Adversary Model

- **Network**
  - Attacker has complete control over the network (read, intercept, inject messages)

- **System**
  - Attacker can modify software stack on disk, reset machine
  - Attacker cannot break hardware protections

- **Cryptography**
  - Attacker cannot break cryptography (hash function is \textit{collision resistant}; signatures are \textit{unforgeable})
SRTM Property

Temporal property: software stack was loaded sometime in the past

No Resets during $T_R$ to $T_{Read}$

Verification of Stale Data

Server Reset

Begin

Reset @ $T_R$

Call to OS Code @ $T_{os}$

Read Log @ $T_{Read}$

Verify Log @ $T_V$

End
Shortcomings of Static Root of Trust

- **Integrity measurements are done at load-time, not at run-time**
  - Time-of-check-time-of-use (TOCTOU) problem

- **Coarse-grained, measures entire system**
  - Every host is different: different firmware versions, different drivers, different patches, different apps, different spyware
  - What does a PCR mean in this context?
  - TCB includes entire system!

- **No guarantee of execution**
Authenticated Sessions

- **Approach:**
  - Combine authentication (code attestation) with key exchange to share secret key between client & server
  - Protect subsequent interaction using key (authorized actions)
  - Simple reset attack will not work any more

- **Related challenges in web authentication**
  - Secret might be acquired by adversary (reflected XSS, session hijacking)
  - User may unintentionally use secret on adversary’s behalf (CSRF)
Inside the TPM
Components on TPM chip

- Non Volatile Storage (> 1280 bytes)
- PCR Registers (≥16 registers)
- Other Junk
- I/O
- Crypto Engine: RSA, SHA1, HMAC, RNG

- RSA: 1024, 2048 bit modulus
- SHA1: Outputs 20 byte digest
Non-volatile storage

1. **Endorsement Key (EK)** (2048-bit RSA)
   - Created at manufacturing time, cannot be changed
   - Used for “attestation” (AIK used instead of EK for privacy)

2. **Storage Root Key (SRK)** (2048-bit RSA)
   - Used for encrypted storage
   - Created after running `TPM_TakeOwnership( OwnerPassword, ... )`
   - Can be cleared later with `TPM_ForceClear` from BIOS

3. **OwnerPassword** (160 bits) and persistent flags

Private: EK, SRK, and OwnerPwd never leave the TPM
PCR: The Heart of the Matter

**PCR**: Platform Configuration Registers
- Many PCR registers on chip (at least 16)
- Contents: 20-byte SHA1 digest

**Accessing PCR #n**:
- **TPM_Extend(n,D)**: \( PCR[n] \leftarrow SHA1(PCR[n] \ || \ D) \)
- **TPM_PcrRead(n)**: returns value(PCR(n))

PCR\(\text{s}\) init. to default value (e.g. 0) at boot time
Using PCRs: The TCG Boot Process

On power-up: TPM receives a **TPM_Init** signal from LPC bus

**BIOS boot block** executes:

- Calls **TPM_Startup (ST_CLEAR)** to initialize PCRs to 0
  [can only be called once after TPM_Init]
- Calls **PCR_Extend(n, <BIOS code>)**
- Then loads and runs BIOS post boot code

**BIOS executes:** Calls **PCR_Extend(n, <MBR code>)**
- Then runs MBR (master boot record), e.g. GRUB.

**MBR executes:** Calls **PCR_Extend(n, <OS loader code, config>)**
- Then runs OS loader
  · · · and so on
The Main Point

After boot completes, PCR registers measure the entire software stack that booted on the machine:

- BIOS and hardware configuration
- Boot loader and its configuration
- Operating system
- Running apps
Sealed Storage
Using PCR values after boot

- **Step 1:** `TPM_TakeOwnership( OwnerPassword, …)`
  - Creates 2048-bit RSA Storage Root Key (SRK) on TPM
  - Cannot run `TPM_TakeOwnership` again without `OwnerPwd`
  - Done once by IT department or laptop owner

- **(optional) Step 2:** `TPM_CreateWrapKey / TPM_LoadKey`
  - Create more RSA keys on TPM protected by SRK
  - Each key identified by 32-bit keyhandle
Protected Storage

- **Main Step:** Encrypt data using RSA key on TPM
  - `TPM_Sea`l Returns encrypted blob; arguments:
    - `keyhandle`: which TPM key to encrypt with
    - `KeyAuth`: Password for using key `keyhandle`
    - `PcrValues`: PCRs to embed in encrypted blob
    - `data block`: at most 256 bytes (2048 bits)

- **Main point:**
  - Blob can only be decrypted with `TPM_Unseal` when PCR-reg-vals = PCR-vals in blob
  - *Embedding PCR values in blob ensures that only certain apps can decrypt data*
Sealed storage: applications

- **Lock software on machine:**
  - OS and apps sealed with Master Boot Record’s PCR
  - Any changes to MBR (to load other OS) will prevent locked software from loading
  - Prevents tampering and reverse engineering
    - e.g., software integrity on voting terminals

- **Microsoft Bitlocker**
  - Disk encryption: only unmodified OS kernel can decrypt disk

- **Web server: seal server’s SSL private key**
  - Goal: only unmodified Apache can access SSL key
Question

- What property of encryption scheme is essential to guarantee that other apps cannot decrypt sealed blob?

- IND-CCA2 security
  - TPM_Seal is the encryption oracle
  - TPM_Unseal is the decryption oracle
  - Other apps cannot get TPM_Unseal to directly decrypt the sealed blob because PCR values won’t match
IND-CCA secure encryption
(chosen-ciphertext attacks)

IND-CCA security: \forall \text{PPT attackers } A \\exists \text{ negligible function } f \\exists n_0 \\forall \text{ security parameters } n \geq n_0 \text{ Prob } [d = b \mid A \text{ plays by the rules}] \leq \frac{1}{2} + f(n)
Dynamic Root of Trust
Shortcomings of Static Root of Trust

- Integrity measurements are done at load-time not at run-time
  - Time-of-check-time-of-use (TOCTOU) problem
- Coarse-grained, measures entire system
  - Every host is different: different firmware versions, different drivers, different patches, different apps, different spyware
  - What does a PCR mean in this context?
  - TCB includes entire system!
Dynamic Root of Trust

- Leverages security primitive provided by latest CPUs and TPMs (AMD SVM, Intel TXT)

- Late launch
  - Supports isolated code execution
  - Example: AMD’s SKINIT instruction
SKINIT (Secure Kernel Init)

- **Accepts address of Secure Loader Block (SLB)**
  - Memory region up to 64 KB

- **SKINIT executes atomically**
  - Enables DMA protection for entire 64 KB SLB
  - Disables interrupts to prevent other code from gaining control
  - Debugging access is disabled (even for h/w debuggers)
  - Causes TPM to reset PCRs 17-23 and hash SLB contents and extend PCR 17
  - Begins executing at SLB’s entry point
Basic Flicker Architecture

1. Pause current execution environment (legacy OS)
2. Execute security-sensitive code in isolation using late launch
3. Preserve session-state with TPM sealed storage
4. Resume previous environment
What Is S?

- Self-contained code in an application
- Data secrecy and integrity requirements
- General-purpose computing
- Some examples
  - Manages a private key for web server or CA
  - Manages Access Control List (ACL)
  - Is a compute client in distributed setting
TCB Reduction with Flicker

Today, TCB for sensitive code S:
- Includes App
- Includes OS
- Includes other Apps
- Includes hardware

With Flicker, S’s TCB:
- Includes Shim
- Includes some hardware

Adversary controls everything else
Properties of the Flicker System

- Isolate security-sensitive code execution from all other code and devices
- Attest to security-sensitive code and its arguments and nothing else
- Attest to a remote party that security-sensitive code was executed in isolation
- Add < 250 LoC to the software TCB
Recall: Shortcomings of Static Root of Trust

- Integrity measurements are done at load-time not at run-time
  - Time-of-check-time-of-use (TOCTOU) problem
- Coarse-grained, measures entire system
  - Every host is different: different firmware versions, different drivers, different patches, different apps, different spyware
  - What does a PCR mean in this context?
  - TCB includes entire system!
External Verification: Attestation

- Trust in attestation rooted in hardware TPM (Trusted Platform Module)
- Attestation covers code that \textit{executed} and its \textit{inputs and outputs}
Flicker Context Switch with Sealed Storage

- **Seal** data under combination of code, inputs
- Data unavailable to other code
Limitations of Flicker

- Overhead of late launch is high for certain applications
  - Example: web server application S that performs sensitive signing operations and protects the signing key
  - TrustVisor: An alternative architecture that addresses this issue (optional reading)

- Only works for self-contained applications
Summary: Trusted Computing

- Leverage Trusted Platform Module in commodity computers to achieve security properties
  - crypto + hardware + software

- Properties (understand caveats):
  - Attestation:
    Prove to remote server what expected software was loaded/executed on machine
  - Hardware protected (encrypted) storage:
    Only “authorized” software can decrypt data
What Is It Good For?

- **Verified Boot in Chromium OS**
  http://www.chromium.org/chromium-os/chromiumos-design-docs/verified-boot
  Useful discussion of attack scenarios and associated security guarantees

- **BitLocker Drive Encryption Overview**
  A compelling application
Trusted vs Trustworthy

- A **trusted** system or component is one whose failure can break the security policy
- A **trustworthy** system or component is one that will not fail

- Trusted Computing as used in the title of this lecture is really intended to mean “Towards Trustworthy Computing”
From “Trusted” to “Trustworthy”

- Trusted computing systems are useful to convince a third party that a “trusted” program P is running on a remote system.

- We will discuss methods for establishing that P is “trustworthy” in the next two modules of the course (some discussion already completed on run-time defenses for isolation and control flow integrity).
  - Model checking, static & dynamic analysis, type systems
  - Easier to apply if “P” is small and simple (i.e., smaller and simpler TCB is better)
Acknowledgement

- Many of the slides used in this lecture are based on slides provided by Jon McCune and on slides from Dan Boneh and John Mitchell’s CS155 computer security course at Stanford
Additional Details about Late Launch
A cloud application

Client seals VM to VMM measurement:
- VM code and data are encrypted
- Can only be decrypted on valid cloud server
- Cloud operator cannot easily access data
AMD Secure Virtual Machine

- Two categories of hardware support
  - Virtualization support
    - External DMA access protection for memory
    - Intercept selected guest instructions / events
    - Much more...
  - Late launch with support for attestation
    - New instruction: SKINIT (Secure Kernel Init)
    - Requires appropriate platform support (e.g., TPM 1.2)
    - Allows verifiable startup of trusted software
      - Such as a VMM
      - Based on hash comparison
SKINIT TPM Operations

- **TPM v1.2 includes notion called** locality
  - Similar to software privilege level
  - 4 is highest, 0 is lowest
  - Certain PCRs associated with localities
  - PCR 17 is associated with locality 4
  - SKINIT is the only locality 4 operation

- **SKINIT sends contents of SLB to TPM**
  - TPM hashes SLB to create a measurement
  - TPM resets PCR17, sets PCR17 = 0
    - Distinct from boot-time value of PCR17= -1
    - Allows verifier to know that SKINIT was executed
  - TPM performs PCR_Extend(17, hash(SLB))
Secure Loader Block Layout

- ESP
- EAX
- SL Stack
- SL Code and Static Data
- SL Header
- Length
- EP Offset
- SL Runtime Data Area
- SL Image (hash area)
SKINIT Security Properties

- **Verifier receives attestation after SKINIT**
  - Knows SKINIT was used
  - Knows software TCB includes *only* the SLB
  - Knows exactly what SLB was executed

- **SLB can be written to provide add’l props.**
  - Knows any inputs to SLB
  - Knows any outputs from SLB
  - Knows exactly when SLB finished executing
AMD SVM Security Discussion

- **Property**: Verifiable untampered code execution
- **SKINIT + TCG 1.2** provide very strong security properties
- **Minimal TCB**: Only hardware and application need to be trusted
  - “Late launch” or “dynamic root of trust” remove trusting the operating system, Bios, and even DMA-capable devices