Trusted Platform Module and Applications

Slide credits: Based on presentations from Dennis Kafura (TPM) and Bryan Parno (Flicker)
Trusted Platform Module

Integrity Measurement, Reporting, and Evaluation
Motivation

- **Reliance on remote clients/servers**
  - Financial records and e-commerce
  - Electronic medical records
  - Cloud computing

- **Threats to clients from remote servers**
  - Malicious servers masquerade as legitimate ones
  - Legitimate servers subject to attack
    - Malware
    - Viruses
    - Rootkits

- **Threats to servers from corrupted remote clients**
  - Penetrating firewalls
  - Release of confidential data
Motivation

- **Need**: mechanisms to verify the integrity of remote clients/servers
  - Correct patches installed
  - Advertised/expected services exist
  - System not compromised

- **Solution**
  - Provision of critical services by a trusted platform module (TPM) on the local host
  - Capability of host to measure integrity of host software
  - Protocol to communicate the integrity measurements from the host to a remote party
  - Means for remote party to assess the integrity measurements and determine level of trust in the host
Trusted Platform Module (TPM)

- Standard defined by the Trusted Computing Group

- Availability
  - Hardware chip ubiquitously available on most machines

- Core functionality
  - Secure storage
  - Platform integrity reporting
  - Platform authentication
TPM Architecture

keys, owner
authorization data

integrity measures

signing keys
when in use

external
interaction

TPM control

trusted Platform Module (TPM)

Tamper-Protected Packaging

Non-Volatile Storage
Platform Configuration Register (PCR)
Attestation Identity Key (AIK)
Program Code

I/O

Random Number Generator
SHA-1 Engine
Key Generation
RSA Engine
Opt-in
Exec Engine

Communications
TPM Architecture

- Non-Volatile Storage
- Platform Configuration Register (PCR)
- Attestation Identity Key (AIK)
- Program Code
- Random Number Generator
- SHA-1 Engine
- Key Generation
- RSA Engine
- Opt-In
- Exec Engine

Communications

I/O

- Trusted Platform Module (TPM)

Tamper-Protected Packaging

- symmetric keys, nonces
- encryption keys
- hashes
- encrypt/decrypt
- initialization
Execution Environment

- **Executable content**
  - Types
    - programs
    - libraries
    - scripts
  - Loaded by
    - kernel
    - application

- **Structured data**
  - class files
  - configuration files

- **Unstructured data**
  - databases
Pragmatics

- **Feasibility**
  - Manageable number of components to measure for typical systems
    - 500 for a workstation configured for general technical work (document authoring, programming, browsing, etc.)
    - 250 for a typical web server

- **Approach**
  - Extensible architecture
  - Provides essential measurement structures
  - Allows future additions
Trusted Building Blocks

- TBB do not have shielded locations or protected capabilities (as does TPM)
- CRTM: core root of trust for measurement
- Keyboard: showing physical presence when needed
Integrity Measurement

- Measure a component before executing it
- Record the measurement as a hash value of the code/data (aka, *fingerprint*)
- Produces a hash chain by combining individual hash values
- Changes in the executing code can be detected by comparing measurement of executing code against recorded value
- The measurements themselves must be protected from *undetected* manipulation
Detecting Malware Attacks

#000: D6DC07881A7EFD58EB8E9184CCA723AF4212D3DB boot_aggregate
#001: CD554B285123353BDA1794D9ABA48D69B2F74D73 linuxrc
#002: 9F860256709F1CD35037563DCDF798054F878705 nash
#003: 84ABD2960414CA4A448E0D2C9364B4E1725BDA4F init
#004: 194D956F288B36FB46E46A124E59D466DE7C73B6 ld-2.3.2.so
#005: 7DF33561E2A467A87CDD4BB8F68880517D3CAECB libc-2.3.2.so
...

#110: F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2 syslogd
...

initial

attack

Measurement before rootkit attack

#110: F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2 syslogd
...

#525: 4CA3918834E48694187F5A4DAB4EECD540AA8EA2 syslogd

Measurement after rootkit attack
Platform Configuration Registers

- At least 16 PCR registers, each register stores 20 bytes

New = SHA-1(current || update)

Zero on reboot, power cycle
Maintaining a Measurement List

- PCR contains the linked hash of all measurements in the list
- Alterations to the list values can be detected

New = SHA-1(current || update)
**Questions**

- How is the AIK generated?
- Where is it stored?
- How does the challenger validate the measurement list (ML)?

1. $C$: create non-predictable 160bit `nonce`
2. $C \rightarrow AS$: ChReq(`nonce`)
3a. $AS$: load protected $AIK_{priv}$ into TPM
3b. $AS$: retrieve $Quote = \text{sig}\{PCR, nonce\}_{AIK_{priv}}$
3c. $AS$: retrieve Measurement List $ML$
4. $AS \rightarrow C$: ChRes($Quote, ML$)
5a. $C$: determine trusted $\text{cert}(AIK_{pub})$
5b. $C$: validate $\text{sig}\{PCR, nonce\}_{AIK_{priv}}$
5c. $C$: validate `nonce` and $ML$ using $PCR$

*C*: challenger  
*AS*: attesting system  
*AIK*: attestation identity key
Long-term Keys

- The TPM has two long-term key pairs stored in non-volatile memory on the TPM
  - Endorsement Key (EK)
  - Storage Root Key (SRK)

- **Endorsement Key**
  - Private key never leaves the TPM
  - Limited use to minimize vulnerability
  - Identifies individual platform: potential privacy risk
  - Public part contained in endorsement credential
  - EK and endorsement credential loaded by manufacturer

- **Storage Root Key**
  - Basis for a key hierarchy that manages secure storage
  - More on this later...
Attestation Identity Keys (AIKs)

- **AIK**
  - serves as alias for EK
  - platform may have many AIKs to allow a number of unlinkable interactions
  - held in secure storage (see later)
  - guarantees that platform has a valid TPM (but does not identify platform)

- **Privacy CA**
  - must be trusted by platform and challenger
Sealed Storage

- Goal: ensure that information is accessible only when the system is in a known/acceptable state
- System state determined by PCR value
Assessing Integrity

- acceptable
- malicious
- vulnerable-remote
- vulnerable-local
- unknown/uncontrolled

\[
\text{client} \in \text{Distrusted} \leftarrow \exists e \in E(\text{client}) : \neg (e \in \text{Known}) \\
\quad \lor (e \in (\text{Malicious} \cup \text{Uncontrolled} \cup \text{Remote}))
\]

\[
\text{client} \in \text{IntHigh} \leftarrow \forall e \in E(\text{client}) : (e \in \text{Acceptable})
\]

\[
\text{client} \in \text{IntMedium} \leftarrow \neg (\text{client} \in \text{IntHigh}) \land \\
\quad \forall e \in E(\text{client}) : e \in (\text{Acceptable} \cup \text{Local})
\]
Adding Measurement Instrumentation

- Executables
- Libraries
- Kernel modules
- Load modules
- Measurement

Structured data:
- Bash shell
- Applications
- Sysfs

Unstructured data:
- Applications
Performance

<table>
<thead>
<tr>
<th>mmap type</th>
<th>mmap latency (stdev)</th>
<th>file_mmap LSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>no_SHA1</td>
<td>1.73 (\mu s) (0.0)</td>
<td>0.08 (\mu s)</td>
</tr>
<tr>
<td>SHA1</td>
<td>4.21 (\mu s) (0.0)</td>
<td>2.56 (\mu s)</td>
</tr>
<tr>
<td>SHA1+extend</td>
<td>5430 (\mu s) (1.3)</td>
<td>5430 (\mu s)</td>
</tr>
<tr>
<td>reference</td>
<td>1.65 (\mu s) (0.0)</td>
<td>n/a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Measurements via sysfs</th>
<th>Overhead (stdev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>measure</td>
<td></td>
</tr>
<tr>
<td>no_SHA1</td>
<td>4.32 (\mu s) (0.0)</td>
</tr>
<tr>
<td>SHA1</td>
<td>7.50 (\mu s) (0.0)</td>
</tr>
<tr>
<td>SHA1+extend</td>
<td>5430 (\mu s) (1.6)</td>
</tr>
<tr>
<td>reference</td>
<td></td>
</tr>
<tr>
<td>sys fs</td>
<td>4.32 (\mu s) (0.0)</td>
</tr>
<tr>
<td>open/write/close</td>
<td></td>
</tr>
</tbody>
</table>

- vast majority of cases does not require +extend
Performance

<table>
<thead>
<tr>
<th>File Size (Bytes)</th>
<th>Overhead (stdev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4.21 µs (0.0)</td>
</tr>
<tr>
<td>512</td>
<td>10.3 µs (0.0)</td>
</tr>
<tr>
<td>1K</td>
<td>16.3 µs (0.0)</td>
</tr>
<tr>
<td>16K</td>
<td>197 µs (0.1)</td>
</tr>
<tr>
<td>128K</td>
<td>1550 µs (1.1)</td>
</tr>
<tr>
<td>1M</td>
<td>12700 µs (16)</td>
</tr>
</tbody>
</table>

- increase in overhead for computing fingerprint
Flicker:
An Execution Infrastructure for TCB Minimization

Jonathan McCune, Bryan Parno, Adrian Perrig, Michael Reiter, and Hiroshi Isozaki
Trusted Computing Base (TCB)

DMA Devices (Network, Disk, USB, etc.)

CPU, RAM
TPM, Chipset

App 1

... 

App S

App 1

... 

App S

OS

DMA Devices (Network, Disk, USB, etc.)

CPU, RAM
TPM, Chipset

OS

Shim
Flicker’s Properties

- Isolate security-sensitive code execution from all other code and devices
- Attest to security-sensitive code and its arguments and nothing else
- Convince a remote party that security-sensitive code was protected
- Add < 250 LoC to the software TCB
Outline

- Introduction
- Background
  - Trusted Platform Module (TPM)
  - Late Launch
- Flicker Architecture and Extensions
- Flicker Applications
- Performance Evaluation
- Related Work and Conclusions
TPM Background

- The Trusted Platform Module (TPM) is a dedicated security chip
- Can provide an *attestation* to remote parties
  - Platform Configuration Registers (PCRs) summarize the computer’s software state
    - \( \text{PCR}_\text{Extend}(N, V): \text{PCR}_N = \text{SHA-1}(	ext{PCR}_N | V) \)
  - TPM provides a signature over PCR values
  - A subset of *dynamic* PCRs can be reset without a reboot
Late Launch Background

- **Supported by new commodity CPUs**
  - SVM for AMD
  - TXT (formerly LaGrande) for Intel

- **Designed to launch a VMM without a reboot**
  - Hardware-based protections ensure launch integrity

- **New CPU instruction (SKINIT/SENTER) accepts a memory region as input and atomically:**
  - Resets dynamic PCRs
  - Disables interrupts
  - Extends a measurement of the region into PCR 17
  - Begins executing at the start of the memory region
How is Flicker Different? Goals

1. Isolation

2. Provable protection

3. Meaningful Attestation

4. Minimal TCB

- TXT/SVM mean only first two goals because the TCB size (VMM) is huge...
Architecture Overview

■ Core technique
  ▪ Pause current execution environment (untrusted OS)
  ▪ Execute security-sensitive code with hardware-enforced isolation
  ▪ Resume previous execution

■ Extensions
  ▪ Attest only to code execution and protection
  ▪ Preserve state securely across invocations
  ▪ Establish secure communication with remote parties
Execution Flow

- **App**
- **OS**
- **Module**
- **SKINIT**
- **Reset**
- **CPU**
- **RAM**
- **TPM**
  - **PCRs:** STOP STOP 00000
  - **K1**

**Shim**

**STOP**
Attestation

PCRs: 0 0 0

Inputs

Outputs

STOP

STOP

Shim

TPM

K_{-1}

PCRs: STOP STOP 0

TPM

K_{-1}
Attestation

PCRs:

What code are you running?

0

Inputs

Outputs

... 

Sign

0

(OS, K-1)

App 1

... 

App S
Context Switch with Sealed Storage

- Seal data under combination of code, inputs, outputs
- Data unavailable to other code

![Diagram of context switch with sealed storage]

- Seal data under combination of code, inputs, outputs
- Data unavailable to other code
Outline

- Introduction
- Background
- Flicker Architecture and Extensions
- Flicker Applications
  - Developer’s Perspective
  - Example Applications
- Performance Evaluation
- Related Work and Conclusions
Developing With Flicker

- Sensitive code linked against the Flicker library
- Customized linker script lays out binary
- Application interacts with Flicker via a Flicker kernel module

```c
#include "flicker.h"
const char* msg = "Hello, world";
void flicker_main(void *inputs) {
    for(int i=0;i<13;++i)
        OUTPUT[i] = msg[i];
}

Made available at: /proc/flicker/output"
Default Functionality

- Shim can execute arbitrary x86 code but provides very limited functionality.
- Fortunately, many security-sensitive functions do not require much.
  - E.g., key generation, encryption/decryption, FFT.
- Functionality can be added to support a particular security-sensitive operation.
- We have partially automated the extraction of support code for security-sensitive code.
Existing Flicker Modules

- OS Protection: Memory protection, ring 3 execution
- Crypto: Crypto ops (RSA, SHA-1, etc.)
- Memory Alloc.: Malloc/free/realloc
- Secure Channel: Secure remote communication
- TPM Driver: Communicate with TPM
- TPM Utilities: Perform TPM ops
Outline

- Introduction
- Background
- Flicker Architecture and Extensions
- Flicker Applications
  - Developer’s Perspective
  - Example Applications
- Performance Evaluation
- Related Work and Conclusions
Application: Rootkit Detector

- Administrator can check the integrity of remote hosts
  - E.g., only allow uncompromised laptops to connect to the corporate VPN
Application: SSH Passwords

Start

Gen \{K, K^{-1}\}

\begin{align*}
K & \quad \text{Shim} \\
K^{-1} & \quad \text{Shim}
\end{align*}

Encrypt_{K}(\text{passwd})

OK!

Encrypt_{K^{-1}}(\text{passwd})
Other Applications Implemented

- Enhanced Certificate Authority (CA)
  - Private signing key isolated from entire system

- Verifiable distributed computing
  - Verifiably perform a computational task on a remote computer
  - Ex: SETI@Home, Folding@Home, distcc
Outline

- Introduction
- Background
- Flicker Architecture and Extensions
- Flicker Applications
- Performance Evaluation
- Related Work and Conclusions
## Generic Context-Switch Overhead

Each Flicker context switch requires:
- SKINIT
- TPM-based protection of application state

### Results

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKINIT</td>
<td>14 ms</td>
</tr>
<tr>
<td>Unseal application state</td>
<td>905 ms</td>
</tr>
<tr>
<td>Reseal application state</td>
<td>20 ms</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>939 ms</strong></td>
</tr>
</tbody>
</table>
### Rootkit Detection Performance

<table>
<thead>
<tr>
<th>Task</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKINIT</td>
<td>14 ms</td>
</tr>
<tr>
<td>Hash of Kernel</td>
<td>22 ms</td>
</tr>
<tr>
<td>PCR Extend Result</td>
<td>1 ms</td>
</tr>
<tr>
<td>TPM Quote</td>
<td>973 ms</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1023 ms</strong></td>
</tr>
</tbody>
</table>

- **37 ms Disruption**
- **Non-Disruptive**

Running detector every 30 seconds has negligible impact on system throughput.
SSH Performance

- Setup time (217 ms) dominated by key generation (185 ms)
- Password verification (937 ms) dominated by TPM Unseal (905 ms)

Adds < 2 seconds of delay to client login
Optimizing Flicker’s Performance

- Non-volatile storage
  - Access control based on PCRs
  - Read in 20ms, Write in 200 ms
  - Store a symmetric key for “sealing” and “unsealing” state

Reduces context-switch overhead by an order of magnitude
Hardware Performance Improvements

[ASPLOS 2008]

- Late launch cost only incurred when Flicker session launches
- TPM (Un)Seal only used for long-term storage
- Multicore systems remain interactive
- Context switch overheads (common case) resemble VM switches today (~0.5 μs)
Related Work

- Secure coprocessors
  - Dyad [Yee 1994], IBM 4758 [JiSmiMi 2001]
- System-wide attestation
  - Secure Boot [ArFaSm 1997], IMA [SaZhJaDo 2004], Enforcer [MaSmWiStBa 2004]
- VMM-based isolation
  - BIND [ShPeDo2005], AppCores [SiPuHaHe 2006], Proxos [TaLiLi 2006]
- “Traditional” uses of late launch
  - Trustworthy Kiosks [GaCáBeSaDoZh 2006], OSLO [Kauer 2007]
Conclusions

- Flicker greatly reduces an application’s TCB
- Isolate security-sensitive code execution
- Provide fine-grained attestations
- Allow application writers to focus on the security of their own code