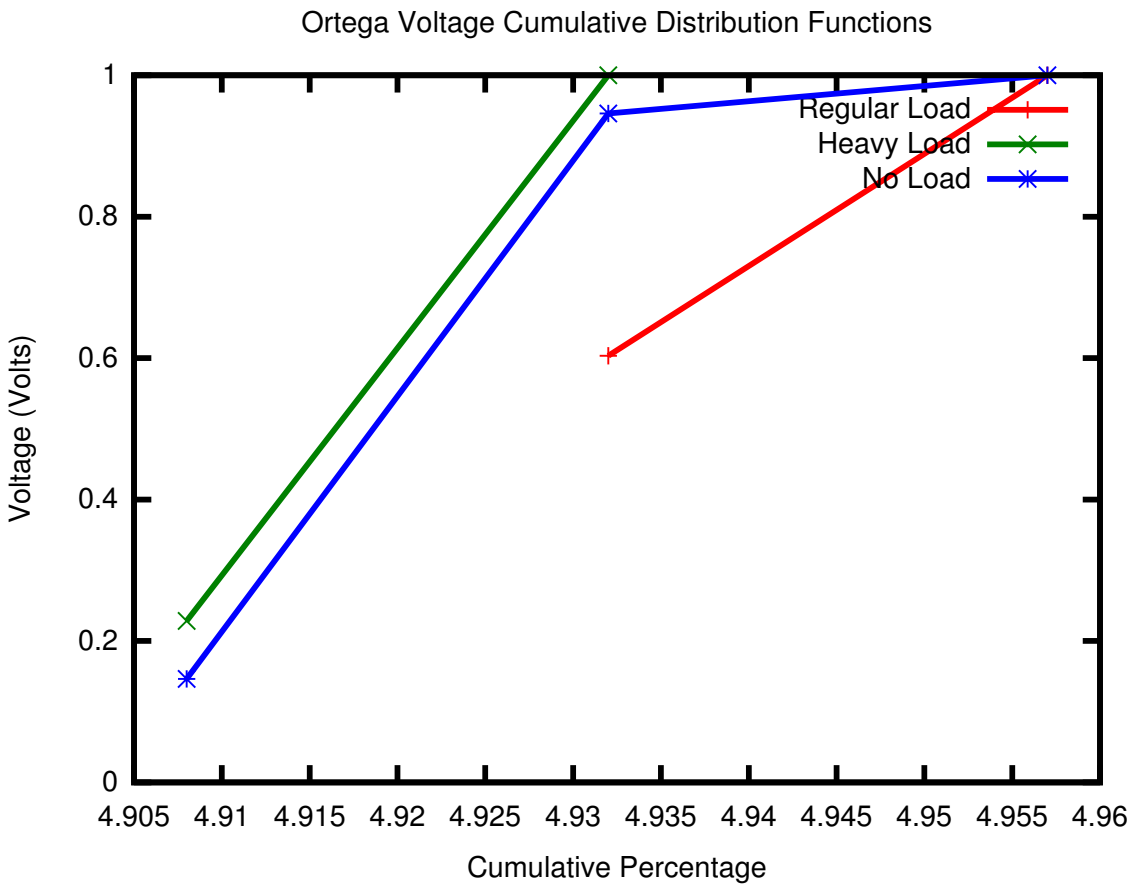

FIGURE 4.1: Voltage cumulative distribution functions of a single iMac 12.2 named Ortega.

What we found was unsurprising in some ways. The voltage provided by the target host while the machine was under load was on average lower than the voltage provided when the machine was under no load. This can be seen in more detail in Figure **??**. Despite this result, the "regular load" samples we took with our original batch of samples proved to be outliers. We believe this possibly to be due to temperature differences during the times that we took measurements (the original batch of samples was collected during a warmer day than the load samples). On their website, Adafruit mentioned that measurements from the low cost power gauge are susceptible to a small amount of noise due to thermal changes [**?**].

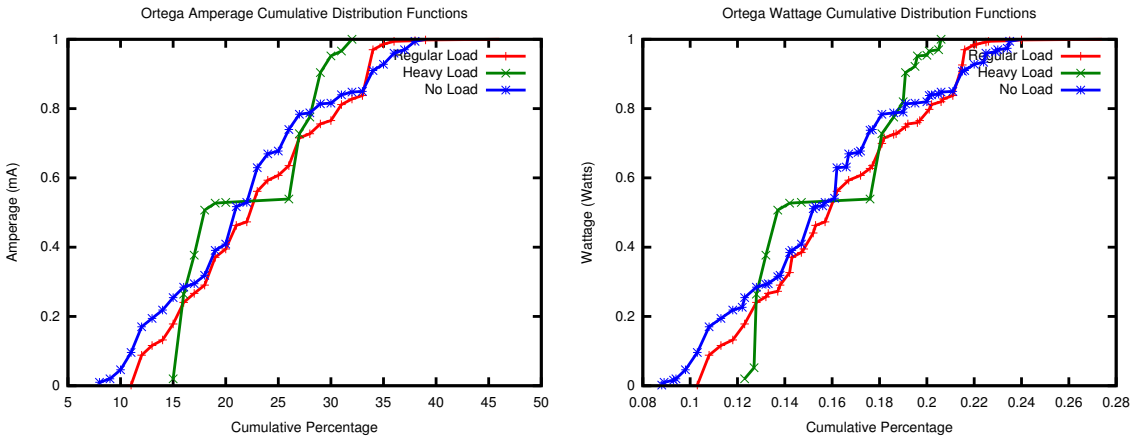

FIGURE 4.2: Cumulative distribution functions for amperage and wattage measurements on Ortega host with three load variations.

Class Label	**Accuracy**
Ortega Heavy Load	100%
Ortega No Load	98%
Ortega Regular Load	96%

TABLE 4.1: Identification accuracy under varied loads of the machine, Ortega.

While the smartphone was connected to the target host (labeled as "heavy load"), the power gauge measured the amperage and wattage provided by the host to be more consistent. This can be observed in Figure **??**.

When we piped the data we collected under varied load to our machine learning classifer described in Section **??**, we found that our classifier was successful in identifying each of the variations. Our classifier consistently identified all but 3 samples from a data set of 150 samples collected while the machine was under a varied load. As shown in Table **??**, our classifier was able to distinguish between the three host states with an upwards of 99% accuracy. These results attest to the possibility of using the fingerprinting methods described in this work to not only profile machines, but also to profile the devices connected to the machines.

In this case of varied load on a single target host, amperage mode and wattage mode proved to be better metrics for our machine learning classifier than voltage mean. The information gain of each of these attributes can be seen in Table **??**.

Feature	**Information Gain**
Amperage Mode	1.55 bits
Wattage Mode	1.36 bits
Wattage Standard Deviation	1.32 bits
Amperage Standard Deviation	1.32 bits
Voltage Mean	1.27 bits

TABLE 4.2: The 5 highest ranked features by information gain for Ortega samples with varied load.

Despite this result, voltage mean was again a useful metric for classification and still showed up in the 5 highest information gain attributes.

4.2 Future Work

While highly accurate, the work presented in this thesis will require a larger, more diverse data corpus for the results of our classification trials to be effectively validated. Ideally a larger data set containing measurements from several more homogeneous computer labs will continue to show even more promise for USB Fingerprinting via USB power readings. Although our feature vectors proved to be sufficient for the classification trials we conducted, it would undoubtedly increase the accuracy of our machine learning classifier to add more features.

4.21 Measurement Tools

One drawback to our approach is that the power gauge we used is not a multimeter replacement [**?**] and therefore might not be as accurate or precise as we need. Additionally the maximum amount of decimal points the device is capable

of collecting and reporting when taking power measurements is three. This leaves only around $2 * 10^3 = 2000$ possible values for voltage and wattage (this is a rough estimate taking into consideration that these values varied between 4-5 volts and 0-1 watts respectively). Considering the number of PCs in 2013 was estimated to be more than 1.5 billion [**?**], more decimal points of precision will be necessary in order to to avoid multiple machines sharing the same or a very similar fingerprint (because of our feature extraction procedures, the chance of this happening is greatly reduced but still possible).

4.22 Measurement Consistency

Due to time constraints, we were unable to determine whether the USB port used to plug in our power gauge made any difference in the measurements we collected. Despite the fact that different ports from the same host shared the same USB stack (hardware, firmware version, driver version), one could postulate that slight imperfections in each port could cause unforeseen variance in the measurements taken. In light of this concern, we did return to a machine we had collected samples on during a different day to collect and compare measurements. From visual inspection it seemed that these measurements were consistent but future trials measuring the variance in power readings over different USB ports on the same host would make this concern more transparent.

CHAPTER V

RELATED WORK

Fingerprinting has been employed as a device identification method using various different approaches. Some of these methods have been used to identify devices on a WiFi network [**?**], internet users' browsing habits (using attacks resistant to celebrated obfuscation platforms such as https and Tor) [**?**], [**?**], and of course hosts connected to another device via a data transfer medium such as USB [**?**]. Fingerprinting works because of information leaks (or side channels) resulting from either measurable signals caused by hardware imperfections in analog circuitry or visible usage patterns (such is the case in website fingerprinting mentioned above) to uniquely identify devices. Other related works include attempts to subvert machine learning classification techniques such as the ones used in this thesis to identify machines and their characteristics [**?**].

5.1 USB Fingerprinting

Adam Bates et al. used timing data they gather from low level USB interactions to form uniquely identifying host fingerprints. Their work was shown to be resistant to concept drift and relay attacks while being employable for a low cost. Despite the benefits of their approach, the paper's reported host identification accuracy was between 53%-80%. We were able to improve upon their results with our average accuracy of 94% when identifying randomly picked hosts out of our corpus. In order to have confidence that a machine has been correctly identified, methods of USB Fingerprinting must be refined so that an even higher identification accuracy may be reached.

5.2 Remote Fingerprinting

Remote fingerprinting schemes identify devices via web traffic analysis [?], [?] and metadata leaked via characteristics of their communication methods (such as browser plugins, fonts installed, etc) [?]. Although the accuracy of these methods is celebrated, network fingerprinting techniques are easily fooled by systems who spoof their visible characteristics at a network level, such as Honeyd [?] or user agent spoofing browser extensions [?]. Another approach focuses on timing analysis of 802.11 probe request frames to create unique fingerprints for devices connected to a WiFi network. This method suffers from a significantly lower accuracy than the previous and is further decreased when target devices are further from the WiFi access point.

5.3 Learning Based Classifier Evasion

Methods proposed by Laskov et al. suggest ways to subvert machine learning classifiers such as the ones employed by this thesis. In their study, they tested the publicly available PDFRate (a machine learning classifier based program to determine if a PDF is malicious or not) to demonstrate the viability of arbitrarily altering confidence scores of a machine learning classifier [?]. Their goal was to show that a skilled adversary could confuse the classifier into giving a false classification. They found that by altering a small subset of features the machine learning classifier considered, they were able to take a dataset of PDFs which was previously classified by PDFRate as malicious and trick the classifier into thinking that 75% of the files were actually benign. Their collection of attacks relies on the ability to alter data before classifying it. Because of this our proposed methods of USB Fingerprinting are

safe from these learning based classifier attacks unless the data collected is altered by an adversary before being submitted to our machine learning classifier.

CHAPTER VI

CONCLUSION

In this thesis we created a viable USB Fingerprinting technique for uncovering uniquely identifying information about USB hosts using power measurements. We showed that other information such as operating system and model can also be extracted from our USB Fingerprinting technique with an upwards of 99% accuracy. We showed that this information extraction is possible using just a few inexpensive devices and a publicly available machine learning classifier with minimal configuration. We achieved an accuracy of 94% when uniquely identifying machines (some seemingly identical with matching specifications) from a large data corpus. This high accuracy was an improvement from previous works that used similar machine learning classifier USB Fingerprinting methods.

This thesis lays foundation for more rigorous work in fingerprinting using USB power readings. It also suggests the possibility of using this type of fingerprinting with other power providing interfaces such as phone connectors or Apple's Thunderbolt. Future work will involve validating USB Fingerprinting via power readings by incorporating larger, more diverse data corpuses and automating data collection. Future work remains to determine whether USB Fingerprinting can be used to detect attacks on a target host and whether it is resistant to attacks itself.

REFERENCES CITED

[1] Adafruit. Adafruit power gauge mini-kit product page. http://www.adafruit.com/products/1549, June 2014.

[2] Adafruit. Adafruit usbtinyisp avr programmer kit product page. http://www.adafruit.com/product/46, June 2014.

[3] Adafruit. Setting up with arduino ide for attiny85 boards. https://learn.adafruit.com/introducing-trinket/setting-up-with-arduino-ide, June 2014.

[4] A. Bates, R. Leonard, H. Pruse, K. Butler, and D. Lowd. Leveraging usb to establish host identity using commodity devices. *University of Oregon, Tech. Rep. CIS-TR-2013-12*, 2013.

[5] H. Blodget. The number of smartphones in use is about to pass the number of pcs. http://www.businessinsider.com/number-of-smartphones-tablets-pcs-2013-12, December 2013.

[6] M. contributors. Universal serial bus revision 2.0 power delivery specification. http://www.usb.org/developers/docs/usb20_docs/, March 11 2014.

[7] C. P. T. D. Loh, C. Y. Cho and R. S. Lee. Identifying unique devices through wireless fingerprinting. In *Proceedings of the 1st ACM Conference on Wireless Network Security*, WiSec '08, pages 46–55, New York, NY, USA, 2008. ACM.

[8] J. R. R. P. R. Dakshi Agrawal, Bruce Archambeault. The em side—channel(s). In C. P. Burton S. Kaliski, çetin K. Koç, editor, *Cryptographic Hardware and Embedded Systems*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer Berlin Heidelberg, 2002.

[9] P. Eckersley. How unique is your web browser? https://panopticlick.eff.org/browser-uniqueness.pdf.

[10] P. L. Nedim Srndic. Practical evasion of a learning-based classifier: A case study. In *35th IEEE Symposium on Security and Privacy*, 2014.

[11] W. J. C. K. F. P. G. V. Nick Nikiforakis, Alexandros Kapravelos. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. https://seclab.cs.ucsb.edu/media/uploads/papers/sp2013_cookieless.pdf.

[12] B. Parno. Bootstrapping trust in a "trusted" platform. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Security*, pages 1–6, San Jose, CA, August 2008.

[13] S. M. Patrick McDaniel. Security and privacy challenges in the smart grid. *IEEE Security and Privacy*, 7(3):75–77, May 2009.

[14] N. Provos. Developments of the honeyd virtual honeypot. http://www.honeyd.org/, July 2008.

[15] B. M. G. Stefan Mangard, Thomas Popp. Side-channel leakage of masked cmos gates. In A. Menezes, editor, *Topics in Cryptology*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. RSA, Springer Berlin Heidelberg, 2005.

[16] C. TJ, Teresa. wikihow: How to change a windows serial number. http://www.wikihow.com/Change-a-Windows-Serial-Number.

[17] B. J. R. J. Xiang Cai, Xin Cheng Zhang. Touching from a distance: Website fingerprinting attacks and defenses. In *CCS '12: Proceedings of the 19th ACM Conference on Computer and Communications Security.*

[18] N. B. Xun Gong, Negar Kiyavash. Fingerprinting websites using remote traffic analysis. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2012.