AUTOMATING USB-BASED HOST FINGERPRINTING WITH EMBEDDED DEVICES

by

HANNAH PRUSE

A THESIS

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

June 2013

THESIS APPROVAL PAGE

Student: Hannah Pruse

Title: Automating USB-Based Host Fingerprinting with Embedded Devices

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

Eugene Luks	Chair
Kevin Butler	Advisor

Degree awarded June 2013

THESIS ABSTRACT

Hannah Pruse

Bachelor of Science

Computer and Information Science

June 2013

Title: Automating USB-Based Host Fingerprinting with Embedded Devices

Approved: ______________________________
Kevin Butler

It is important to be able to confirm a computer's identity when performing secure transactions. When plugging a USB device into a host, one needs to be sure that the machine is really what is expected, and not a malicious device answering in its place. This paper presents a method of fingerprinting a machine to determine its operating system and model based on the timing of the events in a transaction with a USB device. We chose USB for this task because it is ubiquitous–almost every device has a USB port. Additionally, USB is a master-slave protocol, which ensures that the device gets answers from the host. Unlike performing over a network, where it is difficult to determine which machine is answering, our scheme uses a direct connection to analyze the unique physical characteristics of the host. We show that small and inexpensive commodity devices, which have relatively low granularity in timing measurements, are sufficient for this task.

This thesis includes co-authored material.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

Determining the identity of a computer has become a problem of considerable importance, especially when trying to determine the trustworthiness of a machine. For example, protocols based on any form of trusted computing principles such as IMA [53] rely on identifying a host machine's trusted platform module (TPM). Within transactions between two machines, a malicious third party can intercept messages and answer in place of the intended target, often with aims of validating the target as something it is not. Such attacks are known as "cuckoo attacks" [47] and are frequently used against distance bounding systems [5] in attacks such as Mafia fraud, in which a man-in-the-middle adversary attempts to prove that it is in proximity to a legitimate prover, when it may not be. Computer forensics investigators constantly rely on computer identity in criminal cases and in defending against cyber crime.

This research focuses on developing a scheme to *fingerprint* a given machine and to determine its operating system and model. Since the USB interface is virtually ubiquitous, and is thus available on almost every computer and device, we wish to use USB communication as a means to identify a machine. Various operating systems and system hardware (i.e., USB busses) can have an influence on the speed and types of events present in any given USB transaction, and thus it may be possible to discover patterns or classes of attribute durations and sequences that define a particular type of machine. Unlike previous fingerprinting schemes, such as wireless device identification [31], Xprobe [56], and Nmap [38], we use a physical connection to identify hosts, rather than relying on visual channels [39] or remote network monitoring, which can yield a quicker result and more accurate results. The combination of software variation amongst USB stacks, differences in USB host controllers, and variations in manufacture of the bus, chipsets, and other hardware portions allow us to *fingerprint* machines based on the timing of USB messages they send to a device connecting to them. Consequently, we call our technique *USB Fingerprinting*, and demonstrate its practicality and efficacy.

While USB analyzers can be used to get fine-grained information to better make these decisions, they suffer from being bulky, expensive, and exotic. To make USB Fingerprinting broadly adoptable, we demonstrate that a USB mass storage device running a modified Linux kernel can be used to effectively and automatically perform the collection of USB data. Such a device is small and inexpensive, and feasible for everyday users.

Thesis Statement: Host fingerprinting can be performed on small, cost-effective embedded flash drives.

The basic idea behind the fingerprint method is to use USB event information collected from several types of machines to train a classifier, and then identify hidden patterns within the data that would be indicative of a particular make or model of machine. The model resulting from the classifier learning process can then be used to identify USB information that had previously been unseen. Various classifiers were used in the development of our work, but we found decision tree-based classifiers to perform most effectively. Based on this framework, we collected over 3750 traces of USB operations from over 75 different machines, representing a diversity of makes, models, and operating systems. Through the use of feature extraction and machine learning classification techniques, we can differentiate between operating systems with 79.7% accuracy.

We show that our current data granularity is insufficient to differentiate between individual machines, but we propose alternate methods for retrieving and analyzing data that yields more accurate and reliable results. The limited granularity of the USB event messages from the device is a constraint that limits our accuracy. In future work, we hope to address this issue by recording events at a lower level and to consider sequencing as an attribute and expand possible applications for USB fingerprinting. The portion of this work in focus is the classification aspect, as well as the data collection device setup. Furthermore, we will only consider the timing of events as an attribute.

CHAPTER II

LITERATURE REVIEW

Fingerprinting

Fingerprinting has become a popular method for device identification, and has been used to identify home electronics [23], websites [10, 21, 37, 44, 25, 26], the operating system of virtual machines [22, 43], and the source of phone calls [4]. The concept of fingerprinting stems from leveraging measurable signals caused by hardware imperfections in analog circuitry to uniquely identify devices. Fingerprinting has been extensively studied and used for identifying RFID smart cards [13], Ethernet cards [20] and 802.11 devices [36, 17, 42, 8, 6], as well as users [45].

Remote fingerprint techniques identify devices using only characteristics of their communication [13]. Identifying machines remotely has been a popular method, resulting in tools like *Nmap* [38] and *Xprobe* [56] that detect operating systems by examining network traffic. While effective in some cases, network fingerprints can be fooled by systems that spoof operating systems at the network layer, such as *Honeyd* [48]. Other remote schemes, such as work by Kohno et al. [31] and Jana et al. [28], identify machines using clock skew data. However, it has been shown that TCP and ICMP timestamps can be disabled or manipulated [46, 40, 54]. Semi-persistent network data has also been used to fingerprint devices [27, 45] and browsers [15, 29]. Services to fingerprint browsers are available commercially, and are of particular interest to advertising agencies [3].

While many remote fingerprinting methods exist, there has been little previous exploration into host identification using USB traffic through a physical connection. A recent work by Wang et al. [55] uses USB-equipped smart phones to detect the host operating system; however, the authors do not achieve OS version-level granularity as we are able to achieve with our method. Butler et al. [32] present a USB fingerprinting method that feeds data from a USB protocol analyzer into a decision tree classifier. Our work obtains comparable accuracy while considering finer-grained machine attributes over a larger corpus of hosts, and provides further insight into the nature of the collected USB.

Machine learning classification techniques have been deployed to create accurate fingerprinting schemes for user re-authentication in smartphones. Li et al. [33] used feature extraction and SVMs to recognize an individual smartphone user's finger movements. Similarly, we leverage basic features to calculate additional attributes for more accurate classification.

Our approach offers several benefits over existing machine classification methods. Work proposed by Desmond et al. [36], can take one hour at a minimum to collect enough data to perform classification. With our scheme, we can collect data and make a decision in at most 4 minutes, and in as few as 12 seconds. Device identification work by Gerdes et al. [20] and Brik et al. [8] differentiates unique network cards of the same model, but is only applicable to network devices. Our USB Fingerprinting technique is applicable to any device using the virtually ubiquitous USB protocol. Network-based OS classification methods such as Richardson et al.'s [52] suffer from more noise than our USB approach, due to its direct connection to the machine. By using the same classifiers investigated by these authors, we are able to distinguish between individual OSes with 99% accuracy. Given an adequate sample size, we are even able to reach 95% confidence of the identity of an individual machine.

Compromise Detection

Network-based (NIDS) and host-based (HIDS) intrusion detection systems [14, 41, 35, 30] can be used for compromise detection. NIDS analyze incoming and outgoing network traffic in order to determine if a system has been infected. HIDS usually refers to software that examines audit logs for suspicious activity, looking for changes in user behavior. Attackers gaining kernel control, however, will be able to control all other software running on a computer, including HIDS.

Garriss et al.'s trustworthy kiosk system [19] uses a smartphone to remotely verify system integrity based on trusted computing techniques. A disadvantage of this approach is the fact that a separate visual identification channel is required. Butler et al.'s Kells system [9] provides similar guarantees, but uses a USB flash drive as a remote verifier. Ensuring physical interaction with the target machine eliminates the need for a visual channel, but the approach is still susceptible to relay attacks, such as Parno's "cuckoo" attack [47].

Distance Bounding

Distance bounding protocols have been used in numerous systems, from computers to radio. Rasmussen et al. [51] demonstrate the need for fast processing speeds on any system implementing such protocols to prevent distance spoofing. Ramaswamy et al. [50] showed that processing delay within networks has become a significant concern, and an individual packet can experience increasing delays. Since our method performs the fingerprinting task over a direct physical connection, we are able to obtain more accurate timing measurements than possible over a network. VIPER [34] demonstrates software attestation with embedded systems, and also shows resilience to similar relay or *proxy* attacks.

CHAPTER III

BACKGROUND

This chapter includes co-authored material from Ryan Leonard and Adam Bates.

USB Protocol

Overview of USB Operation

The USB standard defines a software protocol, firmware protocol, and a set of hardware used in communication between a *host* and a *device* across a *serial bus* [12]. Since USB is a master/slave protocol, the host initiates all interactions. As shown in Figure 1, USB stacks vary from host to host, and are made up of a host controller, a controller driver, a USB driver, and host software. Many machines today support USB, including personal computers, servers, tablets, routers, and many embedded systems. The focus of this work is on the USB 2.0 protocol because, while the USB 3.0 "SuperSpeed" protocol has been codified, it has not been as widely deployed.

For a USB device to be used with a given host, it must go through a setup procedure consisting of three steps. First, is the *bus setup*, during which a set of standard electrical signals is relayed between a host and device between their respective serial interface engines (SIE). This step indicates to the host that a device is connected; the two parties then handshake and negotiate parameters such as communication speed of the device, (high speed, full speed, or low speed). The second step is the *enumeration process*, whereby the host queries the device to determine information about it, such as the device's type (e.g., mass storage, human interface device), its manufacturer and model, and the functionality it supports, among other parameters. Finally, further interactions are passed up to the host's client software through the standard system call interface (e.g., *read(), write(), ioctl()*) and to the device's high-level USB functions (e.g., providing an interface to internal storage, relaying video from a webcam).

Enumeration

In this work, we analyze the *enumeration* period of a USB interaction to make inferences about a host's USB stack. The enumeration period is a good candidate for analysis, as extensive interaction with the host system is not required to force the host to enumerate a device. Thus,

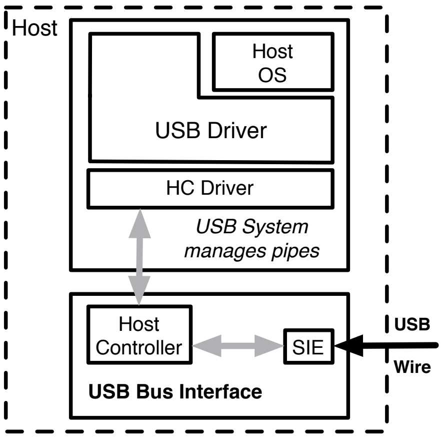

FIGURE 1. Overview of host USB Stack.

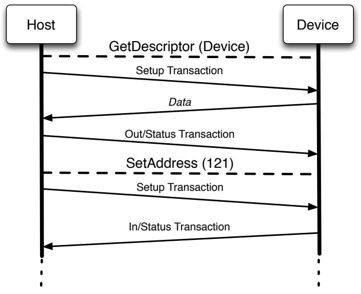

FIGURE 2. Excerpt from USB Enumeration. Control transfers starts are marked by dotted line, and transactions by solid line. Individual packets are not pictured.

it is possible to capture the USB enumeration process on any physically accessible machine, even when lacking login credentials. Additionally, the enumeration process is well defined in the USB 2.0 specification [12], making it easy to interpret. While specific message content and timing will vary depending on the host's USB stack and the connecting peripheral, the presence and purpose of the enumeration period is host and device agnostic.

The enumeration process is a host-driven operation that consists of a three-layered protocol. At the top layer, *control transfers* send abstract information about a device to its connected host. An example of a control transfer is **GetDescriptor(String Manufacturer)**, which informs the host of the device's manufacturer. At the middle layer, *transactions* offer a logical abstraction for bundles of packets. A notable transaction is the **setup transaction**, which describes in detail what the following transactions will be; once the content of the setup transaction is known, the content of subsequent messages is well defined. Note that interrupt requests (IRQ) on a device must be signaled at the end of each transaction, to inform the device's software to queue a future transaction. At the bottom layer there are USB *packets*, which transmit the actual data. Each control transfer is formed by two or more transactions, and each transaction is composed of two or more packets. A sample composition of the USB enumeration process is portrayed in Figure 2.

Classifier Background

Using information extracted from USB enumerations, we employ machine learning classification to predict different characteristics of a target machine. We use the popular machine learning software Weka [24], as it is well respected in the data mining community, and it also provides extensive support for various machne learning algorithms.

In an effort to increase accuracy and ensure the robustness of our models, we analyzed our data with several different supervised learning classifiers. Supervised learning algorithms generate an inferred function to classify previously unseen data instances. They are built with a set of training data instances that contain a vector of attributes, as well as an expected class label. Supervised algorithms then analyze this data, outputting a classification model. Most of the algorithms we use are decision tree-based. We found that decision trees are an excellent fit for USB Fingerprinting data, in which individual features are often sufficient to rule out large subsets of the possible class labels.

J48 Decision Tree The first classifier used in analysis was the J48 decision tree, which is an implementation of Quinlan's C4.5 algorithm [49]. J48 builds a tree classifier by examining the information gain of each attribute, then branching on the attribute yielding the highest amount of information for classification. Information gain is based on the entropy, or the measure of uncertainty, in an attribute. This branching occurs recursively until all instances being examined are of the same class, then a leaf node is created.

Boosting In general, boosting is an ensemble learning method, meaning that it combines multiple weak learner classifiers to create a strong learner and improve accuracy. We employ the AdaBoost algorithm [18], created by Freund and Schapire, which is one of the best known boosting algorithms. AdaBoost trains several weak classifiers, such as decision stumps, then adjusts the weights of each misclassified instance. Each subsequent run of this classifier becomes more and more accurate, as the weights will adjust how instances are classified. Boosting is a common technique to increase accuracy, which is the main reason it was chosen. We used Decision Stumps [2] as the classifier to be boosted, as they are significantly more efficient than J48 trees.

Random Forest The Random Forest classifier [7] proposed by Breiman and Cutler is known for its flexibility and high accuracy [11]. This classifier generates multiple decision trees, providing each with a random subset of the total feature vector, then classifies the instance with each tree. The overall class label is chosen by selecting the mode class label reported by all the trees. Random Forests are extremely efficient in the learning phase.

Support Vector Machines Support Vector Machines (SVMs) attempt to find a large margin hyperplane that separates two classes. This classifier generally achieves high accuracy, as the dividing plane is only affected by a small number of nearest points, or support vectors, thus allowing for less opportunity for misclassification. When more than two classes are present, a separate hyperplane can be learned for each class. SVMs are a very popular choice in many machine learning applications, since their accuracy is usually higher than other linear classifiers; however, our results showed a much lower accuracy using Weka's SMO implementation of SVMs than with the many decision-tree-based algorithms that we used. Rationale for this reduction in accuracy is discussed in later sections. Additionally, when working with our full data set we found the building of SVM models to be prohibitively slow.

CHAPTER IV

METHODOLOGY

With our fingerprinting scheme, we aim to identify several unique features of a machine, and eventualy be able to identify a single machine's unique identity. In this particular work, we focus on identifying a host's operating system. Individual machine identity is also explored.

Operating System Classification

For general operating system classification, we selected the J48 and Boosted Decision Stump algorithms. We began by attempting to build a model that precisely identified the USB enumerations of different operating systems. Our data corpus for these trials, described in Table 2, consisted of the 3 major operating system families. We first built a model for the different classes of OS, disregarding version numbers. Following the default Weka parameters, the model was built on 66% of the dataset, with 34% withheld for evaluation.

For operating system classification, we also attempted to use SVM classifiers on a subset of our features to improve accuracy; however, we saw a distinct drop in accuracy. This failure suggests that combinations of features, which can be easily represented by decision trees, are more helpful in representing the decision boundary between the classes.

Since our data was best suited to a tree-based classifier, we also considered the Random Forest algorithm for this work, due to its high efficiency and high accuracy. RandomForest works similarly to bagging J48 trees; however, since we have very few attributes in this iteration of tha data corpus, Random Forest's multiple trees proved unnecessary. The model obtained by Random Forest yielded the same results as a single J48 tree, and thus we opted for the more efficient J48 algorithm.

Individual Machine Classification

Next, we attempted to build a model that allowed for the identification of a unique individual machine. We used the dataset described in Figure 2, though we altered the class variable to reflect the individual machine, rather than its operating system. Data was partitioned across machines. We once again used the J48 and Boosted Decision Stump algorithms, withholding 34% of the data for later evaluation of the model.

CHAPTER V

DATA COLLECTION

One of the goals of this work was to use small, commodity devices that would be accessible to the general user population. In previous work [32], we used an Ellisys USB analyzer (pictured in Figure 3) for data collection, which was able to collect the USB timing information at a nanosecond granularity; however, it was large, expensive, and required us to manually insert and remove the device 100 times per collection trial. This made using such devices impractical for everyday users. Therfore, this work presents the idea of using a modified USB mass storage device to obtain data traces for analysis. While the granularity of the readings from this device is lower than larger tools, we show that positive results can still be obtained with reasonable accuracy. The most significant contribution of this work was the development of this data collection device.

Device Setup

For our data collection device, we chose to use a Gumstix Overo Fire Computer-on-module (COM) [1] pictured in Figure 4, as it was fully modifiable and offered the same form and functionality as a standard USB mass storage device. It also came equipped with three USB ports that supported host, device, and on-the-go (OTG) protocols. This was important, as our collection process required the device to be connected to both the host machine, as well as a laptop machine running a console into the device. The Gumstix was also inexpensive and allowed the flexibility of installing a custom Linux kernel. The Gumstix is an Open Multimedia Applications Platform (OMAP) 3530-based COM with a ARM Cortex-A8 processor core.

Linux Kernel

Initially, the Gumstix COM was simply a blank device. We needed to be able to modify the kernel to make the device appear as a mass storage device to hosts, and we also needed to modify kernel drivers to log and timestamp the USB protocol messages. We used a micro SD card to install and configure Linux, as it was easier to format and install than the NAND memory embedded in the device itself. It should be noted, however, that the NAND memory can still be written with an image from the SD card, making the device more user-friendly for everyday use.

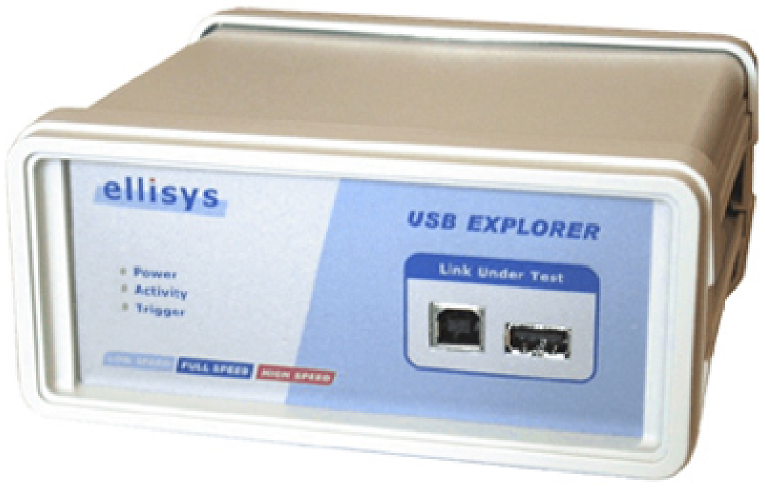

FIGURE 3. Ellysis USB Explorer 200.

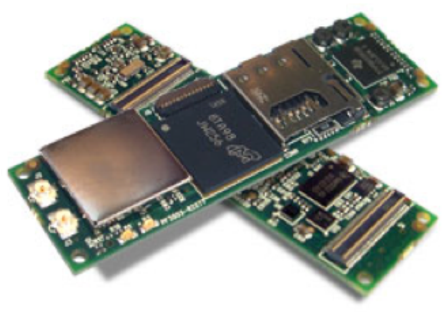

FIGURE 4. Gumstix Overo Fire COM.

First, we configured and installed OpenEmbedded Linux on the Gumstix COM. The Ångstrom distribution was chosen due to its versitility and ability to run on devices with very limited amounts of memory. Since our eventual goal is to create a collection device usable by the average computer user, we wanted an operating system that would could run efficiently on a variety of small devices. We used a micro SD card to install and configure Linux, as it was easier to format and install than the NAND memory embedded in the device itself. It should be noted, however, that the NAND memory can still be written with an image from the SD card, making the device more user-friendly and more cost-effective, should our device be productized.

For the device to capture useful and fine-granularity timestamps, we altered the kernel configuration files to provide realtime support and create a pre-emptable kernel. Realtime kernels ensure the accuracy of time-sensitive processes by calculating the maximum response time to any application event. A pre-emptable kernel allows high-priority processes to interrupt or “pre-empt” the current running process, regardless of the current process’s task. This drastically reduces latency, resulting in more accurate timestamps. To further reduce latency in timestamps and improve accuracy, we further altered the configuraton files to disable all other USB debugging and logging, which may have slowed down the response of the timestamp.

Implementing the timestamps required altering two main driver files: *storage_common.c* and *file_storage.c*. In these files, the driver makes several calls to VDBG, which writes information about the USB transactions to the syslog. In these logs, we also want a nanosecond-granularity timestamp to be placed on each entry, giving us information about how long each event took to be processed. We changed the definition of the VDBG macro to include a call to the kernel-space *do_gettimeofday()* routine in a print statement. This marked each entry with a date in Unix time from which to base our analysis.

To compile and install the new Linux kernel for the Gumstix, the Bitbake build tool was employed to streamline the cross-compilation process. In Bitbake, small files called recipes determine the files and configurations needed for each build. To include the desired functionality into our version of the Linux kernel, we wrote a new recipe indicating the kernel version, patch files, and configurations that we needed.

When compiling the kernel, numerous errors occured which required altering the Bitbake code itself. Most errors were resolved by simply adding *include* directives or removing debugging code. With the bug fixes in place, the bitbake cross-compiler gave no issue, and thus our changes remained in the bitbake codebase.

Setting up the Gumstix as a mass storage device

The Gumstix could be configured to appear to hosts as a variety of USB devices. Our initial tests showed that a USB mass storage device was the best choice, as the traces from its interactions with the host were far more stable than devices with high I/O, such as USB webcams and mice.

Using the Gumstix COM, we are able to emulate manual device insertions and removals with a simple script by using the *modprobe* command to insert and remove the *g_file_storage* module 50 times per trial.

A Mass Storage Gadget (MSG), such as our Gumstix COM, can be configured to appear to hosts as up to 8 SCSI disk drives, or Logical Units (LUNs). The gadget stores information on its LUNs in either a file or a block device called the *backing store.* Since we are only using a single device and only need a single LUN, and because the Gumstix has its own filesystem, we chose to use a backing storage file for the device. This file needed to be created, formatted, and partitioned on the Gumstix as a one-time setup step before *modprobe* could be turn on the file storage module.

The backing storage file was created to represent a simple 64MB device using the following command:

```
dd bs=1M count=64 if=/dev/zero of=/root/data/backing\_file
```

The file was then partitioned and given a filesystem with a series of commands given to the *fdisk* tool. The disk was given 8 sectors, 16 heads, and 1024 cylinders. A new partition was created to span the entire disk, and Windows FAT32 file system was installed. We chose FAT32 because the device would be used on Windows hosts, as well as Macs and Linux, and FAT32 is recognized by all three operating systems.

With the backing store created, the *g_mass_storage* module that controls the mass storage functionality could be interted using the *modprobe* command; the path to the backing store file is passed to the modprobe command:

```
modprobe g\_mass\_storage file=/root/data/backing\_file
```

After this initial setup, a small bash script installed on the Gumstix ran the *modprobe* insertion command, followed by a *modprobe -r* to remove the mass storage module in a loop. A 5-second sleep period was inserted between the insertion and removal to ensure that all data was recorded properly. Each time the device was inserted into a host, this script would be run, which would collect the 50 traces in the device's syslog.

CHAPTER VI

DATA CORPUS

The data we are using is a series of files containing Linux system log messages detailing the communication between an operating system and the Gumstix COM. As described in Section 5.1, the Gumstix timestamped messages from its USB drivers and saved them to its system log. We then offloaded the collected syslogs from the device and concatenated them together. These large files made up our data corpus.

We gathered data from machines of three different operating system types: Windows, Linux, and Mac. Due to issues described in Section VIII, we were able to get significantly more data from Mac machines than from Windows and Linux, which did impact our results. Table 1 shows the distribution of machines from which we collected USB trace data.

OS	**Host Count**
Linux	7
OSX	51
Windows	17
TOTAL:	75

TABLE 1 Number of machines included in data corpus.

Data Preprocessing

As with most data mining tasks, the majority of the time and effort of this project was spent performing pre-processing on the data. Throughout the project, we often needed to iterate back through the knowledge discovery and data mining (KDD) process and return to the pre-processing stage when unsatisfactory classification results were obtained. By altering the way in which the data was formatted and preprocessed, we were able to refine our classification and association results, thus increasing their respective accuracies.

Since the original data was in a system log format, it was inappropriate for data mining in its original form. Not only was it formatted incorrectly for Weka, but it also contained extraneous information that is not needed for classification. Furthermore, since we wanted to focus on the timing of each event, we needed to obtain each event's duration. The original text files only contained the absolute starting time of the event, so the duration attribute needed to be calculated. All of these factors required a number of pre-processing steps to be done first to reformat the data, remove unwanted attributes, and synthesize the desired attributes.

Reformatting

The original data is stored in numerous text files in the form of a Linux system log, with a single entry per line. The data entries are in the following format:

```
Month day hour:minute:second  device_name  gadget_name: seconds microseconds event
```

An example is:

```
Mar  9 15:27:05 overo g_file_storage gadget: 1299684425 200698 get device descriptor
```

The first and most important step was to get the data into a CSV format for use as input to Weka. To accomplish this, we wrote a script in Perl to extract the relevant information from the text file, and output it in a CSV format. Specifically, we extract each event's name, and determined the machine the event came from by extracting its name from the file name. Using the machine type, the script can then easily determine the make (Mac or Dell) and use that in place of the model number. The script also calculates the duration of each event by subtracting the event's absolute starting time from the following event's absolute starting time.

Some of the event names have numeric parameters that differ slightly from instance to instance, such as thread PIDs and buffer sizes. To create a smaller decision tree and a more effective classification, the script was modified to strip the parameters out of the event name, and to place the events into categories. These categories are used as the event name in the final CSV file.

The new CSV file is generated in the following format:

```
Machine Type, Event Name, Duration (in microseconds)
```

In all classification methods, "Machine Type" is used as the class attribute.

Corruption Removal

In a small subset of the original Dell data files, some of the system log messages appear to have been corrupted. Some of the single-line events seemed to have overflowed and are combined with others. In other cases, the event name is completely corrupted beyond recognition and is incomprehensible. To remedy this, we modified our Perl script to identify and remove these corrupted instances by only matching valid events.

Since the percentage of corrupted instances was very low, this removal did not affect the accuracy or integrity of results. Dealing with the corruption is a large problem, however, when attempting to take event sequence into consideration as a classification attribute, because removing, ignoring, or trying to pull apart the events can alter the sequencing and lead to false results. However, for this iteration of the project, we am mainly focusing on considering timing as an attribute.

Resizing Dell Files

We began classifying data from Mac and Dell machines using decision trees, but found that data files for the Dells were significantly longer than for Macs. This extra data per file, in combination with the larger corpus of Dell machines compared to Macs, contributed to overfitting of the decision trees for Dells, drastically affecting the classification accuracy. When trained with cross-validation, the classifier model was very good at classifying the Dell instances and the overall accuracy for the tree was decent, since most of the cross-validated training data contained Dell instances. However, the classification accuracy of Mac instances was below 50% and the tree was not representative of the data we would use as a test set in the future.

Since these results were unsatisfactory, we went back in the KDD process and returned to the preprocessing stage. By cutting out many of the entries in the Dell files, we ensured that both types of files were the same length. From a visual inspection of the Dell files, the removed data did not add significant information to aid in classification, and was thus safe to remove; however, in future iterations we will try to formulate a more effective method to even out the data files, as the removed data could have possibly contained some significant pattern to help train the classifier. In future iterations, we plan to simply gather more Mac data to even out the file sizes. Another method would be to possibly take in file length as another attribute that may help in identifying a machine, since it has been observed that all Dell files are significantly longer than those of the Macs.

Reducing Granularity of the Class Variable

For the initial classification tests, we tried identifying the individual model of each machine, and thus the machine type attribute had 6 possible values (see Table 1 for the model types). The decision trees resulting from this classification displayed poor accuracy. This was due to the fact that the class attribute had too many possible values, and that we were only considering a single attribute for classification purposes.

Since we were unable to increase the number of attributes in the limited time frame we had, we chose to instead reduce the granularity of the class variable and only considered the make of the machine (a Dell or a Mac), instead of the individual model. By doing so, the machine_type attribute had only 2 possible values, giving the classifier better accuracy. The accuracy was also increased due to the fact that the Macs and Dells were reasonably distinct from one another.

CHAPTER VII

ANALYSIS

For this iteration of the work, there are two main aspects of a machine that we aim to identify with our fingerpringing scheme: operating system, and the individual machine identity itself. With the current dataset, we have found success with identifying the first case with reasonably high accuracy. Identifying a completely unique machine, however, is a more difficult problem. Not only does the granularity of data play a large part in identification accuracy, but the problem itself must be treated as an anomaly detection problem. We later discuss the reasons why our scheme had less success in this area and the steps we will take in future efforts.

For the first scenario, we focused on operating system identification, using J48 and Boosted Decision Stumps with a 66/33 percentage split for test and train data. Operating system identification was successful using this classification model, as we identified Linux, Windows, and Apple traces with 79.7% accuracy. The higher numbers along the diagonal in the confusion matrix in Table 2 demonstrate the feasibility of this approach. We note that there is significant misclassification of Linux machines as Macs. We attribute this to the discrepancy between the amount of Mac and Linux data we had available.

Actual\Predicted	a	b	c
a = Mac	44365	711	351
b = Dell	9361	16308	302
c = Linux	4699	959	3027

TABLE 2 Confusion matrix for our machine operating system under strafied 10-fold cross-validation. The diagonal represents correct classifications.

We found that were unable to identify a unique machine with reasonable accuracy, obtaining only around 9% correct classifications. The confusion matrix for this classification is not shown, as it is too large to fit on a page. However, the values on the diagonals of the matrix are very low, indicating that approaching this problem similarly to the operating system identification is not effective with our current data.

We attribute this significant drop in accuracy to a combination of two factors: too few features and a rough granularity of data measurements. Another possible issue is the fact that this problem may be better treated as an anomaly detection problem. We may be able to obtain better results with a semi-supervised learning algorithm, which use both labeled and unlabeled data instances for training. In such an algorithm, we could focus on identifying a “target” machine against any number of “outlier” machines.

These limitations and our proposals for overcoming them are further discussed in Section VIII.

CHAPTER VIII

LIMITATIONS AND FUTURE WORK

Granularity of Events

One limitation of this work is the low granularity of USB messages captured by the Gumstix device. The modifications to the kernel caused USB messages to be logged directly from the driver, where the messages are significantly compressed. Ideally, we would like to capture USB events on a per-transaction level, where transactions are bundles of packets comprising the middle layer of the USB protocol. Interrupt requests (IRQs) are sent at the end of each transaction, and give significant information about a system's USB stack. Using IRQs as features in our classification, we would be able to ascertain the make and model of a machine with significantly higher accuracy.

Number of Attributes

Using only a single attribute is not the most effective way to classify data, as has been shown in the low accuracy of my preliminary tests that attempted to classify individual models of machines. Since the original data from this project was collected last year, was difficult to correctly obtain data on the same machines used in the original trials. Furthermore, during the span of this project, the Gumstix device used for data collection was not set up with our modified Linux kernel and therefore could not be employed for data retrieval.

Data Gathering

When gathering data on Windows machines in the EMU computer lab, we encountered an issue that prevented data from being collected from many Windows OS machines. The Gumstix's continuous module insertion and removal caused conflicting behavior between the Deep Freeze system restore software [16] installed on all the lab's systems. This resulted in a Windows blue screen and automatic restart of the host. All public lab Windows machines were installed with this software, and we were unable to obtain permission to temporarily "thaw" the machine to further investigate and remedy this error. In future work, we will seek out a solution to this conflict, or seek more Windows machines without Deep Freeze installed.

CHAPTER IX

CONCLUSION

This chapter contains co-authored material.

USB Fingerprinting is a technique that can allow for differentiation of machine make, model, and OS with over 79% accuracy. We showed that a commodity embedded USB flash drive is sufficient for collecting data, thus obviating the need for expensive dedicated USB analyzers to perform these operations.

Ultimately, we have discovered that host fingerprinting using variance in USB communications is plausible, and we have supported this claim through experimentation with decision tree classifiers. We are able to identify machines of the same operating system with a high level of accuracy, and we strongly believe that with some slight modifications of approach, be it via pre-processing or a different choice of classifier, we will be able to identify individual machine models with a similar technique.

This work opens an intriguing avenue for future investigation, particularly for developing distance bounding schemes to determine a possible attackers distance. Additionally, our technique has great potential to be expanded as a protection mechanism for the embedded device itself, thus acting as a self-monitoring mass storage device. With more data and feature extraction techniques, we hope to further improve classification accuracy and make our fingerprinting scheme more flexible for other applications.

REFERENCES CITED

[1] Overo Fire. http://www.gumstix.com/store/catalog/product_info.php?products_id=227, March 15 2011 (current release).

[2] Wayne Iba Ai and Pat Langley. Induction of one-level decision trees. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 233–240. Morgan Kaufmann, 1992.

[3] Julia Angvin and Jennifer Valentino-Devries. Race Is On to 'Fingerprint' Phones, PCs. *Wall Street Journal*, November 30 2010.

[4] Vijay A. Balasubramaniyan, Aamir Poonawalla, Mustaque Ahamad, Michael T. Hunter, and Patrick Traynor. PinDr0p: Using Single-ended Audio Features to Determine Call Provenance. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 109–120, New York, NY, USA, 2010. ACM.

[5] Stefan Brands and David Chaum. Distance-bounding protocols (extended abstract). In *EUROCRYPT93, Lecture Notes in Computer Science 765*, pages 344–359. Springer-Verlag, 1993.

[6] Sergey Bratus, Cory Cornelius, David Kotz, and Daniel Peebles. Active Behavioral Fingerprinting of Wireless Devices. In *Proceedings of the 1st ACM Conference on Wireless Network Security*, WiSec '08, pages 56–61, New York, NY, USA, 2008. ACM.

[7] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[8] Vladimir Brik, Suman Banerjee, Marco Gruteser, and Sangho Oh. Wireless Device Identification with Radiometric Signatures. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 116–127. ACM, 2008.

[9] Kevin Butler, Stephen McLaughlin, and Patrick McDaniel. Kells: A Protection Framework for Portable Data. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 231–240, New York, NY, USA, 2010. ACM.

[10] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *CCS '12: Proceedings of the 19th ACM Conference on Computer and Communications Security*, October 2012.

[11] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[12] Compaq, Hewlett-Packard, Intel, Microsoft, NEC, and Phillips. Universal serial bus specification, revision 2.0, April 2000.

[13] Boris Danev, Thomas S Heydt-Benjamin, and Srdjan Capkun. Physical-layer Identification of RFID Devices. In *Proceedings of the USENIX Security Symposium*, pages 199–214, 2009.

[14] Dorothy E. Denning. An Intrusion-detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

[15] Peter Eckersley. How Unique Is Your Web Browser? Technical report, Electronic Frontier Foundation, 2009.

[16] Faronics. Deep Freeze Product Page. http://www.faronics.com/html/deepfreeze.asp.

[17] Jason Franklin, Damon McCoy, Parisa Tabriz, Vicentiu Neagoe, Jamie V Randwyk, and Douglas Sicker. Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting. In *Proc. USENIX Security Symposium*, 2006.

[18] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting, 1995.

[19] Scott Garriss, Rámon Cáceres, Stefan Berger, Reiner Sailer, Leendert van Doorn, and Xiaolan Zhang. Trustworthy and Personalized Computing on Public Kiosks. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys '08)*, pages 199–210, Breckenridge, CO, USA, June 2008.

[20] Ryan M. Gerdes, Thomas E. Daniels, Mani Mina, and Steve F. Russell. Device Identification via Analog Signal Fingerprinting: A Matched Filter Approach. In *In Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2006.

[21] Xun Gong, Negar Kiyavash, and Nikita Borisov. Fingerprinting Websites Using Remote Traffic Analysis. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 684–686. ACM, 2010.

[22] Y. Gu, Y. Fu, A. Prakash, Z. Lin, and H. Yin. OS-Sommelier: Memory-only Operating System Fingerprinting in the Cloud. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 5. ACM, 2012.

[23] Sidhant Gupta, Matthew S Reynolds, and Shwetak N Patel. ElectriSense: Single-point Sensing Using EMI for Electrical Event Detection and Classification in the Home. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, pages 139–148. ACM, 2010.

[24] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[25] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naive-Bayes Classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, CCSW '09, pages 31–42, New York, NY, USA, 2009. ACM.

[26] Andrew Hintz. Fingerprinting Websites Using Traffic Analysis. In Roger Dingledine and Paul Syverson, editors, *Privacy Enhancing Technologies*, volume 2482 of *Lecture Notes in Computer Science*, pages 171–178. Springer Berlin Heidelberg, 2003.

[27] Xin Hu and Z. Morley Mao. Accurate Real-time Identification of IP Prefix Hijacking. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, pages 3–17, Washington, DC, USA, 2007. IEEE Computer Society.

[28] Suman Jana and Sneha Kumar Kasera. On Fast and Accurate Detection of Unauthorized Wireless Access Points Using Clock Skews. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, pages 104–115. ACM, 2008.

[29] Samy Kamkar. evercookie, October 13 2010 (current release).

[30] Gene H. Kim and Eugene H. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, CCS '94, pages 18–29, Fairfax, VA, 1994.

[31] Tadayoshi Kohno, Andre Broido, and K. C. Claffy. Remote Physical Device Fingerprinting. *IEEE Trans. Dependable Secur. Comput.*, 2:93–108, April 2005.

[32] Lara Letaw, Joe Pletcher, and Kevin Butler. Host Identification via USB Fingerprinting. *Systematic Approaches to Digital Forensic Engineering (SADFE), 2011 IEEE Sixth International Workshop on*, pages 1–9, May 2011.

[33] Lingjun Li, Xinxin Zhao, and Guoliang Xue. Unobservable Re-authentication for Smartphones. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium*, 2013.

[34] Yanlin Li, Jonathan M McCune, and Adrian Perrig. VIPER: Verifying the Integrity of PERipherals' Firmware. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 3–16. ACM, 2011.

[35] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA Off-line Intrusion Detection Evaluation. *Computer Networks*, 34(4):579 – 595, 2000.

[36] Desmond Loh, Chia Yuan Cho, Chung Pheng Tan, and Ri Seng 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.

[37] Liming Lu, Ee-Chien Chang, and MunChoon Chan. Website Fingerprinting and Identification Using Ordered Feature Sequences. In Bart Gritzalis, Theoharidou Dimitris, and Marianthi Preneel, editors, *Computer Security, ESORICS 2010*, volume 6345 of *Lecture Notes in Computer Science*, pages 199–214. Springer Berlin Heidelberg, 2010.

[38] Gordon Lyon. Nmap Free Security Scanner. http://nmap.org/, July 16 2010 (current release).

[39] Jonathan M. McCune, Adrian Perrig, and Michael K. Reiter. Seeing-is-believing: Using Camera Phones for Human-verifiable Authentication. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 110–124, 2005.

[40] Greg Minshall. TCPDPRIV. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html, February 05 2004 (current release).

[41] B. Mukherjee, L.T. Heberlein, and K.N. Levitt. Network intrusion detection. *IEEE Network*, 8(3):26–41, 1994.

[42] Nam Tuan Nguyen, Guanbo Zheng, Zhu Han, and Rong Zheng. Device Fingerprinting to Enhance Wireless Security Using Nonparametric Bayesian Method. In *Proceedings of the 30th IEEE International Conference on Computer Communications*, April 2011.

[43] Owens, Rodney and Wang, Weichao. Non-interactive OS Fingerprinting Through Memory De-duplication Technique in Virtual Machines. In *Proceedings of the 30th IEEE International Performance Computing and Communications Conference*, pages 1–8. IEEE, 2011.

[44] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, pages 103–114, New York, NY, USA, 2011. ACM.

[45] Jeffrey Pang, Ben Greenstein, Ramakrishna Gummadi, Srinivasan Seshan, and David Wetherall. 802.11 User Fingerprinting. In *MobiCom '07: Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, pages 99–110. ACM Press, 2007.

[46] Ruoming Pang, Mark Allman, Vern Paxson, and Jason Lee. The Devil and Packet Trace Anonymization. *SIGCOMM Comput. Commun. Rev.*, 36(1):29–38, January 2006.

[47] Bryan Parno. Bootstrapping Trust in a "Trusted" Platform. In *Proceedings of the 3rd USENIX Workshop on Hot Topics in Security (HotSec'08)*, pages 1–6, San Jose, CA, August 2008.

[48] Niels Provos. A Virtual Honeypot Framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, 2004.

[49] J.R. Quinlan. *C4. 5: Programs for Machine Learning*, volume 1. Morgan Kaufmann, 1993.

[50] Ramaswamy Ramaswamy, Ning Weng, and Tilman Wolf. Characterizing Network Processing Delay. In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 3, pages 1629–1634. IEEE, 2004.

[51] Kasper Bonne Rasmussen and Srdjan Capkun. Realization of RF Distance Bounding. In *Proceedings of the 19th USENIX Security Symposium*, pages 389–402, 2010.

[52] David W Richardson, Steven D Gribble, and Tadayoshi Kohno. The Limits of Automatic OS Fingerprint Generation. In *Proceedings of the 3rd ACM workshop on Artificial Intelligence and Security*, pages 24–34. ACM, 2010.

[53] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, USA, August 2004.

[54] Gaurav Shah, Andres Molina, and Matt Blaze. Keyboards and Covert Channels. In *Proceedings of the 2006 USENIX Security Symposium*, pages 59–75, August 2006.

[55] Zhaohui Wang and Angelos Stavrou. Exploiting Smart-phone USB Connectivity for Fun and Profit. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 357–366, New York, NY, USA, 2010. ACM.

[56] Fyodor Yarochkin, Meder Kydyraliev, and Ofir Arkin. Xprobe. http://ofirarkin.wordpress.com/xprobe/, July 29 2005 (current release).