

UCLinux: a Linux Security Module for Trusted-Computing-based Usage Controls Enforcement

David S. Kyle and José C. Brustoloni
University of Pittsburgh
Computer Science Department
6429 Sennott Square
Pittsburgh, PA 15260, USA
{dkyle,jcb}@cs.pitt.edu

ABSTRACT

Usage controls allow the distributor of some information to limit how recipients of that information may use it. The Trusted Computing Group has standardized Trusted Platform Modules (TPMs) that are built into an increasing number of computers and could greatly harden usage controls against circumvention. However, existing operating systems support TPMs only partially. We describe UCLinux, a novel Linux Security Module that, unlike previous work, supports TPM-based attestation, sealing, and usage controls on existing processors and with minimal modifications in the operating system kernel and applications. Experiments show that UCLinux has modest impact on the system's boot latency and run-time performance.

Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection—*Information flow controls*; D.4.6 [Operating Systems]: Security and Protection—*Cryptographic controls*

General Terms

Security, Design, Performance

Keywords

Trusted Computing, Usage Controls, Trusted Platform Module, TPM, Trusted Computing Group, TCG, Linux, UCLinux, Linux Security Module, LSM, Encrypted File System, Open Digital Rights Language, ODRL

1. INTRODUCTION

Usage controls allow the distributor of some information to limit how recipients of that information may use it. For example, the distributor of a document may allow recipients to read but not to print or to copy the document.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

STC'07, November 2, 2007, Alexandria, Virginia, USA.
Copyright 2007 ACM 978-1-59593-888-6/07/0011 ...\$5.00.

Usage controls can be useful in many applications. For example, usage controls can help ensure that recipients comply with non-disclosure agreements or regulatory data-handling requirements, such as those set by HIPAA (Health Insurance Portability and Accountability Act) [17]. However, effective support for usage controls is still rare in computer systems.

Some existing applications do provide usage controls. However, those controls are software-based and usually can be reverse-engineered and circumvented. For example, Adobe Acrobat allows the creator of a PDF document to encrypt it and disallow its printing. Recipes for circumventing such controls can be easily found on the Web [4].

The Trusted Computing Group [15] has standardized Trusted Platform Modules (TPMs) [16] that are built into an increasing number of computers and could greatly harden usage controls against circumvention. TPMs provide a hardware-based root of trust that enables two new primitives, attestation and sealing [1]. The distributor of some information can use attestation to obtain assurance that the recipient's system configuration is trusted, before sending that information to the recipient. A trusted configuration is one that (1) seals information and respective usage policies to the configuration that was attested to the information's distributor, and (2) enforces the distributor's policies for how information provided by the distributor may be used, even if the recipient strives to do otherwise. Sealed information is encrypted with a key that the TPM will not reveal if the system configuration differs from the one attested to the distributor.

Both attestation and sealing require operating system modifications that are still lacking in mainstream operating systems. Microsoft Vista [8] can use TPMs for secure boot and whole-disk encryption, but it does not support attestation. TcgLinux [11] is a version of the Linux operating system that supports attestation, but not sealing. Enforcer [5] is a Linux Security Module (LSM) [3] that supports sealing and a form of certificate-based authentication that could be used in lieu of attestation. However, Enforcer is unable to prevent circumvention of distributors' policies by recipient system administrators. Microsoft Next Generation Secure Computing Base (NGSCB) [7] would support both attestation and sealing. However, it requires extensive hardware, operating system, and application modifications and appears to have been discontinued.

This paper describes UCLinux, a novel LSM that, unlike previous work, supports attestation, sealing, and TPM-based usage controls on existing processors and with mini-

mal modifications in the operating system kernel and applications. Experiments show that UCLinux has modest impact on the system’s boot latency and run-time performance.

The rest of this paper describes the design and implementation of UCLinux. Section 2 reviews related work, and Section 3 presents our threat model. Section 4.1 describes UCLinux’s usage-controlled file system. Section 4.2 describes the *prelog* system UCLinux uses to enable sealing. Section 4.3 explains the *root tripping* process UCLinux uses to protect usage-controlled files from administrator access. Section 4.4 describes the UCLinux boot process. Section 4.5 describes UCLinux’s encoding of usage policies. Section 4.6 summarizes how UCLinux enables transmission of usage-controlled files. Section 5 discusses the LSM that implements UCLinux. Section 6 reports results of experiments conducted to evaluate UCLinux’s performance. Finally, in Section 7, we conclude and discuss future work.

2. RELATED WORK

TcgLinux modifies the operating system kernel to implement an Integrity Measurement Architecture (IMA) [11]. IMA measures and extends into the TPM the secure hash of every executable file in the system, when the file is first executed. Such applications may also measure and extend into the TPM the secure hash of configuration or other files they depend on. IMA’s extensive kernel modifications support attestation by ensuring that TPM state is always consistent with the system’s actual configuration. IMA changes the TPM state for every new file that is executed, including files that are irrelevant to system security. These changes are order-dependent, even when execution order does not affect system security. This order-dependency makes IMA incompatible with sealing, because information can be sealed to only one specific TPM state. To overcome this limitation and support both attestation and sealing, UCLinux uses Trusted Computing Base (TCB) *prelogging*, as discussed in Section 4.2.

Enforcer [5] uses the TPM to seal an encrypted file system to only part of the system’s configuration. This part includes the BIOS, boot loader, operating system kernel, and Enforcer LSM, but not daemons and root setuid or other applications. A file signed by a trusted third party can configure expected secure hashes of applications and constraints on what applications can open specified files. A trusted authority can certify such a recipient configuration, and distributors could verify such certificates instead of attestation. However, Enforcer provides no guarantee against recipient system administrators. For example, after the recipient system boots and mounts the encrypted file system containing information received from a distributor, a system administrator can log into the system interactively and tamper with Enforcer’s operation so as to circumvent the distributor’s usage policies. UCLinux uses root tripping, described in Section 4.3, to prevent similar tampering.

Trustworthy SELinux [18] implements usage controls on top of IMA [11] and SELinux [9]. SELinux is an LSM that extends Linux’s access control architecture. Conventionally, Linux offers only discretionary access control (DAC). In DAC, any user can arbitrarily grant access to objects the user creates, but such policies are not enforced against privileged users. On the contrary, SELinux provides mandatory access control (MAC). In MAC, once set, access policies

cannot be overruled by users, even privileged ones. Consequently, it may be easier to prevent circumvention of usage controls in a system with MAC than in one with DAC. Trustworthy SELinux includes mechanisms for automatically translating usage policies written in XACML into SELinux Loadable Policy Modules (LPMs). These mechanisms facilitate usage control configuration and enable usage-controlled objects to be accessible while the system has privileged interactive users. However, users often find SELinux difficult to manage and use, and IMA does not allow information to be sealed.

Trusted SECTET [6] builds upon Trustworthy SELinux with a higher-level access control expression language, SECTET-DSL. SECTET-DSL (Domain Specification Language) provides an abstraction layer useful to domain experts and business experts, with automatic translation to XACML. This system provides an intriguing alternative to the ODRL system we use, especially in database systems.

IBM’s vTPM [13] virtualizes TPMs so that multiple virtual machines can securely use a same physical TPM. A virtual machine monitor (VMM), such as Xen [2], can preserve isolation between virtual machines (VMs). UCLinux could be used to enable attestation and sealing in VMs running on Xen and vTPM.

3. THREAT MODEL

UCLinux assumes that computer hardware and administrative rights may be owned by someone who seeks to violate usage controls. A “win” for such an attacker includes gaining unrestricted or less restricted access to a usage-controlled file, or modifying the usage policy specified by a file’s distributor. UCLinux assumes that the attacker may use any software-based tools and may move to other computers storage devices containing usage-controlled files. UCLinux assumes, however, that the attacker lacks tools or expertise to perform hardware-based attacks, such as snooping on the system’s bus, timing analysis, or examining a computer’s TPM in an electronic microscope.

4. DESIGN

4.1 Usage-controlled file system

This section describes UCLinux’s storage for usage-controlled files.

UCLinux stores usage-controlled files in a usage-controlled file system (UCFS). A UCFS differs from a conventional file system in three ways. First, each file has an associated usage policy. Usage policies can be stored as file metadata, e.g., Linux extended attributes. Second, files and usage policies are encrypted with a secret key. This can be accomplished by storing them in a conventional encrypted file system (EFS). For example, Linux 2.6 supports loopback file systems and *dm-crypt*, which can be used for this purpose. Third, the EFS’s secret key is sealed to a trusted configuration. A trusted configuration is one that does not allow privileged or unprivileged users to circumvent the system’s usage controls.

4.2 TCB prelogging

This section describes the method UCLinux uses to enable TPM-based data sealing.

A trustworthy system derives its trustworthiness from its components. In our system, those components consist of the

BIOS, the boot-loader (GRUB), the initial RAM file-system (initramfs) bootstrap files, the Linux kernel and its modules, UCLinux, and a subset of the executable files and shared libraries present in the system. Together, these comprise the Trusted Computing Base (TCB). Unprivileged applications, such as word editors, need not be trusted; therefore, we do not include every executable in the TCB. Likewise, only shared libraries used by trusted executables need to be present in the TCB. To configure which executables and shared libraries are part of the TCB, UCLinux uses a file called the TCB list. The TCB list contains a list of trusted files, including the respective full path-name and SHA-1 secure hash.

At boot time, UCLinux extends into the TPM all secure hashes contained in the TCB list. The resulting TPM state does not depend on execution order. System secrets (e.g., the secret key of the system's UCFS) can be sealed to this predictable state.

4.3 Root tripping

This section describes how UCLinux makes TCB prelogging compatible with attestation.

With TCB prelogging, attestations reveal a system's entire expected TCB, including TCB components that have not yet been loaded in memory and whose secure hashes have not yet been actually measured. If a system has been compromised, its actual configuration may differ.

UCLinux uses *root tripping* to ensure attestations are trustworthy. Before UCLinux transfers CPU control to any application with privileged effective user, UCLinux verifies that the application is in the system's TCB list, and the application's actual secure hash matches the value in the TCB list. When UCLinux opens for such a process any file (e.g., configuration) included in the system's TCB list, UCLinux also verifies that the file's actual secure hash equals the value in the TCB list. If there is a discrepancy between actual and expected secure hashes, UCLinux extends into the TPM the actual secure hash and drops all security associations.

In UCLinux, *security associations* are any objects that the system should be able to access only while it is in a trusted configuration, as represented by the TPM state. Security associations may include, for example, the system's UCFS or trusted network connections.

Once all security associations have been dropped successfully, the program that caused the trip executes normally. On the other hand, if any security association lingers, the tripping program will sleep indefinitely, and the privileged user will be unable to execute any untrusted programs. The computer may ultimately crash, but untrusted processes will be unable to violate usage controls. Upon reboot, however, the computer will once again enter a trusted configuration, and be able to access usage-controlled files.

UCLinux treats similarly the logging into the system of an interactive privileged user (e.g., root or other administrator). Existing systems typically give privileged users unrestrained access to any objects. It would be hard for UCLinux to ensure that no interactive command can be exploited by a malicious administrator to circumvent usage controls. To prevent such circumvention, when a privileged user logs interactively into the system, UCLinux extends into the TPM an entry reporting this event and drops all security associations. Because extensions change TPM state, after root tripping the system will typically have to be rebooted into

the trusted configuration before the system can regain access to dropped security associations.

UCLinux employs a *shepherd process* to manage and protect security associations. This process is responsible for gaining access to a security association, and later dropping that access when a root trip occurs. UCLinux defines three primitives: *register*, *start*, and *finish*. Each of these primitives communicate with the kernel to perform their respective functions. The shepherd process for UCLinux's UCFS is called `mount-efs` (for file security daemon).

A shepherd process uses *register* when it first starts up, before it performs any security-sensitive actions. This communicates to the kernel that the process is a shepherd process, and must be allowed to drop its security association before a root trip finishes. Once register finishes successfully, the shepherd process can perform whatever work is necessary to access the security association. For example, `mount-efs` reads the UCFS's secret key from `/etc/crypt.key`, unseals it, and pipes it into `cryptsetup`, which sets up a cryptographic loop-back device, and mounts the file system. After its security association is accessible, a shepherd process uses *start*; the kernel puts the shepherd process to sleep, and only wakes it up when a root trip occurs.

When a shepherd process wakes up, it must drop its security association. For example, `mount-efs` unmounts the UCFS and destroys its cryptographic loop-back device. If a shepherd process is sure that its security association is fully protected, it uses *finish* to communicate to the kernel that it is safe to finish the root trip, and allow untrusted programs to run with a privileged effective user. If the shepherd cannot be certain that its security association is dropped, then it simply exits.

Any number of security associations can be protected in this manner. On a root trip, the kernel wakes each shepherd process, and ensures that each one calls *finish* before it completes the root trip.

4.4 Boot-up Process

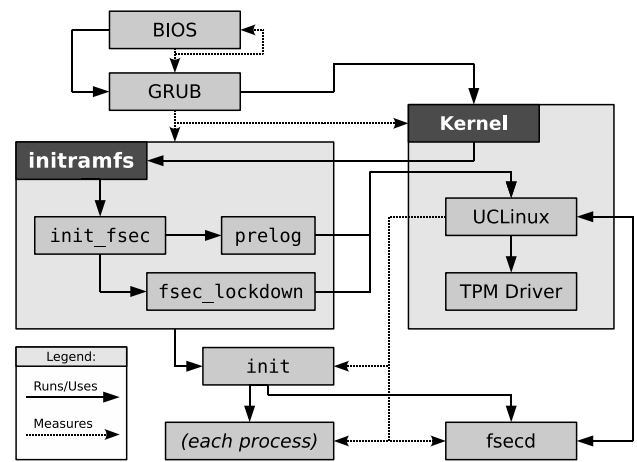


Figure 1: Boot-up process

This section describes UCLinux's boot-up process.

UCLinux uses an *authenticated boot* scheme. UCLinux's boot process extends into the TPM the secure hashes of each software component that could affect the trustworthiness of the system. If any of the components has been modified

(perhaps maliciously), its actual secure hash is extended into the TPM; the system still boots, but the TPM does not reveal sealed data, such as the UCFS's key. This differs from a *secure boot* scheme, such as that employed by Microsoft's Windows Vista operating system. Secure boot would cause the operating system to refuse to start if any secure hash is unexpectedly modified.

When a computer with UCLinux first boots up, the BIOS measures itself, its configuration, and the boot loader and extends those measurements to the TPM, in accordance with TCG specifications. The boot loader, a version of GRUB, then measures the Linux kernel and the initramfs. The initramfs consists of a gzipped CPIO (similar to tar) archive which contains a minimal root file-system. Conventional Linux systems use the initramfs to load kernel modules necessary for mounting the real root file system. Our modified initramfs adds the TPM driver (modified by us to provide an interface for the kernel to extend to the TPM), the UCLinux LSM, two programs which initialize the TCB, `prelog` and `fsec_lockdown`, and a script which drives the initialization, `init_fsec`.

On boot, the initramfs boot process runs normally, loading the kernel modules it contains, until just before it switches the root file system from itself to the real root file system. At that point, it executes `init_fsec`, which runs `prelog` and `fsec_lockdown`. `prelog` reads the TCB list from file `/etc/prelog`, stored within the the real root file system, and loads each of its entries' values into the kernel, which in turn extends the hash values into the TPM.

Once `prelog` is finished and the real root file system is mounted into place, `fsec_lockdown` tells the kernel to start enforcing root tripping (Section 4.3). We designed the initramfs to run without user input, and without possibility of interruption. Therefore, a single PCR extension by GRUB representing the initramfs is sufficient to guarantee the system is in a trusted configuration once the initramfs finishes.

When the initramfs finishes, the system boots up normally, and eventually runs an `init.d` script, which executes `mount-efs`. The latter program manages the UCFS, ensuring that the UCFS is available only while the system is in a trusted configuration, as explained in the previous section.

4.5 Usage policies

This section discusses how UCLinux represents usage policies.

UCLinux represents usage policies in two forms, an XML-based Open Digital Rights Language (ODRL) derivative plaintext format, and extended attributes applied to files in a more-easily machine readable format. ODRL is an open standard which represents digital rights through sets of permissions granted to various parties to access and use content under certain constraints. For example, a PDF document could only be viewed by a particular person, but not be printed. We extended it by adding an `open` permission, which is enforced by the kernel, and by adding a digest to the `party` identification tag, so that TCB programs, identified by their digest, can be party to the ODRL, instead of just people. For example, instead of a particular person being able to view a PDF document, only a particular version of `xpdf` could be used to view it, while being unable to print it.

While ODRL was designed to be machine readable, it is

not well suited for interpretation by an operating system kernel. Just parsing XML requires a substantial amount of code, and interpreting the possibly complex nuances of ODRL would require even more; therefore, we use user-space code to extract the kernel-pertinent information from the ODRL, and put into a form easily understandable by our kernel module. To perform this action, we wrote a shared library, `libchfsec` (CHange File SECurity), which is used by certain programs called *embedders*.

Each extended attribute begins with a “user.frights.” prefix. Attributes with this prefix can only be added, modified, or removed by trusted processes. We use the following extended attributes to record policy data:

user.frights.apps.i.hash The library stores the hash of each program, with a different number for *i*, for which the ODRL specifies an `open` permission in this attribute.

user.frights.apps.i.name The library stores a plain, human readable, name for each program. This information is not used by the kernel.

user.frights.apps.i.uid The library stores a computer, and human readable, identifier for the specific version of the program in this attribute. (e.g., “application/gnome/gpdf/2.14” for gPDF version 2.14”). This information is not used by the kernel.

user.frights.apps.n The library stores the number of programs listed using the above attributes in this attribute.

user.frights.perms.open The presence of this attribute indicates that the file must be protected by the kernel. Other potential permissions could be listed as attributes prefixed with “user.frights.perms.”, but only “open” is recognized by the kernel.

user.frights.perms.open.expdate The expiration date of the file, if applicable, is listed here. Other ODRL constraints could be listed similarly. The only other constraint enforced by the kernel is the “start date” constraint, which ends with “.startdate”, and governs when a file can first be opened.

Embedders are responsible for receiving usage-control protected files, saving them in the UCFS and applying the proper policy extended attributes, as specified by an accompanying ODRL policy. The kernel only enforces the “open” permission, so when embedders are applying usage policy to a file, `libchfsec` examines the ODRL and determines what programs it gives that permission to, and what expiration/start time constraints are indicated, and puts that information into the file's extended attributes. The rest of the ODRL information is put into a binary format¹, compressed, and stored in an extended attribute. We expect that user-space programs, responsible for enforcing that remaining policy information, can directly use ODRL.

¹We designed our own binary format for ODRL. Once binary XML is well standardized, it should be used instead. We use a binary format because Linux extended attributes are limited to one file system page.

4.6 Attestation

This section summarizes how UCLinux transmits files with usage policies.

To allow distributors of information subject to usage controls to verify the trustworthiness of a UCLinux-based platform, UCLinux uses a version of the TLS v1.0 protocol [14], extended to include attestation during its handshake [10].

The TLS v1.0 protocol is the standard for secure web-based communication. It works on existing TCP sockets and provides data confidentiality, data integrity, and certificate-based server authentication and optional client authentication. Two communicating hosts can initiate TLS by executing the TLS handshake. The handshake consists of four stages. Stage 1 is for negotiating handshake parameters, such as TLS version, encryption and digital signature algorithms. During stages 2 and 3, the hosts perform server authentication and optional client authentication, and compute a shared session key for symmetric encryption and authentication. The final stage verifies the integrity of all handshake messages and asserts negotiated security parameters. An extension of the TLS v1.0 protocol provides a framework for exchanging additional information in the client hello and server hello messages. For example, TLS extensions are used to pass additional parameters for the Elliptic Curve Cryptography (ECC) encryption algorithms.

UCLinux uses the existing TLS extensions framework [12] to incorporate platform attestation in the TLS protocol. The framework allows adding new extensions without breaking backward compatibility. The extension adds two additional handshake messages for attestation exchange, taking place during a new stage inserted between stages 3 and 4.

The first message is sent from the client to the server. This message includes the client's attestation identity key (signed by a mutually trusted PrivacyCA), a TPM-produced quote which lists the current PCR values (signed with the identity key), and a measurement log. Once that message has been received, the server verifies the identity key's signature, the quote's signature, and checks that the measurement log is consistent with the PCRs listed in the quote, and checks that the measurement log matches a known trustworthy configuration.

5. IMPLEMENTATION

This section describes UCLinux's implementation as a Linux kernel module.

The core of the system is the `uclinux.ko` kernel module. Primarily a Linux Security Module, it performs other functions as well. The LSM system provides a broad set of hooks into various operating system functions that enable an LSM to provide security models other than the usual UNIX permissions system allows. For example, SELinux is implemented as an LSM. The LSM portion of our module intercepts all attempts to open files, write extended attributes, execute programs, mmap files, and attach to processes for debugging.

5.1 Program execution hook

When a program executes, the LSM first examines it. The LSM searches a hash table using the full path of the program file as key. Each table entry contains the most recent actual value and timestamp of the secure hash of the entry's file. If the LSM has not measured yet that file, or the file has been

more recently modified, the LSM measures the file's secure hash and stores it in the respective table entry. If the measurement has the value expected according to the TCB list, the process is marked as trusted. Otherwise, if the program has privileged effective user, then the kernel trips, blocking the process until all security associations can be confirmed as closed. Similarly, whenever a trusted process mmmaps a file with executable permission (i.e., a shared library), the module performs the same operations on that file, and trips if needed. This ensures that trusted programs will only load trusted shared libraries.

5.2 File handling hook

Whenever a process opens a file, the `inode_permission` hook triggers, and the LSM examines the extended attributes of the file. If they indicate that the file is protected, the LSM checks if the current process is part of the TCB, and if it is, whether the process's hash matches any of the hashes listed in the file's extended attributes. If it matches, the process is allowed to open the file. If not, access is denied. Once the trusted program has been allowed to open the file, its own content protection code must enforce any other usage restrictions specified in the file's policy. The LSM does not itself enforce application-specific usage policies. The LSM enforces only whether particular programs can open a file. The LSM also checks permissions, in the same manner, with the `file_permission` hook, which triggers any time a file is read or written to.

5.3 Extended attributes hook

If a process attempts to modify an extended attribute, the LSM checks whether the attribute is relevant to UCLinux (such attributes have a `user.frights.` prefix). If it is, then the system checks to see if the process is part of the TCB, and has had the `embedder` flag set. If it has this flag, then the modification is permitted. Otherwise, access is denied. Because extended attributes are essential to the run-time enforcement of usage policies for protected files, only trustworthy programs should be able to write them. We believe only permitting a subset of programs in the TCB to modify file policies will make it easier to develop secure TCBs; however, this precaution is not strictly necessary. Reading the extended attributes has no security concerns; therefore, no restriction is applied.

5.4 Program tracing hook

If a process attempts to use the `ptrace` system call to debug another process, the system checks whether the tracee is part of the TCB. If it is, and the tracing process is *not* in the TCB, then `ptrace` fails. If an untrusted process were able to successfully trace a trusted process, the trustworthiness of the latter could not be guaranteed. We allow trusted processes to use `ptrace` on trusted processes because some important Linux systems rely on innocuous uses of `ptrace` (e.g., the `/proc` filesystem). Naturally, these trusted `ptrace` users must be incapable of abusing the powerful potential of that system call.

5.5 User-space communication

In addition to the LSM functionality, the module also provides a means for the user-space support programs (e.g., `prelog`, `fsec_lockdown`, `mount-efs`), to communicate with the kernel via a character device file (`/dev/fsec`). Those

programs execute the `ioctl` syscall to load prelog information, lock down the kernel, or communicate security association primitive commands. The module also provides `/proc/measurement_log`, which can be read to obtain a current measurement log for purposes of remote attestation.

5.6 Kernel source changes

There are a few changes we made to the Linux kernel source itself. We added an internal kernel interface to the TPM driver included in the mainline source, to allow `uclinux.ko` easier access to the TPM hardware. With a normal Linux kernel, data can be leaked when a process releases a page of memory, and the kernel allocates that page to another process. To protect usage-controlled data, we modified the memory management code to zero out memory pages whenever the kernel returns memory allocated to trusted processes back to the free-list. Finally, we completely disabled core dumping for trusted processes, since a core-dump file would contain all of the data currently in memory of a crashing trusted process.

5.7 Generating a prototype configuration

We built the system around a prelog file, which contains the file-name, mode and hash of every member of the TCB. As this file contains every program which must run as root, and every shared library those programs load, generating this file is not trivial. Our implementation provides a shell script which makes generating this file easier, but still time-consuming. These instructions generate a prelog file which will allow the computer to boot without tripping:

1. Run `prelog_initialize`, provided in our implementation. This scans the output of `dmesg` for any TRIP messages, and adds the files that caused them to the prelog.
2. Reboot the computer. As it boots, many "TRIP" messages will flash by.
3. Once the computer has booted up, repeat this process from step 1. Stop when the system no longer trips. View `/var/log/messages` to be sure there no longer are any TRIP messages. `/mnt/efs` should be mounted.

Once a prelog file has been generated, the system will be able to run.

A UCLinux-based system will not be secure unless every entry in the prelog has been thoroughly vetted for potential breaches of the usage control system. Performing this task for a complete system is outside the scope of this paper.

6. EXPERIMENTAL RESULTS

This section reports the results of experiments conducted to evaluate a preliminary version of UCLinux.

6.1 Speed

We performed two performance tests: boot-up time and kernel compilation time. We ran both tests on an IBM ThinkCentre, with a 3.00 GHz Pentium 4², and 512 MB main memory. We used the Ubuntu 6.06 (AKA, "Dapper Drake") GNU/Linux distribution.

²Model RK80532PG080512, includes hyper-threading, 800 MHz front side bus, and a 3 GHz 512 KB L2 cache

To evaluate the impact of TCB prelogging on boot-up time, we measured the amount of time the computer takes to start up with and without the UCLinux system. For each trial, we started from a cold boot, and began timing from the moment of hitting "enter" in GRUB, to the moment the GDM log-in screen appeared and was ready for input. For the trials without UCLinux, we used a stock Ubuntu `initramfs`.

We ran three trials each with and without UCLinux. We found that without UCLinux, the system booted in an average 51.16 seconds, with standard deviation 1.31 seconds, and with UCLinux boot-up took an average of 55.93 seconds, with standard deviation 2.05 seconds. UCLinux increased boot-up time by about 9%.

UCLinux can affect system performance after boot-up in two ways. First, UCLinux uses encrypted storage and therefore imposes cryptographic overheads. Second, UCLinux's run-time interception of the various system calls, such as executing and opening files, can make the such calls more time-consuming. To evaluate this overhead, we measure the time necessary to compile the Linux kernel, a task that involves many file operations. We considered two factors: type of storage (encrypted or plaintext), and loading of the UCLinux module (`uclinux.ko`) in the kernel (loaded or unloaded). For encrypted storage, we used AES in CBC mode with the ESSIV IV generation scheme included in Linux³. For tests with `uclinux.ko` loaded, we configured each kernel source file with a usage policy that permits the file's use only with a trusted compiler. We configured the kernel source with 'make `allnoconfig`'.

For each combination of factors, we performed the following steps:

1. Create a virtual hard disk:

```
dd if=/dev/zero of={filename}.0 bs=5M count=80,
```

where `{filename}` is a description of the factors used (plain/aes and nouclinux/uclinux) for that test. This creates a 400 MB file we will use as storage for a loop-back file-system.
2. Create the loop-back device:

```
losetup /dev/loop0 {filename}.0
```
3. Create the cryptographic devices, if necessary:

```
cryptsetup -c aes-cbc-essiv:sha256 create aes0 /dev/loop0
```
4. Mount the virtual disk:

```
mount /dev/loop0 mnt (if encryption is not in use)
```

or

```
mount /dev/mapper/aes0 mnt (if it is).
```
5. Copy the Linux source (already configured and ready to compile) to the mount point.
6. If necessary, add usage policy to the source files.
7. Unmount `mnt`.

³ESSIV generates each sector's IV by encrypting the digest (we use SHA-256) of that sector's number with the disk's key. This protects against watermarking attacks, but CBC remains vulnerable to other attacks against file integrity. Once a better disk cipher block mode is standardized, we would recommend switching to it instead.

8. For each replication, where $\{n\}$ is the replication number (1 to 3):
 - (a) Copy the replications:
`cp {filename}.0 {filename} .{n}`
 - (b) Create a loop back device:
`losetup /dev/loop{n} {filename} .{n}`
for each replication.
 - (c) Create the cryptographic devices, if necessary:
`cryptsetup -c aes-cbc-essiv:sha256 create aes{num} /dev/loop{n}`
 - (d) Mount each virtual disk for each replication at `mnt{n}`
9. Run `cat /dev/{device} > /dev/null`, where `{device}` is the device of a partition not used in the test, to clear the operating-system-level cache. The disk cache is clear for the first replication after all replications are set up, because no disk cache is large enough to hold three copies of the kernel source simultaneously; the first copy would have been evicted by the time the third copy was finished writing.
10. Set up the UCLinux kernel module, if necessary.
11. Execute `sync;time (make;sync) in mnt{num}/linux` to measure the kernel compilation time. This clears all write-back cached data to the disk, then times the kernel compilation. We include a final cache clear in the timing to eliminate cache effects. Repeat from 8 for each replication.
12. Unmount each replication, and destroy their cryptographic translations and loop-backs to prepare for the next set of factors.

	No Encryption			AES-CBC-ESSIV:sha256		
	Real	User	System	Real	User	System
Without UCLinux	02:05.188	01:45.009	00:12.699	02:05.710	01:44.560	00:12.820
	02:02.909	01:44.843	00:12.864	02:03.570	01:44.744	00:12.873
	02:03.060	01:45.316	00:12.542	02:03.901	01:44.977	00:12.864
Std Dev	00:01.274	00:00.240	00:00.161	00:01.152	00:00.209	00:00.028
Mean	02:03.719	01:45.056	00:12.702	02:04.394	01:44.760	00:12.852
With UCLinux	02:16.390	01:42.302	00:20.557	02:17.322	01:42.268	00:20.457
	02:14.783	01:42.759	00:20.777	02:17.113	01:42.632	00:20.734
	02:20.074	01:42.653	00:20.740	02:15.881	01:42.720	00:20.833
Std Dev	00:02.713	00:00.239	00:00.118	00:00.779	00:00.240	00:00.195
Mean	02:17.082	01:42.571	00:20.691	02:16.772	01:42.540	00:20.675

Figure 2: Kernel compile time (in minutes)

Our results are given in Figure 2. Timings are broken down into “Real”, “User” and “System”. In our tests, the use of encrypted storage had negligible performance impact. The UCLinux module impacted most strongly “System” timings, which include only time spent by the compilation task in the kernel. The impact was roughly 61%. As expected, the UCLinux module had negligible effect on “User” timings. The latter include only time spent by the compilation task in use mode, which UCLinux does not modify. The

“Real” timings give the “wall-clock” time taken to compile the kernel, which should equal to the sum of the “System” and “User” times plus unaccounted overheads. The “Real” impact of UCLinux was roughly 10%.

6.2 TCB size

We generated a prototype TCB using the computer platform described in Section 6.1, using the methodology described in Section 5.7. We generated the minimal TCB needed to boot the system to boot without tripping by the time the user logs into GDM and opens a terminal. Our TCB had 419 entries.

7. CONCLUSION AND FUTURE WORK

We have described a modified Linux system, UCLinux, that enables TPM-based usage controls enforcement. It provides the attestation support, sealing support and protection from administrative abuse necessary for a trustworthy usage control system, and does so with existing hardware and limited changes to an existing and widely known operating system.

We are currently working on a modified version of UCLinux that allows files in the TCB list to be updated. It would also be interesting to investigate UCLinux’s use in systems with virtual-machine monitors and more recent CPUs with greater support for Trusted Computing.

8. REFERENCES

- [1] B. Balacheff, Liqun Chen, Siani Pearson, David Plaquin, and Graeme Prouder. *Trusted Computing Platforms: TCPA Technology In Context*. Prentice Hall PTR, July 2002.
- [2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.
- [3] Chris Wright, Crispin Cowan, Stephen Smalley, James Morris, and Greg Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. In *Proceedings of the 11th USENIX Security Symposium*, 5 September 2002.
- [4] Dave Touretzky. Adobe Remedies. [Online]. Available: <http://www.cs.cmu.edu/~dst/Adobe/Gallery/>, 6 May 2006.
- [5] John Marchesini, Sean W. Smith, Owen Wild, Josh Stabiner, and Alex Barsamian. Open-Source Applications of TCPA Hardware. In *Proceedings of 20th Annual Computer Security Applications Conference*, December 2004.
- [6] Masoom Alam, Xinwen Zhang, and Jean-Pierre Seifert. Trusted SECTET: A Model-Driven Framework for Trusted Computing based Systems. In *11th IEEE Enterprise Distributed Object Computing Conference*, 2007.
- [7] Microsoft. Next Generation Secure Computing Base. [Online]. Available: <http://www.microsoft.com/technet/security/news/ngscb.mspx>, July 2003.

- [8] Microsoft. BitLocker Drive Encryption: Executive Overview. [Online] Available: <http://technet.microsoft.com/en-us/windowsvista/aa906018.aspx>, 5 April 2006.
- [9] NSA. Security-Enhanced Linux. [Online] <http://www.nsa.gov/selinux/>, 27 August 2007.
- [10] Peter Djalaliev and José Brustoloni. Secure Web-Based Retrieval of Documents with Usage Controls. *submitted for publication*.
- [11] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcb-based integrity measurement architecture. In *Proceedings of the 13th Usenix Security Symposium*, August 2004.
- [12] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Rfc3546 - Transport Layer Security (TLS) Extensions. [Online] <http://www.ietf.org/rfc/rfc3546.txt>, June 2003.
- [13] Stefan Berger, Ramón Cáceres, Kenneth Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. vTPM: Virtualizing the Trusted Platform Module. In *15th USENIX Security Symposium*, July 2006.
- [14] T. Dierks and C. Allen. RFC 2246: The TLS Protocol: Version 1.0. [Online] <http://www.ietf.org/rfc/rfc2246.txt>, January 1999.
- [15] Trusted Computing Group. Homepage. [Online] Available: <https://www.trustedcomputinggroup.org>.
- [16] Trusted Computing Group. Trusted computing platform alliance (TCPA) main specification version 1.1b. [Online] Available from <http://www.trustedcomputinggroup.org>.
- [17] U.S. Department of Health and Human Services. Office for Civil Rights - HIPAA. [Online] <http://www.hhs.gov/ocr/hipaa/>, 29 July 2007.
- [18] Xinwen Zhang, Masoom Alam, Jean-Pierre Seifert, and Ruth Breu. Usage Control Platformization via Trustworthy SELinux. Technical report, Samsung Information Systems America, 2007.