A Security Assessment of Trusted Platform Modules

Computer Science Technical Report TR2007-597

Evan R. Sparks

Evan.R.Sparks.07@Alum.Dartmouth.ORG

Senior Honors Thesis

http://www.cs.dartmouth.edu/~pkilab/sparks/

Department of Computer Science

Dartmouth College

Advisor: Dr. Sean W. Smith

June 28, 2007

Contents

Abstract

Trusted Platform Modules (TPMs) are becoming ubiquitous devices included in newly released personal computers. Broadly speaking, the aim of this technology is to provide a facility for authenticating the platform on which they are running: they are able to measure attest to the authenticity of a hardware and software configuration. Designed to be cheap, commodity devices which motherboard and processor vendors can include in their products with minimal marginal cost, these devices have a good theoretical design. Unfortunately, there exist several practical constraints on the effectiveness of TPMs and the architectures which employ them which leave them open to attack. We demonstrate some hardware and software attacks against these devices and architectures. These attacks include Time of Check/Time of Use attacks on the Integrity Measurment Architecture, and a bus attack against the Low Pin Count bus. Further we explore the possibility of side-channel attacks against TPMs.

1 Introduction

The term "Trusted Platform Module" refers to a specification (and implementation of that specification) of a microcontroller which has been designed by the TPM Work Group, a part of the *Trusted Computing Group* (TCG). The primary purpose of a TPM is to provide low-volume public key cryptographic services to a machine using a private key stored on the TPM itself and to store credentials only to be used when the system is in some specific configuration. Using the TPM in conjunction with a trusted system BIOS, computers equipped with a TPM can boot into a trusted operating system, and can verify the configuration of software which is running on them to third parties.

One requirement of Trusted Platform Modules during development of the

TPM spec was that they be producible at a low cost. This restriction led to a low requirements for hardware security. While the chip itself is required to be FIPS 140-2 compliant, this protection does not cover all possible attack vectors against it (particularly for the lower "levels" of the FIPS 140-2 standard).

The process by which third parties can verify software configurations remotely is known as "Remote Attestation". This process allows the TPM to measure and sign the authenticity of a program at load time. This signature can be part of a challenge-response protocol with a third party, so that they can be sure that the program that is running is the one which they expect. Unfortunately, on modern computers, measuring a program at load time is not sufficient to verify its authenticity. If an attacker can modify a program after it has been measured, then he can run it in whatever configuration he prefers, unbeknownst to the third party.

In this paper, we will explore a number of new attacks against TPM's both in terms of hardware and software. On the software side, we will exploit the ability of modern operating systems to modify arbitrary regions of memory at any time in order to change a running program after it has loaded. We modify the behavior of the program such that it does the *opposite* of what a third party would want it to do, without the detection of the TPM or the third party.

On the hardware front, we explore two kinds of attack. First, we look at "faking" the trusted boot process in such a way that a TPM is taken into a state where it should not be, given the machine's hardware and software configuration. This completely circumvents the remote trust concept, since the TPM no longer has any reliable information on the platform. Second, we explore the feasibility of side-channel attacks on the TPM's cryptographic system.

Related Work Kursawe, et. al [8] look at a number of passive attacks against TPMs by monitoring signals across the bus that the TPM resides on in their test

machine. They also hint at the possibility of more active attacks such as the ones we will demonstrate in this paper. Part way through our independent work on the reset attack, Bernhard Kauer [2] demonstrated TPM Reset Attacks and proposed a countermeasure; his work has been accepted into USENIX Security 2007. Sadeghi, et. al [12] also discuss testing TPMs for specification compliance. Before TPMs came into existence, researchers at the IBM corporation (including my adviser) created the IBM-4758, a FIPS 140-1 level 4 system designed to be a secure cryptographic coprocessor and key store, which provided many of the same features as a TPM. [6] This device offers a number of physical security features that TPMs lack. These features (among others) make the device rather expensive, and as such it is not a suitable substitute for a TPM in most use cases. Paul Kocher [10] invented (in the public world) and demonstrated the first side-channel attacks on RSA, which relied on timing. He also invented differential power analysis attacks against RSA engines [9]. David Brumley and Dan Boneh [3] have demonstrated a timing attack against an implementation of RSA which employs the *Chinese Remainder Theorem* (CRT) and Montgomery Reductions.

2 Motivation

The mission statement of the TCG reads "Through the collaboration of platform, software, and technology vendors develop a specification that delivers an enhanced HW and OS based trusted computing platform that enhances customers' trusted domains." [15] There are a variety of usage situations for this architecture, including risk management, asset management, e-commerce, digital rights management, and security monitoring and response, among others. It is vital that if we are to trust the architecture to enhance the security of these situations, that we understand how the architecture works, and where it might fail.

The aim of this research is to show that people need to be careful when deciding to adopt the TCG architecture in their particular usage situation, and to reiterate that there is no such thing as a security solution that works perfectly. We seek to highlight a number of problems with the TCG architecture and in specific TPM implementations. It is important that adopters of the TCG architecture understand and weigh these risks when making the decision to adopt the architecture.

3 Background

In this section we provide background information to the reader about the inner workings of certain aspects of the TCG architecture and TPMs. While not exhaustive, this section will help the reader be familiar with a number of terms and concepts used throughout the paper.

3.1 The TCG Architecture

The TCG architecture is a proposed standard created by the TCG to enhance enable the creation of a trusted computing platform. Its main features enable Secure I/O, memory curtaining, sealed storage, and remote attestation. [11] The reliability of these features all rest on a single piece of hardware, known as a Trusted Platform Module. The TPM and the platform's BIOS make up the core root of trust for the platform. That is, if one of these systems is compromised, then the entire system fails.

3.2 Trusted Platform Modules

Trusted Platform Modules, as mentioned before, are the basis for the TCG architecture. They provide a number of key features, including non-bulk RSA encryp-

tion and decryption, secure storage of private and master keys (and other credentials), integrity measurement (through the use of *Platform Configuration Registers* (PCRs)), and pseudo random number generation. We focus on the integrity measurement features of TPMs and their ability to securely store private keys.

The Endorsement Key is the critical piece of information stored on the TPM which keeps this system secure. An endorsement key is a private key which the TPM uses to generate all other keys that it sends to other parts of the platform. All keys created are bound to a specific endorsement key. They reside in key blobs which can only be decrypted with the endorsement (or the key which created them). It is a requirement of the TPM specification that this Endorsement Key be protected and never be exposed to external reading. [14] This key is used in the generation of the *Attestation Identity Key* (AIK) which is used in remote attestation. The AIK cannot be used by the TPM unless the platform owner provides the TPM with a 160-bit AIK authentication value. Further, keys may not be loaded by the TPM unless the key which created them (their *parent key*) has also been loaded, and users can provide the proper authentication credentials to use them.

Platform Configuration Registers are a vital component of TPMs. They are a set of volatile registers which support only the extend operation. When the TPM is initialized, the PCRs are originally set to a null value. When the system is measured, the TPM takes a SHA-1 hash of the data that is reported to it, and extends a particular PCR with that value. It does this by concatenating the old value of the PCR with the newly reported value, and taking a SHA-1 of the result. By this process, only a particular sequence of measurements with the same results will leave a particular PCR in the same state. PCRs are useful because they are non-volatile, non-writable by anyone but the TPM, and because of properties of

SHA-1, difficult to coerce into a desired state, except by having everything on the platform be the same. A third party can keep a database of 'acceptable states', which will simply be a copy of the PCR values and the public key of a particular platform. Only if that platform can sign those values (and a random challenge), will the third party believe that the platform really is who it says it is.

3.3 The Low Pin Count Bus

Most TPMs deployed today are microcontrollers that sit on the *Low Pin Count* (LPC) bus. Designed as a replacement for ISA, the LPC bus is a common fixture on modern motherboards. [7] As a result, on most modern Intel-based computers, the most common devices found on the LPC bus are the BIOS and legacy input and output devices (via SuperI/O). The reasons behind putting the TPM on the LPC bus are threefold. First, the TPM needs to be logically close to the BIOS. By locating the TPM on the same bus as the BIOS, it is easier to measure the BIOS directly, rather than having to go through the CPU or other equipment. Second, this bus is easy to write software for, since from a software level it looks just like an ISA bus. Third, at 33MHz this bus is sufficiently fast for relaying results of low volume cryptographic operations back to the system in a reasonable amount of time.

4 Experimental setup

In this section, we will discuss the experimental setup of some new attacks on TPMs and the architectures which employ them. Our reset attack is a replication of the attack which was performed by Bernhard Kauer; we were scooped. [2]

4.1 Software attack

We run our software attack on an IBM NetVista PC, equipped with a Atmel TPM v1.1b. This attack has also been shown to run on more recent PCs equipped with STMicro v1.2 TPMs. The PC is running Linux v2.6.15.6 with the TPM device driver statically compiled into the kernel. It uses Trusted GRUB (a secure boot-loader) to undergo a trusted boot process at boot time. We make the assumption that a user with root access has the ability to insert a *loadable kernel module* (LKM) with insmod. Even if this assumption cannot be met, any vulnerability in the Linux kernel which allows a (root or non-root) user to run arbitrary code in kernel mode will allow the attack to take place. These types of vulnerabilities have occurred in the Linux kernel frequently. [1] Critics of this attack will make the claim that a truly TCG compliant system will only allow *trusted* kernel modules to be inserted, and will only run a trusted kernel. However, "trusted" does not mean "trustworthy." Commodity operating systems have a long history of not being trustworthy, even if users choose to trust them. In reality loading only "trusted" modules would severely limit the operating power and usability of the system. Also, given a vulnerability like this, the system will not detect the rogue code running. Further, given the size and complexity of modern operating systems, deciding whether or not to fully trust a particular version of an operating system is difficult.

4.2 Reset attack

We use a similar IBM NetVista PC as used in the software attack for our reset attack. The motherboard in this machine has support for a TPM "module" daughter-board. There is a 28-pin header on the board which attaches a small TPM board to the motherboard. This makes considerable design sense for IBM,

since the TPM specification is constantly changing and the ability to easily swap an old version TPM out of the motherboard in exchange for a new one is a reality of the industry. Fortunately for us, this design also allows us to both monitor the LPC bus and interpose the TPM with relative ease. To drive the reset, we use 3-inches of insulated 30-gage wire, as well as some ribbon cable to make the TPM easier to access physically.

We use an Agilent 16803A Logic Analyzer to monitor signals across the LPC bus on both of these boards. This device will eventually allow us to "record" a trusted boot process.

4.3 Timing attack

In our timing attack against TPMs, we use the Boneh and Bromley [3] variant of RSA Timing Analysis attack against the RSA engine in an STMicro v1.2 TPM. The data sheet of this chip [13] tells us that the RSA engine uses CRT to do RSA quickly. We run this attack on an Intel Core2 Duo equipped Dell PC running OpenSolaris, due to the high-resolution timing features that Solaris supports. This PC contains a STMicro v1.2 TPM.

5 Discussion of attacks

In this section, we discuss and analyze some specific attacks on TPMs and the TCG architecture. Namely, we examine the exploitation of a *Time of Check/Time of Use* (TOCTOU) vulnerability in the Remote Attestation model of the TCG architecture. We then explore malicious bus attacks on an Atmel v1.1b TPM sitting on an LPC bus. Finally, we look at the susceptibility of these TPMs to timing analysis attacks.

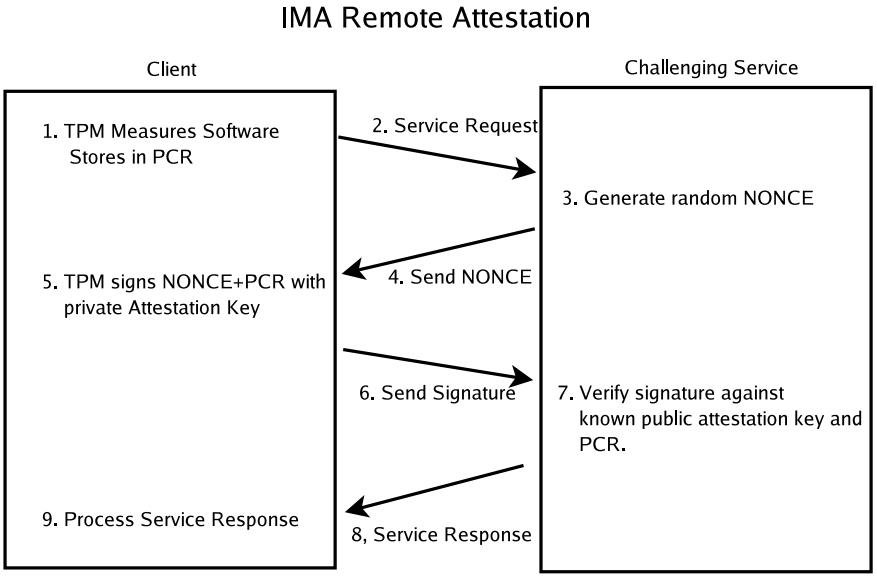

Figure 1: IMA Remote Attestation

5.1 Software attack

One key facility that TPMs provide is the ability for the TPM to attest to the configuration of a piece of software to a third party. The TPM measures that configuration of the software at load time, signs this configuration (using a key derived from its secret key). Then the user passes this configuration on to a third party which presumably uses this configuration information to decide whether or not to allow the measured platform to run a piece of software or join a network session.

The issue here is that there is no mechanism for an attestation requester to make sure that things haven't changed since the TPM last measured. Current attestation setups call for a combination of secure boot, followed by post-boot measurements at the application level. A current and popular attestation scheme for Linux is called *Integrity Measurement Architecture* (IMA), developed by IBM. Figure 1 illustrates this process. A trusted platform must be able to verify that it

has loaded proper software by having appropriate values in its *Platform Configuration Registers* (PCRs). A challenging service provider will only supply services to the platform if it can prove that it has the proper software loaded.

IMA employs use of an "mmap file hook", as well as a "load module hook" to measure files before they are mapped into memory and to measure modules before they are integrated into the kernel, respectively. [4]

Notice that a key component of this system is that if measures a binary at *load* time. If a binary changes after it has been loaded, the configuration of the system will not match the attested configuration. In the traditional model, this is not a problem because the memory that makes up a binary's `.text` segment is mapped as read-only in the operating system's page table. With IMA, the `.text` segment will be measured before the program is loaded into RAM. This means that if a program can change this segment of RAM *after* it has been loaded, that its behavior can be modified, even though the measurement in the TPM shows that the program is in its original state.

The page table is the data structure which the kernel uses to map virtual addresses to pages in physical RAM. In Linux, this structure also contains information on the permissions that a process has to read or modify the particular segment of RAM. Each process has its own page table. As such, these need to be efficient (both in terms of time and space) data structures. The current model in use in Linux systems is illustrated in Figure 2. In this 4-level model, a virtual address is first looked up in the *Page Global Directory*, which gives an entry in the *Page Middle Directory*, which then points to a specific *Page Table Entry*, which contains a pointer to a physical location in RAM.

When a program needs access to a virtual address, the kernel needs to look it up in the page table to see which physical page needs to be read or written to. If this address is paged out, it will need to swap it back in, in order to give the

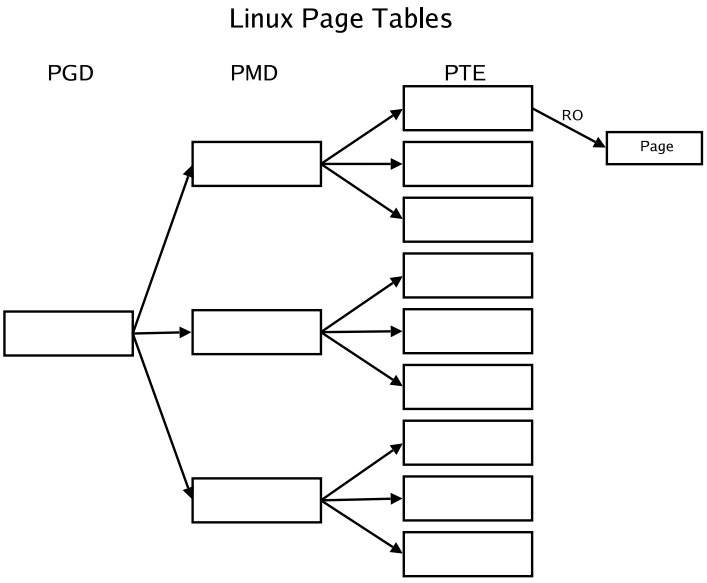

Figure 2: Linux 4-level Page Table

program the ability to read or write to that data. It does this by traversing the tree shown above, and seeing the properties that are stored in the rest of the data structure. If, for instance, a program tries to write to an address on a page that is not listed as writeable, the kernel will raise a page fault.

Our attack consists of a kernel module which takes 3 parameters: a target process ID, a target virtual address, and some data to write at that address. In order to show that simply monitoring a process's page tables is not a sufficient defense against this attack, our module will not modify the target process's data structures at all, but will modify those of insmod, the program which puts it into the kernel. Equivalently, the module could modify any other process's page table, and the attack would still work. The attack will simply copy the page table entry that maps to the corresponding target virtual address into the kernel's page table at an arbitrary address. The page table entry contains both the permissions and status associated with that page, as well as that page's virtual address. After this entry is copied into the kernel's page table, we modify the entry and mark it as

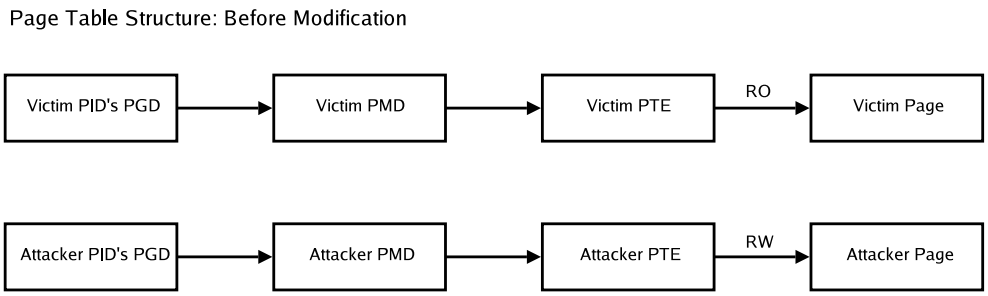

Figure 3: Page Table Before Attack

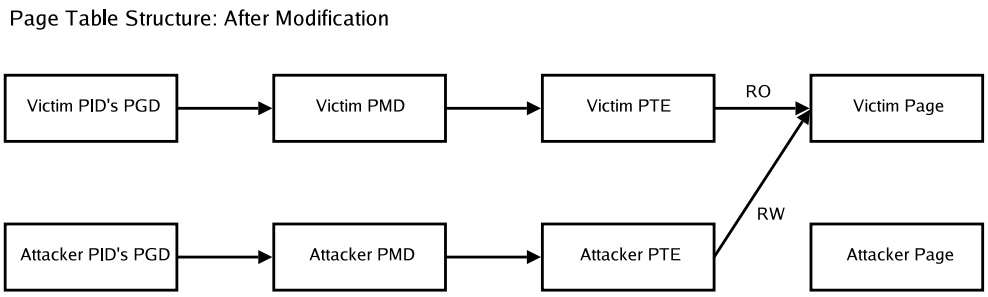

Figure 4: Page Table After Attack

writable. Then, we will write data supplied by the user of the kernel module to the corresponding page in physical memory. By doing this, we are modifying the `.text` segment of the program, and will change the way the program executes when it gets to an instruction in the modified address range. Figures 3 and 4 illustrate this attack.

We have constructed a small example login program to demonstrate this attack. The program takes in a password, and if the password matches the hard-coded password in the program, we execute a shell; otherwise, we quit. After disassembling this program with `gdb`, it is determined that it suffices to change a `je` opcode to a `jne` opcode in the `.text` segment, to reverse the decision whether to execute this shell. Instead of running a shell upon receipt of an appropriate username and password pair, the program will exit. Further, upon receipt of an invalid password, the system will execute a shell.

On x86 architectures, the change between a `je` and a `jne` opcode is a difference of one bit. The hexadecimal representation of `je` is 0x74, while it is 0x75 for `jne`. After we have loaded and measured our login program, we insert our module, feeding it the virtual address of the `je` instruction in the running login program, as well, the process ID of the login program, and an overwrite value of 0x75. The module remaps the page table entry of the kernel to have one entry point to the target page, and writes 0x75 to the target physical address. When the program runs, the wrong instruction executes when we enter in a bad password, and we are given a shell. When the module is unloaded, everything is restored to its original state, and it looks to the outside world (and the TPM) like nothing has changed. Future measurements will not pick up on the fact that the program changed.

We use a "toy" example in our demonstration for the purposes of clarity, however this technique can be applied quite easily to any program. This attack shows that if a user has means to modify arbitrary regions of memory, they can render the measurements of the TPM useless, unless the TPM keeps continuous measurements of the loaded program's memory.

5.2 Reset attack

Figure 5 contains a diagram of how a trusted boot process takes place in the TCG architecture. Trusted boot is a key process in the architecture, and by compromising this process, we are able to compromise the entire TCG platform. Below is a diagram of how the trusted boot process occurs, once the TPM has been correctly initialized by the BIOS.

First, the BIOS feeds the TPM its own measurements, the TPM will then extend these measurement values in its PCRs. The TPM and BIOS then measure the boot loader, which reports back its measurement, and if that matches what the

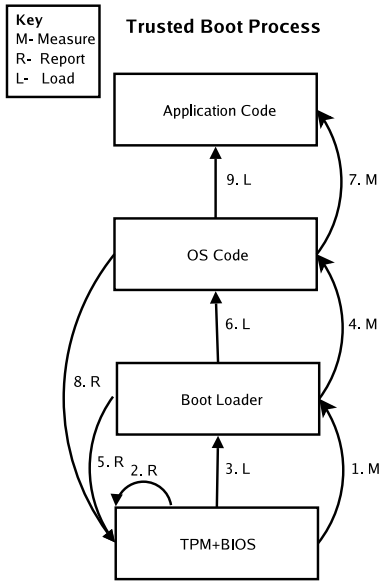

Figure 5: Trusted Boot Process

BIOS expects, it is allowed to load. The boot loader will then measure the OS and feed the measurements into the TPM, and only allow it to load if its measurements match what the boot loader expects. Each stage up the process is similar, and it is easy to see how we build a chain of trust where we finally have a trusted OS loaded. Applications built on top of this platform can then "know" that the system has followed this process by making sure the PCRs contain the correct values. If we assume that SHA-1 is a preimage resistant hash function, and that the PCRs cannot be easily reset, then we cannot fake this process.

However, if the TPM receives a hardware reset, independent of the rest of the system actually restarting, it will think that the system has been restarted, and will return to its uninitialized, inoperable state, before the BIOS first communicates with the TPM. That is, the PCR values will be reset, and we will be able to initialize the TPM using a malicious device driver. This process is shown in Figure 6.

In order to drive the hardware reset, we temporarily connect the LRESET line

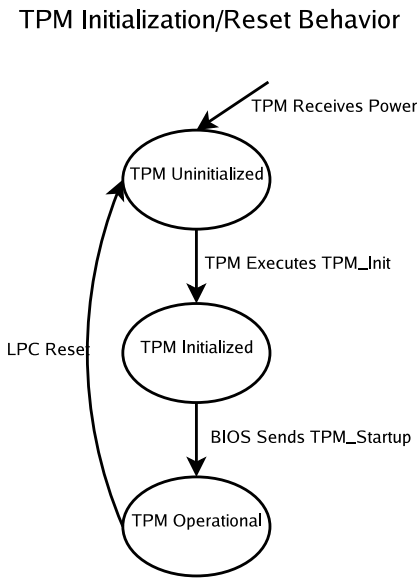

Figure 6: TPM Initialization/Reset Behavior

of the LPC bus to the GROUND line using a short piece of wire. This has the effect of resetting every device on the LPC bus, including the TPM. The TPM will then automatically execute the TPM_Init instruction and be initialized. If an attacker wanted to avoid resetting his or her keyboard and other devices on the bus, they could physically isolate the LRESET line which feeds into the TPM and only send the TPM a reset signal.

Once the TPM has been initialized, we must feed it a TPM_Startup(ST_CLEAR) command. We do this using a modified TPM device driver. Since there is no locality check in place for the TPM_Startup() command, we can execute this operation. We can then perform whatever TPM_Extend()'s are necessary to get its PCRs into the "trusted" state. The measurements required to do this could be recorded using a logic analyzer as shown by Kursawe et. al [8]. It may also be possible to figure out what these values would be by reverse engineering the BIOS and mimicking the way it takes measurements of the system to determine which measurements should be fed to the TPM.

Our aim here would be to record a trusted boot process of a particular system with a particular TPM. This "recording" will actually be a copy of the measurements reported to the TPM by the system during the boot process. With this recording in hand, our next step is to change the platform. To demonstrate the power of this attack, we could swap out the trusted hard drive of the machine, and swap in another hard drive running a different operating system. When the system reboots, the TPM will measure it, and it will be different from the trusted platform. Thus, the TPM will not be willing to disclose any attestation keys that are bound to that platform. The next step of our attack is to send the TPM a 'reset' signal, convincing it that the system is being restarted, and needs to be measured again. We will then feed the previously recorded measurements of the *trusted* system. When this process is finished, the TPM be in the same state that it was after the trusted platform was booted, but will actually be running in an untrusted configuration. With the TPM in this state, we will be able to get it to perform whatever operations we wish that are bound to the trusted platform.

We have demonstrated the possibility of issuing the TPM a reset command, and getting it into a state where it will extend arbitrary measurements into its newly reset PCRs. Kursawe et. al have demonstrated the possibility of recording the trusted boot process. By combining these two experiments, it is trivial to see that this attack is possible.

5.3 Timing attack

In order to pull off Kocher's timing attack, we need to be able to take high resolution timing samples of how long it takes for a TPM operation to complete while it is executing a cryptographic operation which uses a private key.

This attack is based on Brumley and Boneh's [3] attack against a version of OpenSSL which employed the CRT to perform the RSA decryption operation.

We chose this method because according to the spec sheet of the TPM we used, the TPM employs CRT for RSA decryption. [13] By performing a "TPM_Seal" operation on a series of specially crafted input strings, we are able to measure differences in the amount of time it takes to complete each operation (regardless of whether the operation fails), we should be able to iterate through and successfully "guess" each bit of the key.

Unfortunately, successful completion of this attack requires approximately 2100 timing samples per bit, and since each timing sample takes approximately 0.8 seconds to complete, this attack will take close to 40 days to complete, a time frame which is out of the scope of this paper.

However, this attack should be theoretically possible since we can see that RSA operations on different inputs take variable amounts of time. In order to see this, we first crafted two inputs, which, according to Brumley and Boneh's attack should yield different average execution times for a TPM_Unseal operation. We then ran the operation on each input approximately 2400 times, measuring the execution time for each. The distributions of these two groups of samples are approximately normal. After removing outliers, a T-Test was then run against the two distributions to ensure that the difference between their means is statistically significant. The resulting p-value of 0.03 indicates that we can reject the null hypothesis that these two distributions are the same. This indicates that timing attacks should be possible against TPMs. We also observed that RSA operations on the same inputs take the same amount of time using a similar method, indicating that RSA blinding is not happening.

6 Countermeasures

In this section, we discuss a variety of countermeasures which may help prevent the attacks described above, and enhance the overall security of the TCG architecture.

6.1 Software defenses

The core problem with the Remote Attestation model is that if memory is changed after it is measured, an attacker can run untrusted code. One possible solution here is to build a system in which the attacker *cannot* change memory after it is measured. In the real world, however, this is a difficult proposition. A computer has to be able to read and write its memory in order to function, and it is difficult to enforce this policy. Secure page tables may be an appropriate solution to the system above, but they do not protect against every possible avenue of attack. In versions of the Linux kernel prior to 2.6.5, a root user could write to arbitrary segments of the `/dev/mem` character device, which is a device which contains the system's physical memory. Overwriting memory at the right location is equivalent to the attack demonstrated above, but requires no modification of page tables.

Instead of trying to restrict the usability of a platform to users and platform owners by prohibiting modification of system memory or page tables, the system should simply *monitor* the state of memory of the application it is attesting to. If some protected memory of a particular program changes, the system will notice this and incorporate the change into its measurements. Optionally, the system could actively report such a change to the operating system or the challenging service provider.

Nihal D'Cunha has proposed such a system which employs use of the Xen hypervisor system to monitor the status of a particular program's memory. [5] If a

protected segment of memory is modified, the system detects it, and the relevent PCRs are updated. Thus, the change is reflected in the system, and cannot easily be hidden.

6.2 Reset defenses

The TPM is vulnerable to a reset attack because of the assumption that the TPM's PCRs cannot be easily reset without resetting the entire system. The other crucial assumption is that only the BIOS will be initially feeding measurements to the TPM. Both assumptions are erroneous. With physical access to the LPC bus, it is trivial to drive a reset across just that bus, and not the entire system. Further, if one can isolate the TPM on that bus, it is also trivial to send just the TPM the reset signal.

Bernhard Kauer has also produced OSLO, a secure bootloader which employs AMD's `sk_init` command to bring the PCRs into a special state. This works because the processor transmits a special signal to the TPM along with its measurements which indicates that it is in secure initialization mode. While this is hard to fake from the operating system level, an attacker who could inject arbitrary signals onto the LPC bus could easily fake this command as well, defeating this defense. We have prototyped a board to allow this, but time ran out before we could finish debugging it.

If the TPM is integrated on die with the microprocessor or the BIOS, it will be much more difficult to physically access the lines leading in and out of the chip. It seems to make most sense to join it with the BIOS, since the BIOS and TPM are inextricably linked as the"Core Root of Trust" in the TCG architecture. To address the problem in systems which already exist, good locks on cases and epoxy over anywhere where lines to the LPC bus are exposed will help minimize the threat of physical attack against the LPC bus. Additionally, if the TPM and BIOS can have

an encrypted communication session such that the TPM can positively identify that it is only speaking with the BIOS, then it would be trivial to enforce that only the BIOS can issue a TPM_Startup command and feed the TPM its first measurements. Currently, this ability does not exist in the TPM specification.

6.3 Timing defenses

In order to appropriately defend against timing attacks, TPM vendors should either employ RSA blinding [10] in their implementations, or employ a modular arithmetic core which executes modular exponentiation in constant time. Alternatively, TPMs could return results of all private key operations in constant time. These countermeasures would not defend against other kinds of side-channel attacks, such as differential power analysis. However, it remains to be seen whether TPMs are susceptible to such attacks.

7 Future Work

This research leaves the door open for future researchers to continue to explore a number of attacks and defenses on TPMs and the TCG architecture. We mention a number of potential avenues for research in this section.

7.1 Software

In our software attack, we only discussed attacks against the Linux operating system equipped with a TPM. Similar (and in fact, easier) attacks exist against the Windows operating system, since writing to the `PhysicalMemory` device is possible for device drivers. This attack is the same as the attack against `/dev/mem` described above.

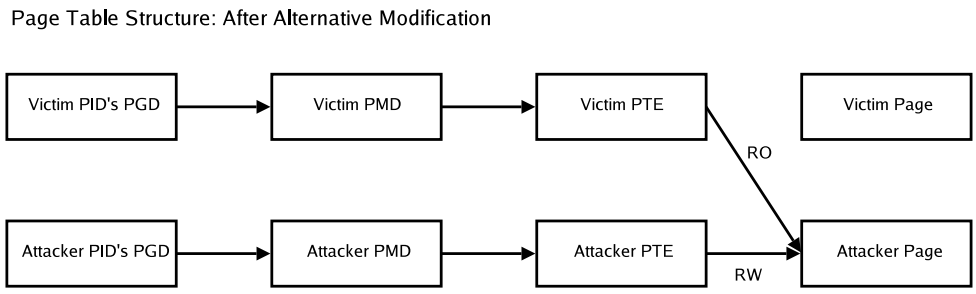

Figure 7: Another Proposed Page Table Attack

Another possible attack against the Linux page table system would work slightly differently from the one mentioned above. Instead of updating the value stored in physical memory at the same location that the process points to, we instead update the process's page table to point to a different page of physical memory that our kernel module has already written to. An illustration of the after-effect this attack can be seen in Figure 7. This attack would not be caught by the monitoring system described above, because the memory at the original physical location has not been modified. Only the process's page table has been updated. A mechanism to monitor that a process's page table has not been updated would be necessary to protect against this type of attack. Nihal D'Cunha has also suggested this defense and implemented a solution based on the XEN hypervisor system. This solution combined with the one mentioned above would protect against all types of page table based TOCTOU attacks on TPMs.

Future researchers might also explore different means of modifying memory after it has been measured. A hardware switch which swaps out one stick of DRAM for another while the computer is running might be a feasible attack. Using DMA to circumvent the control that the operating system has over system memory after that memory has been measured is also a possibility.

7.2 Hardware

Bernhard Kauer proposes a solution to reset attacks based on locality on LPC bus traffic. However, none of the LPC bus traffic is encrypted or authenticated. Thus, anyone with access to the bus could build a device which feeds data with the appropriate locality to the TPM. An FPGA equipped with an LPC interface would be sufficient for such an attack.

Future researchers may also want to look at a less expensive and more simple way of recording the trusted boot process. If it is possible to do this step with cheap, consumer level hardware, the door is open to just about anyone to attack these systems, not just those individuals with access to a logic analyzer. It may also be a good idea to look at other implementations of TPMs other than just those designed by STMicro. We chose STMicro for the purposes of this paper not because we thought their implementation was bad, but rather because they appear to be the largest TPM manufacturer for consumer-grade products.

7.3 Power

Another avenue of exploration would be the feasibility of performing Paul Kocher's Differential Power Analysis attack against a TPM. In our lab, we have an Agilent Infiniium series oscilloscope. This device, combined with a 5-ohm resistor on the power line leading to the chip could measure the changes in current in the chip. This power line could then be connected to a stable external DC power supply (Agilent model E3631A), which will provide a steady 3.3V of power, as opposed to a more noisy traditional PC power supply. This experimental would allow for accurate measurement of power usage by the TPM during cryptographic operations.

8 Conclusions

We have demonstrated a number of attacks on current TPM implementations and the architectures which employ them. Specifically, we have shown that because the TPM does not monitor running processes after it has measured them, the IMA architecture is broken. We have also reproduced the TPM_Reset() attack originally carried out by Bernhard Kauer and shown that it poses a threat to the system. We have also demonstrated that RSA timing attacks against TPMs should be possible.

The ability to securely authenticate a system, rather than a user of that system, is a necessity for the future of computing in an interconnected world. Rootkits, virtualization, and many realities of modern computing make it difficult to authenticate a platform from the application level going down to the hardware. It is necessary to have some kind of hardware that can authenticate the configuration of a particular platform. TPMs and the TCG architecture provide a good starting point for this.

While no perfect solution to this problem exists, there are a number of solutions out there that can improve the security of a system. The TCG architecture combined with TPMs is just one such proposed solution. Unfortunately, there are a number of vulnerabilities in this system which allow an attacker to circumvent the protection that the it provides. Going forward, it is important that those considering platform deployment and those designing the TCG specification keep in mind reasonable hardware protection systems. Specification designers should also remember that if all it takes is a single software bug to compromise the entire system, then perhaps a design more resilient to partial failure should be considered.

9 Acknowledgments

This research was sponsored in part by the National Cyber Security Division of the Department of Homeland Security (under Grant Award Number 2006-CS-001-000001) and by the National Science Foundation (under grant award CNS-0524695). The views and conclusions do not necessarily represent the sponsors.

I would like to thank Sean Smith, Ted Cooley, and Steve Weingart for their help with this research project. I would also like to thank lab mates Nihal D'Cunha, John Baek, and Patrick Tsang for providing some key insights into understanding the systems I explored. Finally, I would like to thank my family and friends for their continued support throughout the years which no doubt made this thesis possible.

References

[1] Google Search For: Linux Kernel Vulnerabilities.

[2] Bernhard Kauer. OSLO: Improving the security of Trusted Computing. Technical report, Technische Universitt Dresden, Department of Computer Science, 2007.

[3] D. Boneh and D. Brumley. Remote Timing Attacks are Practical. In *Proceedings of the 12th USENIX Security Symposium*, 2003.

[4] David Safford, Mimi Zohar, Alan Boulanger. Trusted Computing For Linux. Slideshow Presentation.

[5] Nihal A. D'Cunha. Exploring the Integration of Memory Management and Trusted Computing. Technical Report TR2007-594, Dartmouth College, Computer Science, Hanover, NH, May 2007.

[6] Dyer, J.G.; Lindemann, M.; Perez, R.; Sailer, R.; van Doorn, L.; Smith, S.W. Building the IBM 4758 Secure Coprocessor. *Computer*, 34:57–66, 2001.

[7] Intel Corporation. Intel Low Pin Count Interface Specification. Technical report, Intel Corporation, 2002.

[8] Klaus Kursawe; Dries Schellekens; and Bart Preneel. Analyzing trusted platform communication. In *CRASH Workshop: CRyptographic Advances in Secure Hardware*, 2005.

[9] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, pages 388–397, London, UK, 1999. Springer-Verlag.

[10] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 104–113, London, UK, 1996. Springer-Verlag.

[11] Pearson, Siani. *Trusted Computing Platforms*. Prentice Hall PTR, Upper Saddle River, 2003.

[12] Ahmad-Reza Sadeghi, Marcel Selhorst, Christian Stuble, Christian Wachsmann, and Marcel Winandy. Tcg inside?: a note on tpm specification compliance. In *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, pages 47–56, New York, NY, USA, 2006. ACM Press.

[13] STMicro Electronics. ST19WP18-TPM-A Datasheet, 2005.

[14] TPM Work Group. TCG TPM Specification Version 1.2 Revision 94. Technical report, Trusted Computing Group, 2006.

[15] Trusted Computing Group. TCG Specification Architecture Overview Revision 1.2. Technical report, Trusted Computing Group, 2004.