System Security

February 12, 2020

Imprint

Authors

This summary was written by

- Theo von Arx.
- Tom Cinbis.

It is based on

- the lecture slides *System Security* (Fall 2019, ETH Zurich) by Srdjan Čapkun, Adrian Perrig, and Kari Kostiainen;
- various papers.

Source Code

The LaTeXsouce code can be found here.

License

1 Basics

1.1 Concepts

Integrity: The data was not changed by a party unauthorized to change it.

Confidentiality: An unauthorized person cannot understand the contents of the message, because the message looks like random bits (protected data belongs to some other users).

Secrecy: Same as confidentiality, but protected data belongs to the sender.

DoS: A service is blocked either by sending enough traffic to overwhelm the service or selectively exploit bugs/errors in the design.

Authentication: The identity of the sender is correct.

Authorization: Examine whether a person has the capability to use a certain service or access certain data.

	Confidentiality	Authentification
Symmetric cryptography	symmetric key encryption	symmetric key authentication
Asymmetric cryptography	public key encryption	digital signatures

Proper Usage

Authenticate-then-encrypt: provides both secrecy and authenticity of the message.

- susceptible to chosen ciphertext attacks, because an attacker can simply send messages and observe the system's behavior upon decryption attempts.
- can potentially be used for a DoS attack

Ecrypt-then-authenticate: provides both secrecy and authenticity of the message.

- crafted messaged are detected immediately
- faster

Encrypt-and-authenticate The MAC output may reveal information on the plaintext and thus insecure.

1.2 RSA

1. Choose primes p, q and let $n = p \cdot q$
2. Let $\phi(n) = (p-1)(q-1)$
3. Choose e relatively prime to $\phi(n)$: $\gcd(\phi(n), e) = 1$
4. Compute d, s.t. $ed \bmod \phi(n) = 1$
5. The public key is: $\text{PU} = (e, n)$
6. The private key is: $\text{PK} = (d, n)$
7. Encrypt with $c = m^e \bmod n$
8. Decrypt with $m = c^d \bmod n$

Intuition: If one can factor n, one can break RSA.

1.3 Block Cipher

Secret key = (map cleartext ⇔ ciphertext)

- Modern block ciphers use a key of K bits to specify a random subset of 2^K mappings.
- If the selection of the 2^K mappings is random, the resulting cipher will be a good approximation of the ideal block cipher.
- **Diffusion:** If plaintext bit i is changed, the ciphertext bit j should change with $p = 0.5$
- **Confusion:** Each bit of c should depend on the whole key

1.4 Hash function

Properties of a hash function: $y = H(x)$

one-way can not be reversed

weak collision resistance given x, can not find $x' \neq x$ s.t. $H(x) = H(x')$

strong collision resistance can not find $x' \neq x$ s.t. $H(x) = H(x')$

2 OWASP

Injection

Occur when untrusted data is sent to an interpreter as part of a command or query.

Broken Authentication

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.

Sensitive Data Exposure

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data.

XML External Entities

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and DoS attacks.

Broken Access Control

Restrictions on what authenticated users are allowed to do are often not properly enforced.

Security Misconfiguration

een issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information.

XSS

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Insecure Deserialization

Insecure deserialization often leads to remote code execution.

Components with Vulnerabilities

Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover.

Insufficient Logging & Monitoring

Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data.

3 Side Channel Attacks

3.1 Introduction

Security proofs are based on models of system and attacker. Most models do not take into account the implementation and how it interacts with the environment.

Examples:

- Faulty Output (Glitch)
- Power Consumption
- EM Emissions
- Heat
- Timing
- Design Details

- Sound
- Data Coupling

Two categories of Side channel analysis:

Simple computation only depends upon the key.

Differential computation depends upon both the input and the key.

3.2 Timing Cryptanalysis of RSA

1. Execution time depends on the key
2. Can be measured for different inputs

Square-And-Multiply

```
// Computes C = m^d mod n
// MSB first
Square-And-Multiply(m, d, n):
  w = bitlength(d)
  x = m
  for i = 1 to w-1:
    x = x*x  mod n
     if (bit i of d) == 1:
       x = x*m mod n
  return x
```

- Key-dependent Branching
- Attacker observes how many 1 are in the key

Greatly reduces the search space for the secret key, but still huge...

Montgomery multiplication

Multiplication the execution time depends on **input and key**.

Attack main idea:

- Process attack bit per bit.
- Try large number of different messages and leverage average execution time.
- For each message, test whether the round bit was 1 or 0.

Protecting

- Change implementation s.t. it is not time dependent.
- Generic protection is hard.
- Performance Penalty

3.3 Cache Attacks

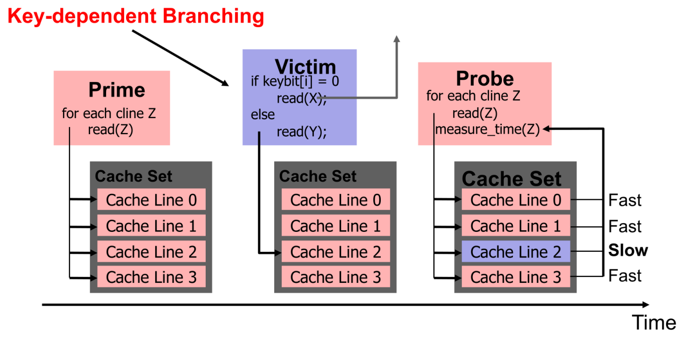
Flush+Reload

For shared memory only.

1. Attacker flushes cache
2. Victim caches data
3. Attacker accesses possible cached data. If this access was fast, then data is cached.

Prime+Probe

Also without shared memory (**NOTE:** we only get the cache address which was loaded by victim)



Variants

1. **Data access** depends on secret data: Data access patterns in the cache leak information about the secret
2. **Control flow** depends on secret data: Code access patterns in the cache leak information about the secret

Cache Timing Attack on AES

Background

During the execution of AES secret key is used to index arrays (S-boxes). The time to lookup an array element depends on whether this element of the S-box has partially or entirely been loaded into the cache.

- Allows complete key recovery.
- Problem is in the design, not in the implementation.

Setting:

- Attacker can send messages and observe overall execution time
- Attacker has same setup (software, hardware) for local tests Execution

Attack:

- For each byte of the key, one index value will have the slowest lookup.
- The total execution will be slow
- Find this index value

$\Rightarrow$ try many messages locally

Because we guess every byte of the key, we need $k * 2^8$ messages in theory, where k is the number of bytes in the key.

Protecting

- **Basic principle:** Try to eliminate secret-dependent cache access patterns
- Specific defenses exist, but eliminating all secret-dependent branching (code and data) is difficult and expensive...
- Most CPUs have special AES hardware $\Rightarrow$ not vulnerable.

3.4 Power Analysis Attacks

Mainly used on Smartcards, RFID chips, Sensor Nodes

- The attacker needs to have physical access
- Measures the consumed power during operation

Square-and-multy in RSA is vulnerable through simple power analysis: Attacker can get entire key.

Protection

Goal: Elimination or significant reduction of the correlation between operand values and power consumption.

- Random change of power consumption in time
- Noise generator
- Physical shielding
- Software balancing
- Hardware balancing

3.5 Acoustic Attacks

- High-frequency sounds caused by vibration of electronic components
- **Different keys cause different sounds**
- Can extract RSA keys based on sound

3.6 Electromagnetic Attacks

TEMPEST: Transmitted Electro-Magnetic Pulse / Energy Standards & Testing

- Compromising emanations may be generated by any electrical information generating or processing equipment.
- Can be sensed and transmitted over air, water, electrical lines, ...
- Typical examples include:
  - Displays
  - Keyboards
  - Cables
  - Processors

4 Tamper Resilience

4.1 Classification

- Tamper resistant
  - Bank vault approach
  - Purpose: Prevent/slow down of break-in
- Tamper responding
  - Alarm approach
  - Erasure or destruction of secret data
  - Purpose: real-time-detection
- Tamper evident
  - If a break-in occurs, evidence of the break-in is left behind.
  - Purpose: detection of intrusion

4.2 Smartcards

- Limited tamper resistance
- PIN + card possession enable user authentication
- Card holds a key

Memory Read-out Attack

Attacker initiates the protocol (the same protocol that the legitimate reader would), and when the CPU accesses memory to get the key, it reads the key from the memory bus.

Protections:

- read critical keys from memory only after authenticating that they are talking to ATM.

4.3 Hardware Security Module

- Tamper resistance
- Limited set of functions
- No support for exporting the key

4.4 Glitch Attacks

In a glitch attack, attacker deliberately generate a malfunction that causes one or more flipflops to adopt the wrong state.
⇒ Replace a single critical machine instruction with arbitrary one.

Example:

```
1  a = answer_address
2  b = answer_length
3  if (b == 0) goto 8
4  transmit(*a)
5  a = a + 1
6  ~~b = b - 1~~ % prevented by glitch
7  goto 3
8  ...
```

Clock-signal glitches are currently the most used. They temporarily increase the clock frequency, such that some flipflops sample their input before the new state has reached them.

Protection

- Low and high voltage sensors
- Frequency sensors and filters
- Light sensor
- Glitch sensor
- Software countermeasures

4.5 API Attacks

Background

A security API is the software layer through which the module's functions are exposed to the external world.

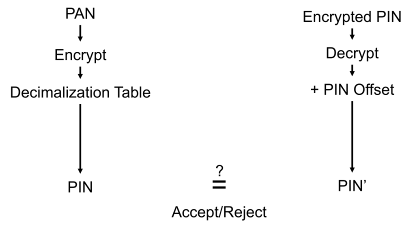
- The HSM is secure
  - Protects Contents (Keys, Secrets, PINs)
  - Can't break in (Tamper-Evident)
- Trick the HSM into leaking the secrets by sending the right requests!
- Most tamper resistant devices are vulnerable to API attacks

PIN Verification at a Bank

Inputs:

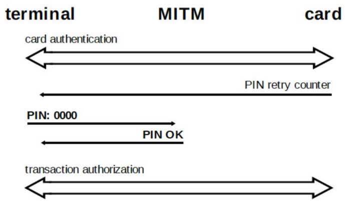
- Encrypted PIN, Account Number (PAN)
- Decimalization Table, PIN Offset

Attack Idea: Supply different decimalization table and PIN offset



Chip and PIN

Man in the middle attack:



Before the transaction, the MITM tells card that transaction will be signed on paper ⇒ No PIN required.

4.6 Protection

- Access Control
- Limit functionality
  - E.g. Do not allow other decimalization tables

5 Security on Commodity Systems

5.1 Application Security

Properties

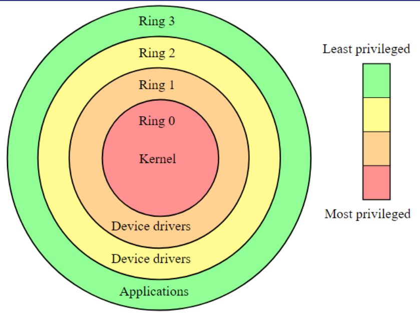
Launch-time integrity: *correct* application was started

Run-time isolation: no interference from malicious software or peripherals

Secure persistent storage Confidentiality, integrity protection of data

5.2 OS-based Security

Privilege Rings



- Today, only rings 0 and 3 are used; 1 - 2 part of kernel
- Main use: limiting access to privileged instructions, I/O-ports

Paging-based Security

Security-relevant data in page table entries:

Supervisor bit: if set, this page is accessible only in ring 0 (isolates OS from applications)

RW bits: to distinguish between read-only and writeable pages

Execution disable (ED) bit: if set, the page is not executable (prevents run-time code injection)

Firewire (DMA) Attack

Background: Firewire allows for fast communication speeds between devices, uses DMA (Direct Memory Access)

Idea: Access to RAM is controlled by CPU. This can be circumvented by DMA.

Attack:

- The attacker uses a Firewire cable to connect to a (locked) PC and issue a DMA request to fetch the contents of RAM
- Later on it can look in the collected data and find out keys and other passwords

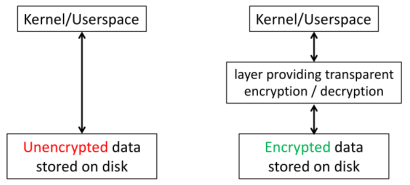
Protection:

- Have no Firewire ports, but there are other ports for similar attacks.
- OS sets up a IOMMU (Input–output memory management unit) to control DMA access to physical memory.

Disk Encryption

Idea: Disk is encrypted and unlocked by a secret key.

- Data on disk is allways encrypted.
- File system is completely unaware:


Kernel/Userspace
Unencrypted data stored on disk
Kernel/Userspace
layer providing transparent encryption / decryption
Encrypted data stored on disk

Simple Approach: Use password only ⇒ can be bruteforced

Better Approach: Store disk encryption key in a secure element (TPM) ⇒ requires more hardware support.

Cold Boot Attack on Disk Encryption

Idea:

- encryption key is kept in memory
- DRAM keeps it's state for a while without refresh, and longer if it is cold.

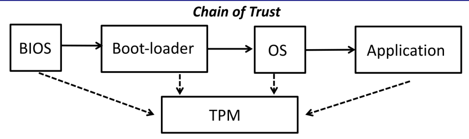
Attack:

- Remove power
- Cool down RAM
- Plug RAM to another platform / Boot with a USB and attack tools to copy memory content
- Recover key (part in memory with high entropy)

Protection:

- Shutdown device entirely
- Erase key from memory on every suspend
  - does not help sudden power loss
- Physical protection
- Avoid to have key in memory

Chain of Trust



Secure Boot: OS boots only if the chain of trust is valid

Authenticated boot: System records chain of trust but OS boots even if the chain is invalid

5.3 Storage protection on smartphones

Background

- Encrypt with user-provided PIN
- no TPM available
- Approach:
  1. At boot the OS asks PIN code from user
  2. PIN is given to the CPU
  3. CPU derives storage encryption key from PIN and processor key

  ⇒ Prevents brute-forcing of extracted storage
- storage must be decrypted on the same device where the CPU is
- A counter indicates how many attempts are left.

NAND Mirroring Attack

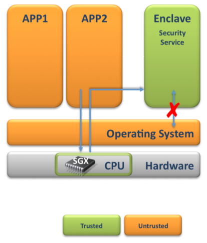
1. Eavesdropping:
   Find the communication protocols and the communication speeds
2. Backing Up:
   Create a copy of the NAND chip
3. Restoring:
   - Power on phone
   - Enter 6 PINs
   - Power down
   - Take out NAND and restore modified parts from backup
   - Put NAND back into phone
   - Repeat
4. OR Cloning:
   - Arbitrarily many clones can be created
   - Always plug in a new cloned chip

Allows brute-forcing of a 4-digit PIN in less than a day.

6 Intel SGX (Software Guard Extension)

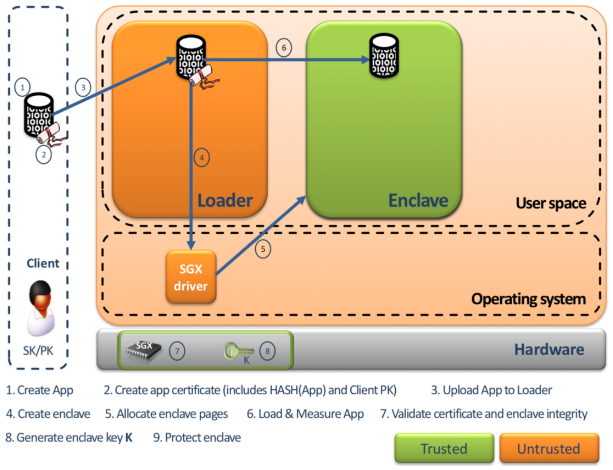
6.1 Basics

- Security enhancements in Intel CPUs
- Enclave = security-critical part of the application
- Main goal:
  - Enable secure execution in otherwise untrusted platforms (OS, ...)
  - Enclave data confidentiality and execution integrity
- Idea: Reduce TCB and trust boundary.



- Trusted are:
  - Intel
  - The CPU
  - Quoting Enclave
  - Intel SGX Trusted Libraries
- Not trusted:
  - BIOS, Firmware
  - OS
  - Other software running on the machine
  - Other hardware (e.g. memory, motherboard)
- Unique set of keys are burned into the processor.
- Parallel Execution of different Enclaves possible.
- Enclave code can not contain secrets (known to OS). Use attestation and secure channel establishment to provision secrets.

6.2 Enclave Creation



6.3 Sealing

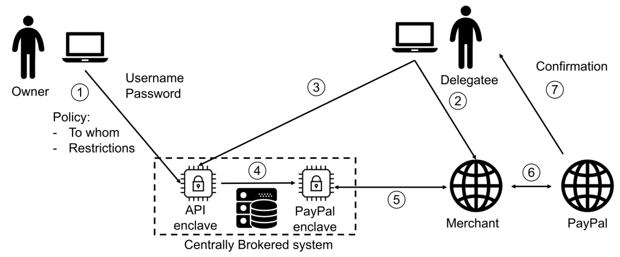
- Store encrypted confidential data on disk. Only Enclave can access it.
- Enclave has no direct access to IO / storage.
- Same Enclave ID on the same platform can unseal it's sealed data.
- Enclave will loose state across reboots, but can unseal it's sealed state.

6.4 Attestation and secure communication

- Two enclaves can communicate securely over an untrusted channel.
- Remote Attestation allows Enclave to check whether other instance is a valid and secure SGX Enclave:
  1. Verifier sends nonce.
  2. Enclave generates report, which contains Diffi-Hellman contribution.
  3. Pass report to quoting enclave which verifies the report.
  4. Signs report with the platform key and send to the verifier.

6.5 Application: DelegaTEE

- Owner's credentials remain confidential.
- The Owner can restrict access to his account (in time, actions, ...)
- Difficult to distinguish between access by the Delegatee and Owner
- Functioning:



- Applications: Email access, website access, e-banking, paypal
- May undermine service (e.g. Netflix) and create market for sharing economy
- Two systems possible: centrally brokered or decentralized peer-to-peer

6.6 Attacks

Cache-timing attack

- PMC
  - Use Performance Monitoring Counters (PMC) to count cache misses (instead of timing L1 vs. L2)
  - Enabled by privileged adversary
- isolate victim
  - Assign the attack process and victim enclave to separate core
  - Reduces noise
- hyper-threading
  - Run uninterrupted victim enclave and attacker in parallel
  - Complicates attack detection
- Attack target
  - Attack non-cryptographic computations
- Prime + Probe by other malicious enclave (can extract 96% of a RSA key).

Generic and efficient defenses are an open problem.

Page Fault Attack

Page faults allow the OS to observe input-dependent control transfer or input-dependent data access. Thereby the OS can infer the execution flow and can observe otherwise protected inputs.

Protection: Rewrite s.t. memory access pattern does not depend on input.

7 Meltdown-Spectre Attacks

7.1 Meltdown

Basics

Architecture	**Microarchitecture**
Abstract	Implemented
Defined in Instruction Set Architecture (ISA)	Follows ISA
E.g. x86, ARM	E.g. Intel core i7, AMD Ryzen
Registers, main memory	Cache, Reorder buffer, branch prediction

Cache location depends on actual data address.

Page tables translate virt. address to physical address

Page table entry (PTE) includes permission bits and who is allowed to access

Pipelining split instruction into

- instr. fetch (IF)
- instr. decode (ID)
- execute (EX)
- memory access (MEM)
- write back (WB)

Out-of-Order execution parallelize execute stage (EX). Run all instr. without inter dependencies in parallel. Retire in-order.

Attack

Idea: Microarchitectural state (caches) is modified even if memory access was not allowed.

```
1: mov eax, [kernel_address]

2: mov ebx, [probe_array + 4096*eax]
```

probe_array is the base point to a large array

1. Use a custom exception handler to page fault.
2. `probe_array` will be accessed at an index which depends on the secret (e.g. data in kernel address space [`kernel_address`]).
3. Check cache load times for all indices of array to get the index which was accessed by prohibited memory read.

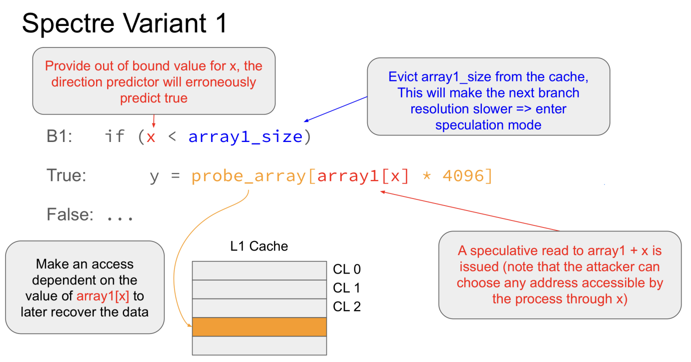
7.2 Spectre

- based on branch prediction and speculative execution
- dynamic predictors use runtime information
- Branch Target Buffer (BTB) does not store information about process id or virt. address it belongs to → same for each core → can be trained by process A but used for B

Spectre 1

Attacker has control over input used in branch (here `x`).

Can only read memory accessible for process, but not to the attacker (no privileged read).


Spectre Variant 1

```
B1:   if (x < array1_size)

True:     y = probe_array[array1[x] * 4096]

False: ...
```

Provide out of bound value for x, the direction predictor will erroneously predict true
Evict array1_size from the cache, This will make the next branch resolution slower => enter speculation mode
Make an access dependent on the value of array1[x] to later recover the data
L1 Cache
CL 0
CL 1
CL 2
A speculative read to array1 + x is issued (note that the attacker can choose any address accessible by the process through x)

Spectre 2

Idea: Mistrain BTB to exec. code which creates a side-channel → spectre gadget.
Unlike Meltdown this does not exploit race condition.

Gadget should:
1. exec memory access controlled by attacker's input
2. exec instr which creates change in microarchitectural state

Attacker needs:
- 1 input to select memory location to read
- 1 input to control how to leak (e.g. base pointer of `probe_array`)

8 OS Security

OS Security Goals secrecy, integrity, availability

Trust Model set of SW and data which the system depends on for correct enforcement of security goals

Threat Model set of operations available to compromise system

8.1 Access control

Whether to allow requests from **subjects** to perform **operations** on **objects**.

protection system defines security requirements of OS. Consists of protection state and protection state operations

protection state operations a subject can perform on obj.

protection state operations enable modification of state

An access matrix defines which operations a subject can perform on an obj. (subject as row, object as column) Store by column: ACL (**A**ccess **C**ontrol **L**ist) stored with objects. Store by row: Capability of a subject

Capability	ACL
- instant revocation is hard	+ instant revocation is possible, especially for single object
+ eventual revocation is easy (create no more caps.)	- revocation requires changing many objects
+ easy to give access to new subjects	- giving access to new subjects requires updating objects
+ enables delegation	- difficult to delegate
+ enumerating objects accessible	- hard to enumerate accessible objects per user
- hard to enumerate legitimate users that can access	+ enumerating legitimate users is easy

DAC

Discretionary **A**ccess **C**ontrol. Is called discretionary, because subject can pass on his permissions directly/indirectly to other subjects.

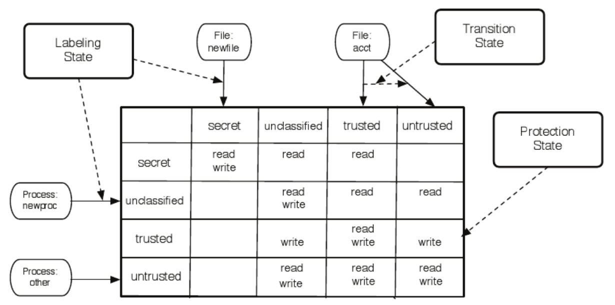
- allow untrusted processes/users to modify protection state
- only works if all processes are harmless and users make no mistakes

MAC

Mandatory **A**ccess **Control**

- only modifiable by admin
- subjects and obj are labels, state represents allowed operations
- transition state is legal way to relabel subjects and objs

MAC



	secret	unclassified	trusted	untrusted
secret	read write	read	read	
unclassified		read write	read	read
trusted		write	read write	write
untrusted		read write	read write	read write

Figure 2.2: A Mandatory Protection System: The *protection state* is defined in terms of labels and is immutable. The immutable *labeling state* and *transition state* enable the definition and management of labels for system subjects and objects.

Reference Monitor

input request of security sensitive operation

output binary answer

authorization module converts request into query for policy store

Secure OS iff system uses reference monitor which satisfies:

1. complete mediation
2. tamperproof
3. verifiable

Does not protect against TOCTTOU attacks.

Covert Channels

Capability to transfer information between processes that are not supposed to be allowed to communicate by the security policy.

There are **storage** and **timing** channels.

Prevent it by:

- memoryless program
- isolate program
- transitivity (if program A is confined and calls program B, then B must also be confined)
- masking (caller determines all inputs to all channels)
- enforcement (enforce that input into covert channel conforms with caller's [security] specification)

8.2 Unix Access Control

Chmod

Only the owner of a file (or `root`) can change file permissions. Even if a user has write access via group permissions he is not able to change it.

Setuid

When an executable file's `setuid` permission is set, users may execute that program with a level of access that matches the user who owns the file.

The kernel ignores `setuid` on scripts for security reasons.

Directory Permissions

read bit (`r`)(`4`) allows to list the files within the directory

write bit (`w`)(`2`) allows to create, rename, or delete files within the directory, and modify the directory's attributes

execute bit (`x`)(`1`) allows to enter the directory, and access files and directories inside (without execute permission we can **not** list, write or create files in the directory)

sticky bit (`T`, or `t` if the execute bit is set for `others`) states that files and directories within that directory may only be deleted or renamed by their owner (or root)

setuid no effects

setgid causes any file created in a directory to inherit the directory's group

`sudo`, `su`

`sudo`:

- used to gain root access using the user's own password.
- users with sudo access defined in `/etc/sudoers` or part of `sudo` group.

`su`:

- used to switch between users using the target user's password

`sudo` allows more fine-grained configuration and thereby more selective privilege escalation. ⇒ prefer `sudo` over `su`.

8.3 Linux Security Model

Process in user space has access rights based on user ID (UID).

UID 0 is reserved for `root`.

kernel process can access everything

root process can order kernel process to access everything

Kernel verifies permissions before system call! Compares UID and GID (Group ID) of subject and object.

Password change

- `/bin/passwd` has suid bit set and thus runs as root.
- This is needed to change the passwords in `/etc/passwd`.
- `/bin/passwd` can access **any** system ressource.
- ⇒ must trust `/bin/passwd`

Many programs can run as root and thus can modify the shadow password file!

8.4 Linux Security Module

Linux Security Module (LSM) is a framework with the ideas:

- Hook security functions and security data structures
- Allow registration and initialization of security modules (e.g. SELinux)
- Allow certain security attributes to be available to userspace services through `/proc`
- Limit performance overhead

8.5 SELinux

No default superuser exists, because SELinuxs default is to deny access. Everything must be specifically granted.

RBAC: Role based access control

- rules specify roles user can use
- rules specify when and where a user can transition into another role

MLS: multi level security

- handling of classified data where *no read up* or *write down* is allowed
- enforce by file system labeling

Modes

SELinux has the following modes:

Disabled: loaded, but not running. Only DAC is enforced

Permissive: Policy is not enforced, but would-be denials are logged

Enforcing: Policy is enforced, access is denied based on rules

Type enforcement

In SELinux every subject/object has a **security context** which consists of three labels:

user label on a subject: user privileges, on an object: objects owner

role label on a subject: subject's role, on an object: no meaning (always defaults to `object_r`)

domain label set of subjects and objects which are allowed to interact with each other

This is called **type enforcement**! SELinux can make two types of decisions:

access decision Is subject allowed to do something within it's domain?

transition decision Is subject allowed to do somthing in another domain (transition into this domain)?

Hard part is not to grant access to files, but allowing **safe domain transitions**.

Allow rule

four elements:

Source type: Domain type of a process attempting access

Target type: Type of an object being accessed by the process

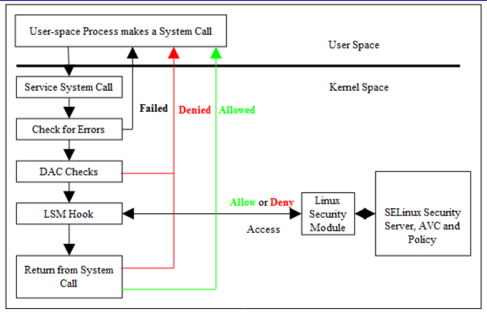
Object class: Class of object that the specified access is permitted

Permission: Kind of access that the source type is allowed

Example rule for passwd (access to shadow file missing):

```
allow user_t passwd_exec_t:  file { getattr execute };
allow passwd_t passwd_exec_t:  file entrypoint;
allow user_t passwd_t:  process transition;
```

System call



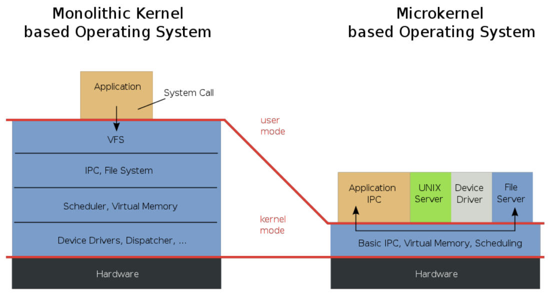
8.6 Other OSs

Qubes

Security by compartmentalization. 1 qube equals 1 VM.

Microkernels

Idea is to minimize TCB. Follows least priviledge principle and removes many parts from kernel space.



Monolithic Kernel based Operating System
Microkernel based Operating System
Application
System Call
user mode
VFS
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
kernel mode
Hardware
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling
Hardware

sel4 is a microkernel which focuses on efficency and high assurance. Uses formal verification of code to guarantee correctness w.r.t. specification.

9 Mobile Platform Security

All concepts and issued discussed below are in Android, but may also apply to iOS or other mobile OSs.

9.1 Mobile OS Security

Basics

Permission-based access control is key design decision for Android. Each app has private storage, own process and no communication to other apps (default). If security sensitive operation is required use **syscall** or **Android Binders**.

Each app has a designated UID.

Android Binders

Allows IPC and RPC (Remote Procedure Call).
Example: Accessing location goes through binder which creates channel between unprivileged app and privileged process in application framework.
Whether an app can use/call a binder is checked upon invocation based on the requested and granted permissions of an app.

Permissions

Three categories of permissions:

normal granted during install by default

dangerous prompt user during runtime

signature granted during install if signature matches signature of app which declared this permission

special must be declared in manifest and granted by user (e.g. system overlays)

Android implements two reference monitors in different places:

App-Framework for system APIs

Linux-Kernel for IPC calls to other apps

Problem with permission based design is manifold:

1. Users can't associate privacy risks with permissions
2. How fine-grained should permissions be?
   - Too fine-grained ⇒ Too many permissions ⇒ Unusable
   - Too coarse-grained ⇒ No real restriction ⇒ Less Secure
3. Habituation
4. Developers tend to request too many permissions
5. Developers fail to protect IPC interfaces (confused deputy attack)
   a) App A provides service that requires permission X
   b) App B does not have X
   c) B calls A → A performs service for B → A functions as deputy

Application Signing

- Applications have to be signed by the developers
- All Updates have to be signed with the same key

⇒ Easy to check for legitimate updates

- Signature matches app to developer, but real ID is not checked

⇒ Signature does not imply trustworthiness

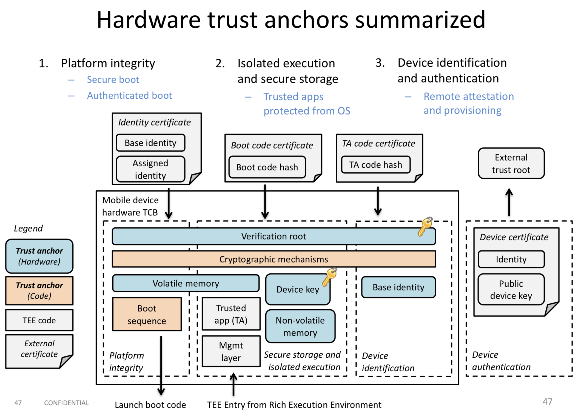
Lessons Learned Android

- permissions concept failed in practice
- MAC for Android can help with privilege separation! Research required
- standard web technology for apps goods if sandboxed on platform
- further simplify APIs
- centralise SW distribution to protect against malicious apps
- vendor customization causes fragmentation and hinders updates

9.2 Mobile Hardware Security

Trust anchors

1. platform integrity
2. isolated execution and secure storage
3. device authentication and remote attestation



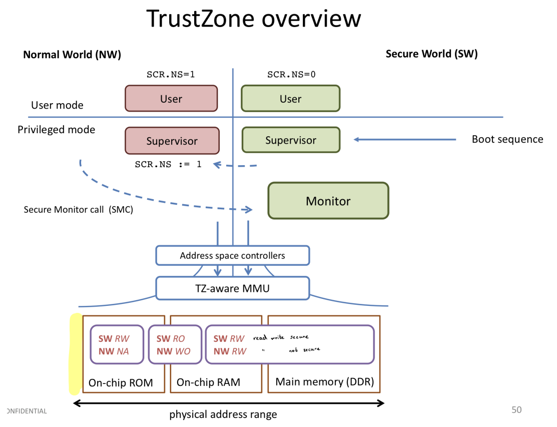
TrustZone

TrustZone apps share the same execution environment. Therefor they mutual trust each other **or** are isolated by trusted code (TEE management layer).

TrustZone does not specify remote attestation.

App development not open.

TrustZone enables secure communication with external resources (via status flag on bus).



Possible attacks against TrustZone:

- side-channel similar to meltdown
- glitch attacks (e.g. PlunderVolt)
- replay attacks (e.g. NAND mirroring)

10 Trusted Computing & Attestation

Acronyms:

TCG Trusted Computing Group

TPM Trusted Platform Module

PCR Platform Configuration Register

TCB Trusted Computing Base

SRTM/DRTM Static / Dynamic Root of Trust for Measurement

SLV Secure Loader Block

TXT Trusted Execution Technology

AIK Attestation Identity Keys (for signing PCRs)

SRK Storage Root Key

IEE Isolated Execution Environment

DMA Direct Memory Access

10.1 Some Current Approaches

Program Code in ROM

Approach: keep entire program in ROM

Advantage: Simple, No injection possible

Disadvantage: No updates possible, ROP attacks possible, no isolation

Verdict: Not practical

Secure Boot

Approach: Only load code with valid signature

Advantage: Only approved software loaded

Disadvantage: No isolation, difficult to prevent roll-back attacks

Verdict: Weak security guarantee

Virtual-machine-based Isolation

Approach: Isolate apps by executing in VM

Advantage: smaller TCB, isolation between apps

Disadvantage: VM still large & part of TCB, complex

Verdict: Step in right direction

10.2 General Approach

1. Isolated Execution
2. Remote Attestation
3. Sealed Storage

10.3 TPM

Basics

Attestation enables verifier to verify what SW is executing on untrusted device

Adversary model of TPM:

- remote attacks
  - compromise OS and apps running on the OS
  - complete control over network communication
- local hardware
  - generally trusted (because HW attack detection is hard)
  - attacker can reboot, install malicious SW, attack malicious USB devices

Basic TPM functions

- PCR registers for integrity measurement chain $PCR_{new} = \texttt{SHA-1}(PCR_{old}||\texttt{SHA-1}(data))$
  - Static PCRs are only reset at system boot.
  - Dynamic PCRs initialized to -1 at system boot, reset to 0 upon entry into IEE.
- on-chip storage for SRK
- manufacturer certificate
- remote attestation with PCRs and AIK
- sealed storage with PCRs and SRK (only accessible under certain integrity measurement)
- random number generator

TPM is **passive** component and **not** tamper proof.

HW attacks possible because TPM is on LPC bus.

Attested Boot a.k.a. Static Root of Trust

Measure all executed SW to verify platform configuration. To verify compare measurements to known hashes of trusted SW.

Shortcomings:

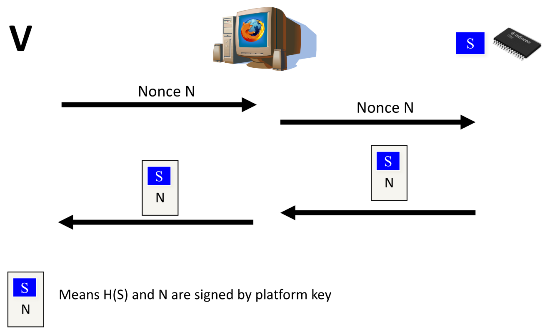
- TOCTTOU: measurement at load-time and not at runtime (inefficient against dynamic attacks)
- coarse-grained: TCP is entire system
- no guarantee of execution
- every system is different ⇒ Database of correct hashes is large

Dynamic Root of Trust a.k.a. Late Launch

Idea: special CPU instruction to create IEE → high assurance of code execution and remote attestation

Required computing primitives:

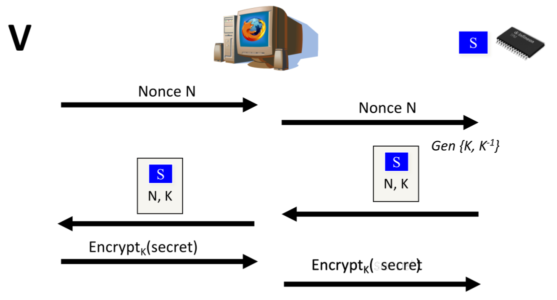
- create IEE with `SKINIT/SENTER`
  1. CPU softreset
  2. reset dynamic PCRs
  3. enable DMA protection
  4. send SLB to TPM
  5. execute SLB
- remote attestation



1. verifier sends nonce to TPM
2. TPM returns measurements, nonce signed with platform key

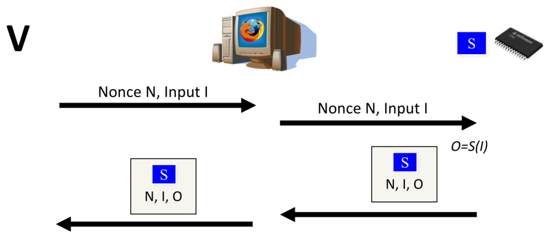
Attestation says, that *a* TPM signed it, but not *which* ⇒ Possible attack!

- establish secure channel



1. verifier sends nonce to TPM
2. TPM generates $\{K, K^{-1}\}$, send back $\{\texttt{measurement, nonce}, K\}$ signed with platform key

- verify output



1. verifier sends nonce to TPM
2. TPM sends back measurement, nonce, input and output signed with platform key

No need to trust OS anymore, therefor minimal TCB.

10.4 SGX vs TPM

SGX advantages:

- Memory encryption
- robust against LPC-bus tampering
- can execute unprivileged code (ring 3)
- multi-threaded execution
- parallel execution of enclave and untrusted code
- enclaves can be interrupted

SGX disadvantages:

- no sealed storage
- remote attestation requires online third party (Intel)
- memory access pattern reveals information about computation

11 Software-Only Root of Trust

Problem: How can we achieve DRTM on machines without HW support such as `SKINIT/SENTER` or SGX?

Goal: Externally verifiable code execution without specialised HW

11.1 Reflection

1. fill entire memory with random data
2. clear system state and disable interrupts (secure-ish execution env)
3. compute hash over memory
4. return hash and system state to verifier

Verifier checks time it took to compute response, received hash and system state. To protect against replay attacks (or precomputed results) verifier asks for complete memory hash and two hashes over random intervalls.

11.2 Genuinity

Slightly different problem setting: Verifier wants to check code integrity, code execution and that the **code ran on the machine** it was expected to run on.

1. verifier sends checksum code to client
2. code uses pseudo random access pattern → causes cache misses → alters performance counters
3. incorporate those hardware parameters into hash

Simulation of the above would be significantly slower.

11.3 SWATT

Rellies on optimal code such that attack can not optimize to gain time.

1. verifier sends nonce as seed for random access pattern
2. walk over memory and compute hash

Attacker would need to check and replace every memory read which reveals its presence. Time consuming task!

Infeasible for large memory. Therefor only use partial memory scans → open new attack vectors such as memory copy attacks.

ICE - Improved Checksum

Include program counter and data pointer in checksum. For any memory copy attack one will differ from the original one.

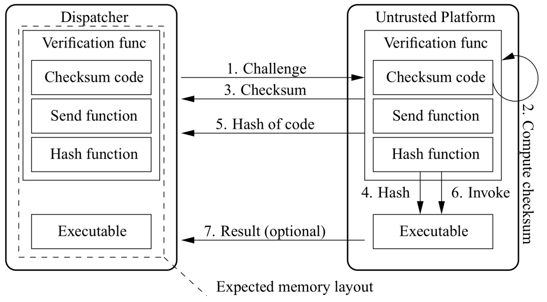
ICE Key Exchange

1. receive challenge from remote party
2. compute checksum and setup TEE (disable interrupts etc.)
3. compute secrets in TEE
4. send back public results

Prevents MITM attacks. Attacker can even know entire memory before protocol runs.

Pioneer

1. verifiy integrity through SW attestation
2. setup TEE
3. setup and run code in TEE



Replace interrupt handlers to prevent attacker to interrupt verification function or executable.

Trick: Write checksum to stack, replace handlers right after checksum computation. If attacker causes an interrupt, the interrupt will overwrite stack with checksum → attestation fails.

Drawback: Requires defense against proxy and overclocking attack.

12 IoT

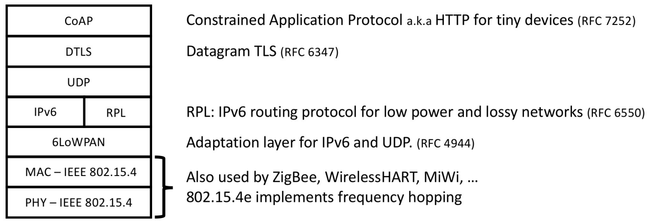
12.1 Light bulbs go nuclear

1. extract firmware update signing key using differential power analysis
2. circumvent proximity checks
3. make bulb join malicious network
4. perform OTA firmware update

Note: classical defence mechanisms like firewalls are not effective

Works because firmware only includes a MAC and is not signed (asymmetrically), therefor extracted key can *sign* new firmware updates.

IoT Stack


CoAP
DTLS
UDP
IPv6
RPL
6LoWPAN
MAC – IEEE 802.15.4
PHY – IEEE 802.15.4
Constrained Application Protocol a.k.a HTTP for tiny devices (RFC 7252)
Datagram TLS (RFC 6347)
RPL: IPv6 routing protocol for low power and lossy networks (RFC 6550)
Adaptation layer for IPv6 and UDP. (RFC 4944)
Also used by ZigBee, WirelessHART, MiWi, …
802.15.4e implements frequency hopping

Encrypt everything?

Not sufficient! device identifiers, traffic volume still leak information, activities

existential leakage single transmission implies real-world event

- Can not be fully eliminated

statistical leakage deviation from normal implies real-world event

- Can be eliminated.

Hide transmission times and encrypt?

Not sufficient! Examples:

- VOIP word reconstruction from encrypted packages (leak from variable bit length encoding and length-preserving encryption).
- side-channel leaks of web apps by analysing traffic

12.2 Device Pairing

Pairing process of establishing a security association between two devices without prior shared knowledge

security association can be anything only known to paired devices (e.g. secret key)

Paring method shall minimise I/O hardware needed and avoid complex user interactions.

Out-of-band extra channel which guarantees authenticity and is assumed to be secure even under MITM attack
Examples:

- Visible Light
- Infrared
- Audio
- Haptic
- Sensing

In-band often relies on physical channel properties and use special encoding (etc.) to detect MITM attack easily.

Bluetooth Low Energy

Four authentication methods supported:

Just Works provides no authentication.

Passkey Entry displays a 6-digit PIN code on one device, and the user needs to type the PIN on the other.

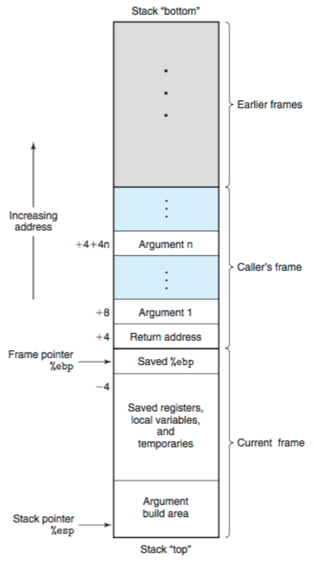
Numeric Comparison displays 6-digit PIN codes on both devices and the user needs to confirm that they are identical.

Out-of-band (OOB) is directly provided with a shared key and assumes the key has been securely established via an alternative OOB channel.

Devices can negotiate and select one based on their I/O capabilities.

13 Stack Attacks

13.1 Stack Frame



Construction

The steps needed to call a function and construct a new stack frame are

1. Push the last argument to the top of the stack;
2. Push the first argument onto the stack;
3. Push the return address onto the stack;
4. Jumping to the called function;
5. Saving the caller's base pointer on the stack;
6. Move the base pointer to the top of the stack
7. Allocating local memory for the callee.

Destruction

The steps needed to seamlessly return and resume the execution of the parent (caller) function are:

1. Moving the stack pointer to the bottom of the callee's frame (value of `ebp`)
2. Restoring the caller's base pointer by popping it
3. Taking the return address off the stack and jumping to it
4. Freeing space allocated for the callee arguments

13.2 Protection

Canaries: A known value that should never change is saved on the stack. If the value is checked and is not the value it should be, the system can conclude that there was a buffer overflow.

Address space layout randomization (ASLR): The addresses of stack, heap and libraries are moved around. Because of this, the addresses of specific calls are different on every call, and attacker can't find the correct address by trial and error.

13.3 Format String attacks

A program such as

```c
int main (int argc, char** argv) {
    char *name = argv[1];
    printf(name);
}
```

is vulnerable to format string attacks. Interesting inputs are:

`printf(%s)`: This will lead the program to print out parts of the memory.

`printf(%n, &i)`: Writes the number of characters printed so far to the address `&i`.

A Comparison SGX, TrustZone and TCG

	SGX	TrustZone	TCG
Execution environment of different apps	isolated	same (small TEE OS)	
Remote attestation	defines protocol and service over third party	not specified	no third party required
Memory encryption	yes		no
Sealed storage	no		yes
Development	open	proprietary	
Support for parallelism	Yes		No
Interrupt Enclaves	Yes		No