

Host Identification via USB Fingerprinting

Lara Letaw, Joe Pletcher, and Kevin Butler
 Department of Computer and Information Science
 University of Oregon
 Eugene, OR 97403

Email: {zephron, pletcher, butler}@cs.uoregon.edu

Abstract—Determining a computer’s identity is a challenge of critical importance to a forensics investigator. However, relay and impersonation attacks can defeat even computers that contain trusted computing hardware. In this paper, we consider how to leverage the virtually ubiquitous USB interface to uniquely identify computers based on the characteristics of their hardware, firmware, and software USB stacks. We use a USB protocol analyzer to collect data on 24 machines connected to a range of different USB devices, and demonstrate through machine learning classification techniques that we can differentiate not only between operating systems, but between seemingly unnoticeable differences in machine model types as well. We also show that we can differentiate between real and virtualized hosts responding to USB stimuli, and point to new ways of recognizing remote attacks. These results are a first step in showing that USB is a novel and effective means of identifying machines, and a valuable tool in the arsenal of a forensics kit.

Keywords—Forensics, security, USB, identification, fingerprinting

I. INTRODUCTION

As the world becomes increasingly digital, we find ourselves inundated with intelligent electronic devices. Computers and computing devices are becoming evermore pervasive and we are never far from platforms on which to access information locally or through networks. However, determining the identity of these computing platforms is a challenging problem, and becomes particularly vexing when identity is inextricably tied to the question of whether a platform is trustworthy. For example, integrity measurement protocols based on trusted computing principles such as IMA [1] rely on accurately identifying a host’s trusted platform module (TPM) through its identity key.

The issue of identity and its subversion is one that bedevils security and forensics experts alike. If information retrieved from a computer during an investigation turns out not to have been a product of that particular machine but of another system answering in its place, this information, and potentially the entire investigation, could be compromised. Such attacks are not mere flights of theoretical fancy: so-called *relay* or *wormhole* attacks have been demonstrated in wireless networks [2], with

RFIDs [3], and with mobile devices using near-field communication [4]. A variant of the attack, known as the “cuckoo attack” [5], renders even computers hosting trusted hardware such as the TPM vulnerable to other machines answering in their place.

This paper proposes using the USB port found on virtually every computer as a conduit for determining its identity. Unlike past fingerprinting proposals such as wireless device identification [6], we are able to identify hosts via a physical connection with the USB interface, rather than relying on visual authentication channels (e.g., [7].) Furthermore, the unique characteristics of USB hardware and software stacks allow us to distinguish between variations in model identifier, operating systems (and sometimes OS version number), and whether a machine is answering from a real or virtual environment. Not only does this allow greater confidence in understanding whether the machine being queried through USB is the one actually responding to these messages, but it provides a means for detecting machines compromised by malware such as virtual machine based rootkits (VMBRs), e.g., SubVirt [8].

We tested the efficacy of this approach using a hardware protocol analyzer and manually collecting over 1700 instances of USB behavior resulting from plugging a device into a host. We used machine learning techniques, notably decision tree analysis, to classify the collected data. Our results show that we can distinguish the operating system running on the host with 100% accuracy across a corpus of 24 machines running three different OSes. We also found that we could classify five different machine models with over 95% accuracy, after first combining two closely-coupled types. Because of the ubiquity of the USB interface and wide diversity of products that employ it, we believe that USB-based fingerprinting mechanisms are a promising and powerful new addition to the forensic investigator’s arsenal for establishing trust in a machine’s identity.

The rest of the paper is structured as follows: Section 2 provides background on USB and decision tree classifiers; Section 3 describes our methodology using a USB protocol analyzer; Section 4 provides results of USB stack fingerprinting and an application of this

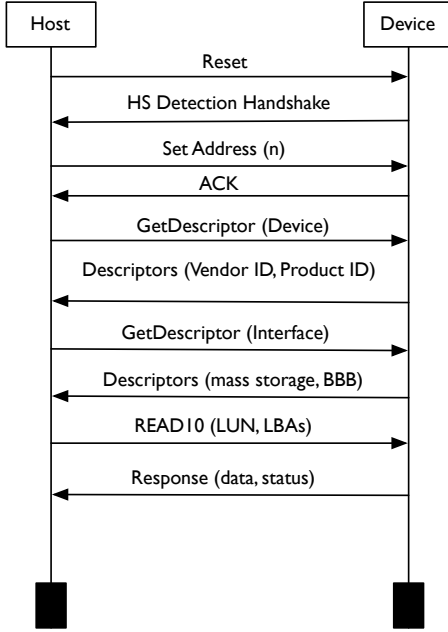


Fig. 1. USB flow diagram for a mass storage device. We are interested in the sequence and timing of enumeration steps like those shown in this diagram.

technique for identifying VMBRs; Section 5 provides discussion and future directions for this research; Section 6 describes related work; Section 7 concludes.

II. BACKGROUND

A. USB basics

At a high level, the USB stack consists of the software, firmware, and hardware on a *host* machine, a *device* connected to the host, and the bus in between. The host portion of the stack normally comprises the operating system, USB device drivers and controllers, the USB port or internal hub, and the internal USB bus. The host itself can be a peripheral (such as a printer or television), in which case there may not be an on-board OS, or it may be an embedded or real-time OS.

USB is a master-slave protocol. That is, when a USB peripheral is attached to a host, it is the task of the host to initiate all communications. In the case where both sides of the USB communication are peripherals (using USB on-the-go [OTG] mode), one side is still required to act as the host. The host communicates with the device through logical pipes which connect to endpoints on the device side. There are two different types of endpoints: single-direction stream pipes and bi-directional message pipes. Stream pipes support bulk, isochronous, and interrupt transfers, and message pipes are only for control transfers. Figure 1 demonstrates a typical message flow of commands between a host and a USB mass storage device.

As we explain in the next section, mass storage is the device class in which we are most interested for this work. Data transfer with USB mass storage is normally accomplished using the bulk transfer type, and a message pipe (which connects to endpoint 0) is used for short communications, such as status requests and responses.

B. Decision tree classifiers

We use decision trees to visually represent the difference between USB fingerprints, and the question-answer process of determining to which type a host machine belongs. Decision trees consist of a root, nodes, and leaves, where the nodes represent a choice or determination that must be made, and the leaves are the final classification decisions. In some cases, more than one path (from root to leaf) may result in the same decision. That is, making a choice at the node of a decision tree does not necessarily imply that some conclusion will not be reached. The path to the decisions, and not any particular node itself, is what is important when using these trees to classify data.

Decision trees were chosen because they are well-known and accepted as classifiers, and because they provide a straightforward graphical representation that is easy to generate and greatly aids in explaining how the fingerprint classifications are made. Though decision trees have proven effective, we intend to explore the use of other classifiers during future stages of our research.

III. METHODOLOGY

Our goal is to devise a technique and a tool for identifying a locally-accessible machine. We want to be able to identify a machine even when it is not connected to a network, in contrast to remote fingerprinting techniques that rely on such availability. Furthermore, identification through USB provides some assurance that we are indeed directly connected to the machine we are fingerprinting.

We hypothesized that the complexity, variability, and ubiquity of the USB stack would make it an ideal candidate for providing substantial and highly-available information from which a distinct machine fingerprint can be extracted. Each hardware and software layer in the USB stack has the potential to affect USB traffic, either through the sequence of USB transactions or the intervals in timing data (e.g., interarrival times for commands). We are specifically interested in fingerprinting the behavior of the host machine in response to various USB devices. That is, the USB stack varies on the host side, but remains constant on the peripheral.

Once in possession of a timing and sequence-based machine fingerprint, we examine methods of using this identifier to distinguish between computers.

A. USB analyzer

In order to test the hypothesis that USB traffic data can provide a way to distinguish between machines, we needed to capture transactions with high precision. To accomplish this, we used an Ellisys USB Explorer 200 protocol analyzer. The USB Explorer hardware was designed especially for developing USB devices and comes equipped with a 60 MHz clock, allowing sub-microsecond timing granularity. The device and host under test are connected through the analyzer. Communications are forwarded to software on an analysis computer, which is also connected to the analyzer. In our tests, this computer is a laptop running Windows Vista Home Basic (32-bit) and is where the USB Explorer software is run to collect and analyze the USB traffic.

B. Host computers

In order to test the viability of our approach, we needed a large corpus of host data. An ideal source was campus computing facilities which, for ease of purchasing and management, often contain many machines of the same type. Also, since it is not usually the case that all equipment in the labs is replaced at the same time, there are many computer types to select from. A complete list of computers used for this study can be found in the full technical report [9].

Primarily for reasons of availability, we concentrated on desktop computers for this work. We looked at nine different models and three major operating systems: Windows, OS X, and Ubuntu. Because we only had access to one machine each for three of the models (iMac 11,1, Dell Latitude E6400, and Dell Latitude D630), we left these types out of our type-classification results. However, they were considered for OS fingerprinting, as they had the same operating system as some of the other machines.

C. Devices

Four USB devices, representing three different USB classes, were used to gather host data. Our objective here was to determine whether a diversity of device classes would correspondingly produce different sets of identifiers. We particularly wanted to determine whether there was an optimal class, or if several devices should be tested on the same machine to gain the maximal set of stable characteristics. A full list of devices used can be found in the full technical report [9].

Early results showed that data collected using a USB mouse tended to be unstable, perhaps due to the sensors used to detect when the mouse is moving. Based on this hypothesis, we opted for using a device without sensors – a USB thumb drive. Further investigation is needed to determine which device class is optimal, or if devices can be used in tandem.

D. Procedure

To help control for the effects of our experimental methods on results, we defined a data collection procedure and followed it closely during all sessions. In brief, our manual procedure, resulting in over 1700 individual trials and data points, was as follows:

- 1) Make sure the computer is switched on and displaying the login screen. For this work, we did not analyze the possible effects of time-since-bootup.
- 2) Disconnect all USB devices to eliminate cross-device interference.
- 3) Connect the host computer to the USB analyzer, using the same port within machine types.
- 4) Record traffic data for 15 seconds.
- 5) Disconnect the device and save the USB trace.
- 6) Perform 15 fingerprinting trials for each machine.

E. Operating system comparison

Given a set of USB event traces across a varied set of desktop computers, we observe that the transaction sequence is consistent within operating systems while differing between them. For the purposes of OS fingerprinting, there are two phases of a USB communication trace in which we are interested: enumeration and bulk IN/OUT transfer. Enumeration happens when the device is first attached to the host. The host asks a series of specific questions, a sequence which is determined by the operating system, in order to decide which driver should be used for future communications with the device. We use this question sequence to distinguish between operating systems. We first take fingerprints from each OS in our study, then we use those for comparison against all other machines.

We are also interested in the sequence of bulk IN and OUT transfer requests, which happens after the device has been configured. Again, we observe that the type and size of these requests is consistent within operating systems. Though we do not require this information to distinguish between the machines in our study, we note that there may be cases in the future where this information could be used to distinguish two machines that appear identical according to the enumeration sequence. For example, an operating system patch might change how data is read from a mass storage device while leaving the descriptor request sequence intact.

F. Machine type comparison

The granularity of identification possible through USB traffic characteristics is not limited to the operating system level; as we show in the next section, we are also able to distinguish between desktop machines that have different model identifiers. The model identifier would normally correspond to the machine's hardware.

Our approach to model fingerprinting is to look at the timing of the USB requests observed during and after enumeration.

More specifically, we are looking at the time intervals between certain USB requests. Each of these intervals would be expected to fall within some range when looking at a single machine type. Therefore, if we find that the interval ranges of a new fingerprint fall far outside the expected ranges, that is an indicator that the new data is from a machine of a different type. The USB requests whose associated timing data we chose to examine are:

- 1) *Suspend*: Duration of the suspend state that occurs prior to device enumeration.
- 2) *Reset*: Duration of the reset that happens after the suspend state.
- 3) *Retry*: Average time between IN transactions.

Throughout this paper, these characteristics will be referred to using their italicized names. Once these three values have been gathered from a machine, they can be used to train or obtain classification data from a classifier.

G. Decision tree classification

We now describe our techniques for mapping a fingerprint to an identification and our methods for telling two fingerprints apart. With OS fingerprints, the task is simple. There are a small number of request types that occur during enumeration, so we encode these types as integers, and create a request sequence. A parallel sequence is constructed from the size of each request. The combination of these two sequences and a label define the OS fingerprint. We compare new, unlabeled, fingerprints with a set of known fingerprints to make an identification of the host under examination.

Comparison of timing-based fingerprints is more complex. To address the more subtle data received, we turn to machine learning techniques, specifically supervised decision tree classification. We used the *Orange 2.0* data mining and machine learning module for Python to construct a decision tree classifier, as shown in Figure 2 and further described in the next section.

IV. EXPERIMENT

The objective of our experiments was to determine if USB traffic characteristics can indeed be used to distinguish between and identify computers, and at what granularity this can happen. We are also interested in whether or not these characteristics can reveal the presence of a virtual machine answering in place of the host.

A. Operating system comparison

The sequence of enumeration events was substantially different between the Windows, OS X, and Ubuntu operating systems we examined, and the intra-OS sequences

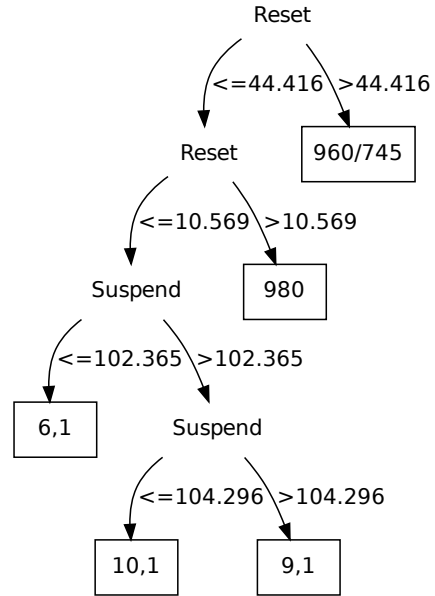


Fig. 2. Decision tree classifier generated from thumb drive fingerprint characteristics. The *Reset* and *Suspend* nodes are associated with two timing characteristics from a USB fingerprint. The box nodes correspond to the machine types. Machines are classified on how their fingerprint characteristics compare with the ranges shown in the decision tree. The *Reset* and *Suspend* values are in milliseconds. The *Retry* timing characteristic was not necessary for distinguishing between the machines.

were consistent. Differences existed in both the order and the type of requests, and the number of times a given request was made.

During the course of our study, some of the computers from which we were taking data were upgraded from OS X 10.6.4 to 10.6.5. After re-fingerprinting these machines, we noticed differences in the transaction sequence post-enumeration. While we did not find specific reasons for this change in Apple’s changelogs, we suspect that modifications to device handling routines were made. The net effect of this change was a demonstration that in certain circumstances, we can accurately identify operating systems at the granularity of a minor revision.

We also looked at two different versions of Windows: Windows 7 Enterprise and Windows Home Basic. By contrast to our experiences with OS X, we found the sequence of USB events for these two operating systems to be identical.

B. Machine type comparison

Using the *Orange* machine learning module, we constructed a decision tree classifier for the machine types examined. *Orange* was given 15 fingerprint trials for each of the 24 machines in our study. Each fingerprint consists of the three timing values we described above,

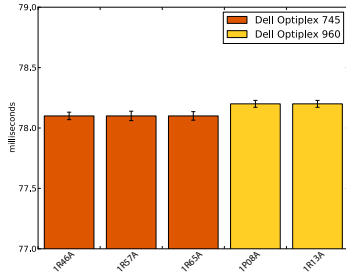


Fig. 3. November 23 average *Reset*, per machine. There is some indication that the closely-coupled 745 and 960 can be distinguished using this characteristic.

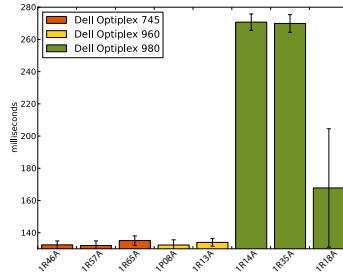


Fig. 4. November 23 average *Suspend*, per machine. The 745 and 960 types are indistinguishable.

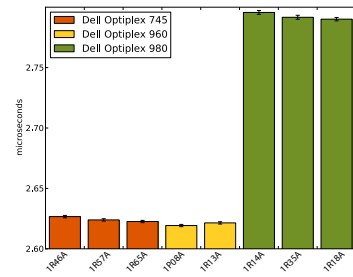


Fig. 5. November 23 average *Retry*, per machine. The 980 type can clearly be distinguished from the 745 and 960 types.

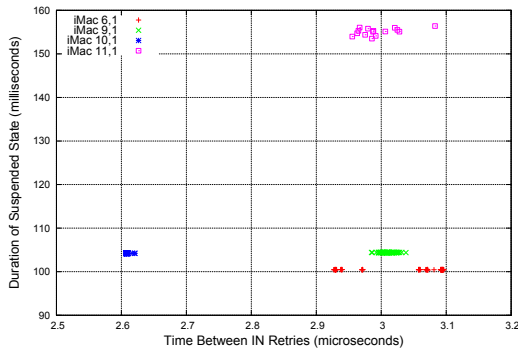


Fig. 6. November 23 iMac fingerprints based on two timing characteristics that shows clean partitioning between models. Overall, the iMac machines seemed to show greater consistency within types than was seen with the Dells.

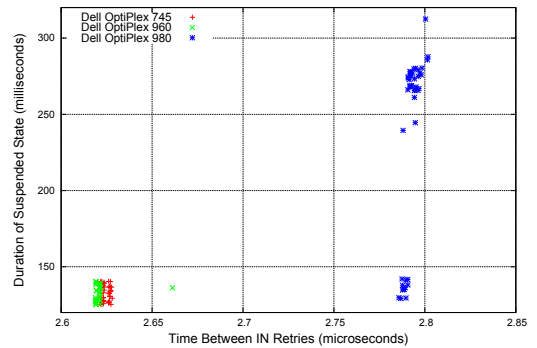


Fig. 7. November 23 Dell fingerprints based on two timing values, where each point represents a trial. These characteristics show distinction between the 980 and the 745/960 models, which are closely clustered. The bimodal clustering of the 980 may indicate that the machines can be classified at an even finer granularity.

and a label specifying which type of machine the data belongs to.

Below are example fingerprints from the five machine types. Some significant figures have been truncated for the purposes of display in this paper:

Suspend	Reset	Retry	Type
130.49 ms	77.93 ms	2.63 μ s	Dell Optiplex 960/745
268.18 ms	11.03 ms	2.79 μ s	Dell Optiplex 980
100.45 ms	10.10 ms	2.97 μ s	iMac 6,1
104.42 ms	10.06 ms	2.65 μ s	iMac 9,1
104.21 ms	10.06 ms	2.61 μ s	iMac 10,1

The Dell Optiplex 960 and 745 types could not be distinguished using the small number of timing characteristics we collected (Figs. 3, 4, 5), so we consider them one type. We are currently examining additional data to determine whether these two types can be separated.

Orange outputs a decision tree that is able to sort through the fingerprint data and classify with 100% accuracy (Fig. 2). We would of course expect this, since

the classifier was given all data. Since the purpose of the classifier is to identify *new machines*, we perform 8-fold cross-validation to evaluate our approach (10-fold validation is standard, but 8 evenly divides our 24 machines). We first randomly partition our machines into 8 sets. Then, we choose each set in turn to act as test data, while the other 7 sets are used to train the classifier. On average, the classification accuracy was 96%. Partitioning of machine model types can be seen in figures 6 and 7.

C. Stability

To check the stability of fingerprinting characteristic values, we re-sampled many of the machines after two weeks. Comparison data for the iMacs in our set is presented in figures 8-13.

For the *Reset* and *Suspend* characteristics, values from the second collection phase were within the original variance range for all but two machines. For the *Retry* characteristic, values were generally inconsistent after

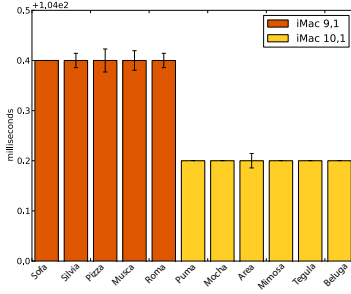


Fig. 8. November 23 average *Suspend*, per machine.

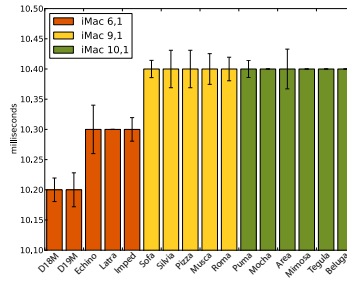


Fig. 9. November 23 average *Reset*, per machine.

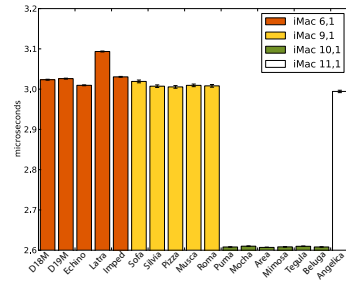


Fig. 10. November 23 average *Retry*, per machine.

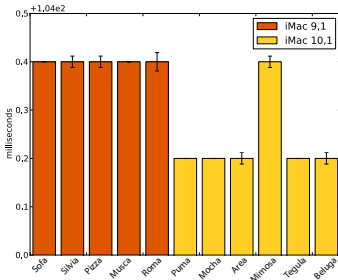


Fig. 11. December 7 average *Suspend*, per machine. Apart from one machine, this characteristic was stable after two weeks.

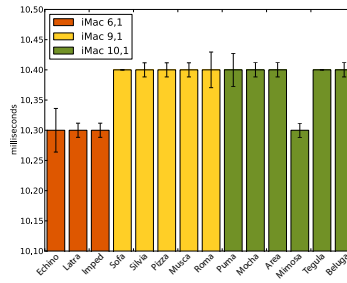


Fig. 12. December 7 average *Reset*, per machine.

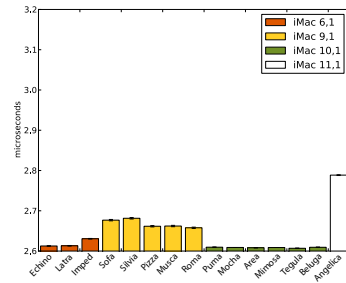


Fig. 13. December 7 average *Retry*, per machine. Values differed from those taken two weeks previous.

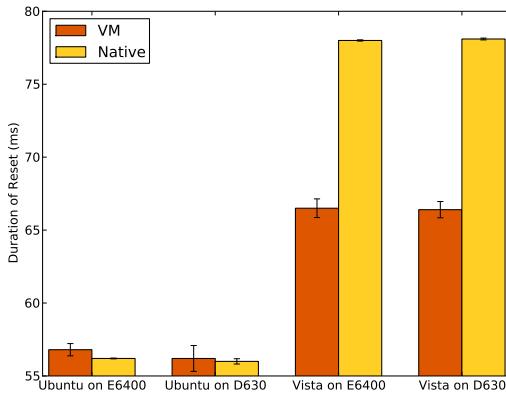


Fig. 14. Timing data with and without a virtual host, on two machines, each dual-booting the same versions of Windows and Ubuntu. In each case, the guest operating system was the same as the host operating system. Results show that, even with just one timing characteristic, there is a statistically significant difference between VM and native for Windows Vista. For Ubuntu, there is a difference between the variability of the timing value.

two weeks. We are currently examining potential reasons for these differences.

D. Virtual machine comparison

We have thus far shown that we can distinguish between operating systems and machine types, assuming the machines are running in an uncompromised state. Now we consider whether we can detect that a machine has been attacked. Virtual machine-based rootkits (VM-BRs) [8] have been suggested and implemented [10] as a way of performing a lower layer attack on a host operating system. Because the virtual machine controls every query a compromised OS could ask, detecting such a rootkit from within a host would be very difficult. Because the timing and enumeration data we collect is not queried through the host OS and instead is more representative of hardware characteristics, it should be possible to detect the presence of a virtual machine.

To model a VMBR scenario, we use VMware to fingerprint images of Ubuntu 10.10 and Windows Vista within their respective identical host OSes. We find that the *Reset* is sufficient for differentiating a VM from the base OS. In the case of the Ubuntu machines, the raw average number of milliseconds did not provide significant distinction, but we find that the value variance was consistently larger for VMs. For the Windows VM experiment, both the average duration and the duration variance were distinct (Fig. 14).

V. DISCUSSION

Our investigations represent an initial attempt to understand fingerprinting techniques as they relate to USB devices. We envision many scenarios in which our approach may be useful. Below, we enumerate some of these potential use cases, which include, but are not limited to: increasing the efficiency of fingerprint collection; investigating relay attacks; the use of more precise clock time measurements; implementing and studying defense mechanisms; attempting to fingerprint a wider range of computers and devices (notably including network devices); and component level fingerprinting.

A. *Quick and efficient data collection*

Despite promising results from the USB analyzer, there are several drawbacks to using it. First, the device is relatively large and bulky, and must be used in conjunction with another computer, and this computer cannot be the same as the one being fingerprinted. Another issue is that the analyzer does not support multi-trial data collection, which makes the process long and tedious, since an actual person is required to manually connect and disconnect the test device several times per minute. Considering we desire a large amount of data per machine, the collection process is overly time-consuming.

We would like to use a different approach in the future. Namely, we are currently investigating whether the behavior of the USB analyzer can be effectively replicated through the use of a small, embedded computer that can be programmed with a USB connect/disconnect script, and which is much easier to carry between test machines. We are using a Gumstix Overo Fire board [11], running the OpenEmbedded Linux distribution [12] and configured to act as a USB mass storage device. The Overo Fire board is a 720MHz ARM Cortex-A8 based computer, containing 256MB of flash and RAM storage, along with a MicroSD card reader. This board was chosen for its small size (roughly the size of a stick of gum), lower power requirements, USB capabilities (containing both OTF and host mode), and availability.

Preliminary results from 24 machines across three different operating systems and four models indicate that the granularity of timing data collected using the Gumstix device is sufficient to distinguish between operating systems and machine types. As these experiments are still in their early stages, we will not go into detail about the classification accuracy achieved using these techniques.

B. *Relay and other attacks*

An important part of our work is using USB traffic to detect that a computer has been compromised. We have

thus far examined a VMBR-like scenario, but, in addition to running a hypervisor directly off hardware, we would like to extend our analyses to other attacks with the potential to affect the USB stack. For example, if an adversary has configured a target computer to forward all user commands to a remote machine, we would expect this to affect the timing of USB transactions. Systems that are especially vulnerable to relay attacks are those that already use NFS, where connection to a network is already necessary so the relay could not be detected by simply disconnecting from the Internet. In this case, the USB traffic might have a different delay between the two network locations if the network links were sufficiently different. Additionally, as work on distance-bounding protocols has shown [13], [14], we can apply techniques to limit the range in which an adversary must be present in order to even attempt to mount an attack based on timing distinguishability.

Another attack type of interest is boot redirection. This happens when an adversary, usually remote, gains access to a computer and causes it to reboot from a location other than what is expected by the user. The alternate boot location could be local or remote. This attack is especially worrisome in light of such technologies as Intel AMT [15], a hardware-based remote management system mostly used by businesses and organizations with out-of-band security and maintenance needs. In addition to boot redirection, Intel AMT includes a dedicated VNC server, wake-on-LAN, and remote shut-down features. This de-localization of control opens up a wide variety of vulnerabilities, some of which may be addressable by USB fingerprinting.

C. *Defenses*

While we have investigated the USB stack characteristics externally, we have not spent much time looking at ways in which we could harden a host to this sort of fingerprinting. The immediate solution appears obvious at first blush: change the enumeration patterns of a host and add random timing delays in areas we benchmark. The implications on peripheral performance are unstudied, however, and more work is needed to determine what sort of impact these perturbations would have on the host, as well as whether or not they are practical on the host, as well as whether or not they are practical. Furthermore, such an analysis can facilitate pinpoint hardware that cannot be spoofed (effectively finding timing minimums).

D. *Wider range of machines*

A future course of investigation involves examining network device state, specifically to see if USB-enabled devices such as routers and other network appliances

have been compromised. These devices often represent attack targets on a network, and cannot easily be audited due to restrictive interfaces. USB-based testing may allow us to more completely and accurately detect dramatic state changes inside these devices indicative of device compromise.

E. Component-wise analysis

Lastly, we would like to try to fingerprint specific chipsets inside a variety of hosts. It is common practice to reuse components throughout a variety of boards, which may share common defining characteristics. If we can identify these hardware characteristics, we can identify more specific attack vectors into a machine by targeting specific device drivers.

VI. RELATED WORK

A. Fingerprinting

Many techniques have been explored for identifying machines via characteristics of how they communicate. Remote fingerprinting tools like *Nmap* [16] and *Xprobe* [17] detect operating systems by examining network traffic. *Nmap* works by probing a target machine with strategically-crafted TCP, UDP, and ICMP packets, and then matching the response against a database of pre-collected fingerprints. Much of what *Nmap* uses to create a fingerprint are differences in TCP options and protocol implementation. Such techniques, however, can be rendered ineffective by *Honeyd* [18], which spoofs operating systems at the network layer.

Other methods aim to identify individual machines, such as work by Kohno et al. [6], which demonstrates the viability of remote fingerprinting using clock skew data. The approach is to look at TCP and ICMP timestamps and use this data to identify miniscule imperfections in how the remote machine keeps time. However, it has been shown that the TCP and ICMP timestamps can be disabled or manipulated [19], [20], [21].

Semi-persistent network/wireless configuration data has also been used to fingerprint at the device level [22], [23], as have browser settings [24]. Persistent cookies – like Flash cookies and *evercookies* [25] – make browser fingerprinting even more reliable. Fingerprinting services are even available commercially, and are of particular interest to advertising agencies [26].

Little previous exploration has been made into identifying a computer based on USB traffic. An exception is recent work by Wang et al. [27], in which a USB-equipped smart phone detects the host operating system and uses this information to choose the type of attack to perform. We note that Wang et al. do not describe USB fingerprinting at the granularity that we achieve.

B. Compromise detection

Network-based (NIDS) and host-based (HIDS) intrusion detection systems [28], [29], [30], [31] can be used for compromise detection. For example, analysis of incoming and outgoing network traffic by a NIDS can indicate that a system has been impregnated with malicious code that relays user requests or steals and transmits user data to the attacker’s remote computer. NIDS are, however, limited in scope to network traffic.

HIDS usually refers to software that examines audit logs for suspicious activity, looks for changes in user behavior (such as frequency of login failures), and/or behavior that matches the profile of specific attack scripts. The disadvantage of this type of approach is that any attacker who has gained control of the kernel will also have control of any software running on the computer.

Garriss et al.’s trustworthy kiosk system [32] uses trusted computing techniques to ensure the system integrity, using a smartphone as a remote verifier. Such an approach has the disadvantage of requiring a separate visual identification channel. Butler et al.’s Kells system [33] provides similar guarantees, but uses a USB flash drive as the remote verifier. This ensures physical interaction with the target machine, obviating the need for a visual channel, but the approach is still susceptible to relay attacks, such as Parno’s “cuckoo” attack [5].

VII. CONCLUSIONS

We have presented a methodology for host fingerprinting through USB stack characteristics, and confirmed the promise of this approach through implementation and experimentation. We are able to identify machines of the same operating system and make highly accurate guesses about the model identifier of a computer, assuming we have previously fingerprinted that model type. Our initial results show that USB stack fingerprinting is a novel approach to determining host identity and for detecting machine compromise. We believe that the continuation of this work will result in the development of even finer-grained identification of hosts via USB.

ACKNOWLEDGEMENTS

We would like to thank Ellisys for very generously allowing us usage of their USB Explorer 200 protocol analyzer for our testing, Daniel Lowd for advice on decision tree classifiers, and Andrew Bruce, Kurt Mueller, and Masoud Valafar for help with editing and proofreading.

REFERENCES

- [1] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, “Design and Implementation of a TCG-based Integrity Measurement Architecture,” in *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, USA, Aug. 2004.

- [2] Y. Hu, A. Perrig, and D. Johnson, "Wormhole attacks in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 370–380, 2006.
- [3] G. P. Hancke and M. G. Kuhn, "An RFID Distance Bounding Protocol," in *Proc. First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SecureComm)*, Washington, DC, USA, 2005, pp. 67–73.
- [4] L. Francis, G. Hancke, K. Mayes, and K. Markantonakis, "Practical NFC peer-to-peer relay attack using mobile phones," *Radio Frequency Identification: Security and Privacy Issues*, pp. 35–49, 2010.
- [5] B. Parno, "Bootstrapping trust in a 'trusted' platform," in *Proceedings of the 3rd USENIX Workshop on Hot topics in security (HotSec'08)*, San Jose, CA, Aug. 2008, pp. 1–6.
- [6] T. Kohno, A. Brodido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Dependable Secur. Comput.*, vol. 2, pp. 93–108, April 2005.
- [7] J. M. McCune, A. Perrig, and M. K. Reiter, "Seeing-is-believing: Using camera phones for human-verifiable authentication," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2005, pp. 110–124.
- [8] S. T. King, P. M. Chen, Y.-M. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch, "Subvirt: Implementing malware with virtual machines," *Proceedings of the IEEE Security and Privacy*, vol. 0, pp. 314–327, 2006.
- [9] L. Letaw, J. Pletcher, and K. Butler, "Host Identification via USB Fingerprinting," Department of Computer and Information Science, University of Oregon, Eugene, OR, USA, Tech. Rep. CIS-TR-2011-02, Mar. 2011.
- [10] J. Rutkowska, "Introducing Blue Pill," <http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html>, June 2006.
- [11] "Overo Fire," http://www.gumstix.com/store/catalog/product_info.php?products_id=227, March 15 2011 (current release).
- [12] "OpenEmbedded," http://www.openembedded.org/index.php/Main_Page, March 15 2011 (current release).
- [13] J. Clulow, G. Hancke, M. Kuhn, and T. Moore, "So near and yet so far: Distance-bounding attacks in wireless networks," *Security and Privacy in Ad-Hoc and Sensor Networks*, pp. 83–97, 2006.
- [14] S. Drimer and S. Murdoch, "Keep your enemies close: Distance bounding against smartcard relay attacks," in *Proceedings of the 16th USENIX Security Symposium*, Aug. 2007, p. 7.
- [15] Intel, "Intel VPro Technology," <http://www.intel.com/technology/vpro/index.htm>, March 2011 (current release).
- [16] G. Lyon, "Nmap Free Security Scanner," <http://nmap.org/>, July 16 2010 (current release).
- [17] F. Yarochkin, M. Kydyraliev, and O. Arkin, "Xprobe," <http://ofirarkin.wordpress.com/xprobe/>, July 29 2005 (current release).
- [18] N. Provos, "A Virtual Honeypot Framework," in *Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 1–14.
- [19] R. Pang, M. Allman, V. Paxson, and J. Lee, "The Devil and Packet Trace Anonymization," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 29–38, January 2006.
- [20] G. Minshall, "TCPDPRIV," <http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html>, February 05 2004 (current release).
- [21] G. Shah, A. Molina, and M. Blaze, "Keyboards and Covert Channels," in *Proceedings of the 2006 USENIX Security Symposium*, Aug. 2006, pp. 59–75.
- [22] X. Hu and Z. M. Mao, "Accurate Real-time Identification of IP Prefix Hijacking," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, ser. SP '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 3–17.
- [23] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall, "802.11 User Fingerprinting," in *MobiCom '07: Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*. ACM Press, 2007, pp. 99–110.
- [24] P. Eckersley, "How Unique Is Your Web Browser?" Electronic Frontier Foundation, Tech. Rep., 2009.
- [25] S. Kamkar, "evercookie," <http://samy.pl/evercookie/>, October 13 2010 (current release).
- [26] J. Angvin and J. Valentino-Devries, "Race Is On to 'Fingerprint' Phones, PCs," *Wall Street Journal*, November 30 2010.
- [27] Z. Wang and A. Stavrou, "Exploiting smart-phone usb connectivity for fun and profit," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 357–366.
- [28] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on Software Engineering*, vol. 13, no. 2, pp. 222–232, 1987.
- [29] B. Mukherjee, L. Heberlein, and K. Levitt, "Network intrusion detection," *IEEE Network*, vol. 8, no. 3, pp. 26–41, 1994.
- [30] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "The 1999 DARPA off-line intrusion detection evaluation," *Computer Networks*, vol. 34, no. 4, pp. 579 – 595, 2000.
- [31] G. H. Kim and E. 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*, ser. CCS '94, Fairfax, VA, 1994, pp. 18–29.
- [32] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang, "Trustworthy and personalized computing on public kiosks," in *Proceedings of the 6th international conference on Mobile systems, applications, and services (MobiSys '08)*, Breckenridge, CO, USA, Jun. 2008, pp. 199–210.
- [33] K. Butler, S. McLaughlin, and P. McDaniel, "Kells: a protection framework for portable data," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 231–240.