System Security HS19

jasonf

September 2019

Contents

1 Background

	confidentiality	authentication
Symmetric	1.symmetric key encryption	2. symmetric key authentication
Asymmetric	3. public key encryption	4. digital signatures

1.1 Symmetric Crypto for Confidentiality

- **Stream Cipher:** processes a message bit by bit (RC4).
- **Block Cipher:** encrypts/decrypts a message in blocks (DES, AES). A secret key indicates which mapping to use: $2^N!$ such mappings, where N is block size. **Ideal Block Cipher:** Use any of the possible mappings → keyspace would be to big instead we approximate with **Practical Block Ciphers**. Use key of K bits to specify a random subset of 2^K mappings.
  - **Diffusion:** the ciphertext bits should depend on the plaintext bits in a complex way. If ptext bit i is changed, the ctext bit j should change with $p = 1/2$.
  - **Condusion:** Each bit of c should depend on the whole key. If one bit of the key is changed the ciphertext should entirely change.
  - **TODO:** repeat Feistel Network and AES sl.9.

1.2 Symmetric Crypto for Authentication

- **MAC (Message Authentication Code)** message and key are processed through MAC function to get MAC. Receiver recomputes MAC using the shared key and verifies that it is as expected.

1.3 Asymmetric Crypto for Confidentiality

- **Public/Private Key encryption:** Encrypt using public key decrypt using private key (RSA).

1.4 Recap of security models and proofs

Security model and properties:
- CPA: secure against chosen plaintext adversary.

Security Proofs:
- Schemes proven secure using hardness assumptions (e.g. one cannot factor n).

Padding Oracle Attack: TODO exercise session 1

2 Side Channel Attacks

Security proofs are based on **models of system and attacker**. Models do not take into account the implementation and that it interacts with the environment. E.g. observing side information leakage.

2.1 Classic timing attack on RSA

- leverage execution time to recover the key.
- Applies to many asymmetric cryptosystems: RSA, EL Gamal, Diffie-Hellman.
- Cryptosystem vulnerable, when:
  1. its exec time depends on the key
  2. exec time measurable for different inputs.
- look at side channel attack (sl.21ff)
- **Hamming weight:** detect how many "1"s in key - Greatly reduces search space but still huge. Due to Key-dependant Branching in code:
- In Montgomery multiplication the exec time depends on key and input (intermediary value).
- Attack main idea: Try large nr of different messages and leverage avg execution time (with statistical test).
- TODO: look at Sidechannel attack on slides 25.ff.

2.1.1 Protecting against Timing Attacks

- Point defenses are easy (make the last reduction in montgomery mult unconditional)
- Generic Protection is hard (ensure all ops take sam time for all inputs.
- Challenge: Performance penalty can be very high, plus difficult to implement.

Masking by Kocher
- Choose different random X for each message M

$$SIGN(m) = [(m * X^d \, mod n] * [(X^-1)^d \, mod n] mod n = m^d \, mod n$$

- $(X^-1)^d \, mod n$ can be computed in advance. Two additional multiplications

2.2 Cache attack on SGX enclaves (irrelevant)

Let us now consider a collocated adversary (e.g. one VM in cloud platform attacks another VM). Attacker and VM share resources like caches, DRAM, disk.

2.2.1 Prime and Probe attack:

- Variant1: Data access depends on secret data (e.g. read var x or write var y based on key). –> Data access patterns in the cache leak information about the secret.
- Variant2: Control flow depends on secret data (e.g. call function x() of y() base on key). –> Code access patterns in the cache leak information about the secret.
- TODO look at slide 36.

2.2.2 Protection against cache attacks

- Try to eliminate secret-dependent cache accesss patterns and control flow branches.
- E.g. Hardenet AES: when reading from one S-box touch each of them.

2.3 Untrusted OS (irrelevant)

Modern proessors include security architectures that attempt to protect applications from untrusted OS e.g.: Intels Software Guard Extensions **(SGX)** (enclave = security critical part of application). Outsourcing data and computation to untrusted cloud is such a scenario.

Background on SGX: OS cannot directly access enclave memory.
- Access control checks when enclave data processed inside the CPU.
- Data encrypted when it leaves the CPU.

Attacking SGX enclaves:
- Challenges: Speed, Noise, Attack detection, Enclave Hardening.
- Techniques: Performance Monitoring COunters(PMC), isolate victim, Hyperthreading, Attack target.

2.4 Cache-timing attacks on AES

- **Vulnerability:** Software implementations using look-up tables to perform internal operations of the cipher, such as S-boxes. Include both the reference and optimized implementations of AES.
- TODO: look at AES reference and optimized algorithm.
- **Basic Idea:** The time needed to access a specific position in the lookup table depends on the address of this position. Attacker compares access time to same setup locally:
  1. For each byte of key, one index value will ave the slowest lookup. → Total lookup will be slow. Find this message byte n that is slowest for constant random rest of message.
  2. Find locally the slowest index i with same setup.
  3. $K[0] \bigoplus n[0] = i$ (modular keysize K).
- **Countermeasures:**
  - Avoid memory access.
  - Alternative lookup tables (different AES formulations).
  - Data-oblivious memory access pattern.
  - Hiding the timing.

  → Most modern CPUS have special AES hardware that does not use RAM! → not vulnerable.

2.5 Power Analysis Attacks:

An attack that enables to retrieve a secret key by observing power traces of a device.
- **Types:**
  - **Simple Power Analysis SPA:** Observation and analysis of power consumption during a single execution (trace); trace depends on the key
  - **Differential Power Analysis:** Statistical analysis of multiple measurements based on the similar principles as timing cryptanalysis; trace depends on the key and on the input plaintext/ciphertext.
  - **High Order Differential Power Analysis:** Complex statistical analysis of multiple measurements
- **Devices:** Mainly used on Smartcards, RFID chips, Sensor Nodes. And the Attacker needs physical access.Needs additional equipment like Oscilloscope.
- **SPA-Main Idea:**
  1. Measure power consumption of instruction sequences that depend on the key (square vs. multiply in RSA).
  2. The key is 1 where there is a multiply after square.
- **DPA-Main Idea:** Power consumption depends not only on the type of the executed instructions but also on values of operands (unlike in the simple power analysis)
- **Protection:**
  - Desynchronization: Random injection of dummy instructions.
  - Noise generator: but then correlations can still be exploited using larger nr of measurements.
  - Software balancing: significant reduction of speed.
  - Shamir's countermeasure: add 2 capacitors s.t. one at a time is charging an one discharging. Power consumption doesn't directly depend on calculations.

2.6 Accoustic attack on RSA:

High frequency sounds caused by vibration of electronic components (capacitors and coils) in the computer's voltage regulation circuit
Different secret keys cause different operations with different power levels (as seen before)
Voltage Regulator changes behavior to stabilize voltage
causes different vibrations → emitted as different sounds
Can't detect individual CPU operations. But modular exponentiation takes more time → Try to find longer patterns

2.7 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,

2.8 Tamper-Resilience

- **Tamper resistance:** systems take the bank vault approach. (e.g. Smartcards)
- **Tamper responding:** systems use the burglar alarm approach.
- **Tamper evident:** systems are designed to ensure that if a break-in occurs, evidence of the break-in is left behind.

Specialized Devices
- Smartcards:
  - hold secret keys
  - perform crypto-operations (dedicated)
  - access-protection with a PIN
  - Limited tamper resistance
- Cryptographical Co-processors:
  - hold secret keys
  - perform crypto-operations (broader set)
  - TPM functionality
  - access protected with a (master) key
  - Tamper resistance

2.8.1 Smartcards

Invasive Attacks

Semi-Invasive Attacks
- **Glitch-Attack:** In a glitch attack, attacker deliberately generate a malfunction that causes one or more flipflops to adopt the wrong state. Hardware and Software countermeasures are taken.

2.8.2 HSM Hardware Security Module

Typical Attacks on HSM:
- Cryptanalysis: the attacker exploits design flaws in the crypto primitives such as encryption algorithms, hash functions, and digital signature schemes (cryptanalysis techniques)
- Protocol analysis: flaws in the protocols in which crypto primitives are used (numerous formal analysis techniques)
  API attacks (introduced by Andersson), extend protocol analysis to APIs

Security API
- the top-level software component of a cryptoprocessor, which governs its interaction with the outside world.
- It extends a cryptographic API by enforcing policy on the interactions as well as providing cryptographic services.

API-Attack:

ISO-0 Attack:

Some Solutions:
- Access Control
- Limit functionality: Enable only what an entity requires is allow to access. Disable all other access to data and functions.
- Formal analysis techniques similar to the ones use for protocols.

Doesn't matter how secure the device is physically if it leaks secrets due to API attacks

Chip and Pin is broken: → read paper

3 Introduction to Security on Commodity Systems

Several Things have to be protected:
- OS: manages and shares hardware resources between applications.
- Hardware resources: CPU, memory, peripherals
- Application: code, data (volatile, persistent)

3.1 Application Security

Three important properties:
1. **Launch-time integrity:** pristine/correct application was started or loaded. (Hash verification of initial code, data)
2. **Run-time isolation:** no interference from malicious software, peripherals. (Prevention of unauthorized modification of code and data, and prevention of runtime attacks e.g. code injection)
3. **Secure persistent storage:** Confidentiality, integrity protection of persistent data.

3.1.1 Trusted OS-based Solution

What type of hardware-support in x86-systems for OS-based application security?

Three main components:
- **Processor:** contains one or more CPU's
- **Chipset:** connects the processor to memory(RAM) and peripherals.
- **Peripherals:** connect via various bus-interfaces to the chipset.

Privilege Rings
0. Kernel
1. Device drivers
2. Device drivers
3. Applications

Memory Management Unit (MMU) Translates virtual addresses to physical addresses by traversing the appropriate page tables (kernel page table, app1 page table)

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

Firewire DMA Access to RAM is tightly controlled by the CPU. But this can be circumvented through DMA(Direct Memory Access)
→ allows fast communication speeds between devices.
→ Attacker can use Firewire cable to issue a DMA request to fetch the contents of RAM. **Solution:** IOMMU controls the DMA accesses!

Attacks using physical access Physical Access Attacks are harder to defend for the OS. E.g remove hard drive and plug it into other system.

Some broken Solutions:
- BIOS Protection: Prevent booting from external sources (Broken!)
- Disk Encryption: The entire disk is encrypted and unlocked providing a secret key (possible to find key can still wipe the disk)
- Disk Encryption 2. Attempt: use password only, decryption key is protected by password (can brute force password)
- Disk Encryption 3. Attempt: leverage a secure element Trusted Platform Module (TPM) chip. Use password to encrypt decryption key on TPM.

Disk Encryption Data on disk is always stored encrypted. User supplies a key/password to the encryption layer. File System is completely unaware. Regardless of the unlocking technique, the encryption key must be kept in memory. Therefore Encryption key can be read even after platform is powered off (cold boot attack).

Cold Boot Attack: Assumption that DRAM loses its content soon as a machine remains without power is wrong. **Solutions:**
- Erase Key: user needs to type in password often.
- Prevent Booting from external media. Does not prevent DRAM component transfers
- Physical Protection: components that respond to enclosure opening or low temperatures (expensive)
- Avoid placing key in memory (requires architectural changes.

TPM Support: lecture 4 slide 39ff

3.2 Application Security: Trusted Execution Environments

Secure launch → Isolated Execution → Secure Persistent Storage

	Hardware-support for **OS-based Security**	Hardware-support for **application security**
CPU	Privilege rings Memory Management Unit	Execution modes, access control checks, memory encryption... (SGX)
Chipset	DMA Remapping tables	
Peripherals	Trusted Platform Module Normal HDD with OS-enforced access control	

Lesson Learned: Trusting OS can make some security functions easier but as the OS is complex its prone to errors.

ARM TrustZone (a TEE) Secure virtual processor realized as a special CPU-Mode, managed by a small trusted OS

Intel SGX (Software Guard Extension) (a TEE) Untrusted OS manages TEE memory but cannot access it.

Bad USB (Attacks by Untrusted Peripherals) change a USB device controller to mimic another device class. E.g. Storage device acts as keyboard and opens console.

Case Study: Smartphone Storage Protection No TPM! Solution: leverage hardware support in the platform (processor specific keys(ARM TrustZone)). Processor chip has device-specific encryption keys - Extracting such keys not easy risk damaging chip. Use PIN and key to derive storage encryption key. Prevents brute forcing of extracted storage.

How to control PIN attempts? slide 52 lecture 4

3.3 Intel SGX

Intel Software Guard Extensions: Intels new architecture containing new instructions and protecive mechanism in the processor. → Hardening even if you trust the OS.

- Company app calls the enclave, which then executes and returns.
- Enclave memory is encrypted (and integrity protected) at the processor boundary.

Sealing:
- Enclave has no direct access to disk or IO/ No access to persistent storage.
- No direct access to trusted clock, limited support for counters
- Can do sealing: store encrypted confidential data on disk.
- When building an enclave, SGX generates a cryptographic log of all the build activities.(MRENCLAVE is a digest of the logs of build process and used as id of an enclave)
- Same ID on the same platform can unseal what was previously sealed.

Secure communication to/from Enclave: Create Enclave Secure Channel....

Remote Attestation: During manufacturing 2 keys are burned into the CPU:
- Fused Seal Key is used as Processors secret
- Provisioning Key serves as a proof for remote Platform

3.4 DelegaTEE: Brokered Delegation using TEEs

Reading!!

Properties:
- The Owners credentials remain confidential.
- The Owner can restrict access to his account, e.g. in terms of time, duration of access, no. of reads/writes etc.
- The system logs the actions of Owners and Delegatees so that post-hoc attribution of their behaviour is possible.
- The system minimizes the ability of a service to distinguish between access by the Delegatee and that of the legitimate Owner.
- Owner does not have to always be online.

4 Computer Architecture

4.1 Computer Architecture Introduction

- Architecture:
  - Abstract Model
  - Instruction Set Architecture (ISA) specifies the interface between software and hardware.
  - Visible to Software.
  - E.g. X86, ARM, RISC-V
- Microarchitecture:
  - Actual implementation of architecture
  - Follows ISA specification
  - E.g. Intel Core i7, AMD Ryzen, ARM Cortex-A53

(Micro-) Architectural State
- Registers
- Main Memory (Stack, Heap)
- Caches
- Branch prediction history
- Reorder buffer
- TLB

Assembly Read chapter in slides!

Optimizations
- Memory Wall: memory is too small
- Solution: Hide the memory latency (Caches, Pipelining, Branch prediction, Out of Order execution)

Cache
- Cache is layered (L1, L2, L3) L3 is usually shared among cores.
- Cache is shared across all applications
- Cache location depends on data address

Pipelining
- Split instructions into small steps: inst fetch (IF), inst decode (ID), execute (EX), memory access (MEM) and register write back (WB).
- 10 cycles to execute 2 instructions.
- Run steps of instructions in parallel.
- Branches: stalls pipeline b.c. next instruction is not known

Out of Order (OoO) Execution:
- Parallelize execute stage
- Utilize all execution units (ALU, FPU,...)
- Requires no inter-instruction dependency.

4.2 Cache side channel attacks

Caches are shared across all applications on a processor. Cache misses leak information. Attacker must control an application on the same Processor/System (Cloud hostin, User vs Kernel space, Javascript)

Cache Side Channel - Flush+Reload
1. Attacker flushes some shared memory in cache.
2. Victim tries to access it and thus reloads it.
3. Attacker accesses it again and gets fast access.

Cache Side Channel - Prime+Probe
- Works also without shared memory.
- Attacker fills cache with his own data (Prime)
- Victim loads some data
- Attacker probes memory (fast access means victim did not access cache)

4.3 Meltdown

Memory Organization - virtual memory
- Abstracts physical memory to processes.
- Each process operates on it own seperate virtual memory (achieves isolation)
- If a process allocates more virtual memory than physically available, either: 1. some of its memory must be swapped to disk or 2. the process must be killed. The physical memory is shared among processes concurrently running.
- Translation of adresses is done through page tables. The OS sets up and maintains the PT. Translation is expensive requires multiple memory accesses.
- A page table entry (PTE) contains permission bits for a page(read only, executable, is supervisor page)

Memory Organization - Kernel
- System calls require a context switch to kernel space (expensive).
- For efficiency modern OS map the whole kernel memory in every process virtual adress space.
- Permissions are enforced in hardware.
- Example mem access read in slides lecture 5 sl. 60
- B.c. mem access values are stored in temp register before it is checked whether permissions apply other inst can use the data before the instruction retires.

Meltdown
- An invalid memory access still fetches data from memory. Permissions are only checked during retirement.
- The returned data becomes available to subsequent instructions.
- No change is architecturally visible (the pipeline is flushed during step 6
- However transient instr (inst that entered pipeline but should not have been issued) have microarchitectural side effects. They modify the cache!
- Meltdown read slide 65 ff.

4.4 Spectre

Microarchitecture - Branch Prediction
- Next instr. is known only when branch instr. is completed.
- Idea: Predict the outcome and proceed with the execution. Misprediction cost up to 20 cylcles b.c. of rollback. But if most of the time we predict correctly performance improves.
- Instructions predicted by the CPU are called speculative instructions. (current CPU's speculate correctly in 95% of times)
- Branch predictor Implementations:
  - Static predictions: e.g. choose most common jump address at compile time.
  - Dynamic predictors: use info gathered at run time to make a choice (branch history):
    - Branch Target Buffer (BTB): stores target addresses of previous instances of a branch.
    - Branch History Buffer (BHB): allows to choose different entries in the BTB for the same branch based on its history

Problems with BTB:
- Takes as input a branch virual address and stores the taken target.
- BTB does not store any info about process ID a virtual address belongs to.
- BTB is not flushed on context switch (i.e. when scheduling another process on a core.

Spectre slide 90!!
1. Pretrain the branch predictor.
2. Execute a sequence of instructions that accesses a secret value.
3. Cause a secret-dependent state change to the microarchitectural state.
4. Observe the microachitectural changes from an attacking process (side channel)
5. Repeat for all interesting addresses.

Defense
- Google developed retpoline: replaces all branches, no branch prediction possible, big performance impact.

5 Operating System Security

- OS controls access to resources
- OS schedules processes
- OS offers security to processes
- safety: Users can perform only authorized operations
- least privilege: Processes perform only their necessary operations
- MLS: Operations can only permit information to be written to more secret levels.

Security Goals
- Secrecy: who can read what files?
- Integrity: who can write to what files?
- Availability: what processes can consume storage/CPU/memory?

Trust Model The set of software and data upon which the system depends for correct enforcement of system security goals. Ideally as little as absolutely necessary. Also known as Trusted Computing Base TCB.

Linux TCB: Kernel, modules, shell, system tools, X server, etc.

Threat Model Set of operations that an attacker may use to compromise. **Compromise:** violate a security goal by finding a vulnerability

5.1 Access Control

The selective restriction of access to a resource. Whether to allow requests from multiple subjects to perform operations on objects.

The security requirements of an OS are defined by its protection system
- A protection system is comprised of a protection state and protection state operations.
- Protection state - Operations that subjects can perform on objects
- Protection state operations - Enable modification of state

Access Matrix Consists of a set of subjects s a set of objects o a set of operations op and a function ops(s, o), which determines the operations that subj s can perform on object o. Matrix can be stored by row or column:
- By column: Access control list (ACL) stored with resource.
- By row: Capability of a given process or user

Unix File Protection: ACL (by column)
- Subjects: owner, group, others
- Operations: read (r), write (w), execute (x)
- rights are stored with the target object to be accessed.

Page Table: Capability (by row) Page Tables define the access domain, access rights are stored with the accessor.

Capabilities vs ACLs slide 37

MAC and DAC
- DAC: A protection system that permits untrusted processes to modify the protection state is called a discretionary access control system. DAC works if all processes are benign, and users make no mistakes (impossible). A subject with certain access permissions can passs on exactly these permissions.
- MAC: A mandatory protection system can only be modified by trusted administrators via trusted software.

Reference Monitor
- Input: a request to perform a security-sensitive operation
- Output: binary response indicating whether the request is authorized by the policy
- Authorization module - the brains of the reference monitor. Must convert requests into a query that can be looked up in the policy store.
- Policy store - a database of protection state, labeling state and transition state.

Secure OS definition: "A system with a reference monitor access enforcement mechanism that satisfies the requirements below:"
- Complete mediation: must mediate all security-sensitive operations.
- Tamperproof: cannot be modified or disabled by untrusted processes.
- Verifiable: small enough to be subject to analysis. Simple policies are intuitive to verify, but complexity makes verification hard

Covert Channels Prevent unauthorized communication among processes. Protection models we discussed so far are not sufficient, processes can leak information through covert channels. Preventing Covert Channels slide 56.

5.2 Linux Security Model

Kernel enforces object permissions!
Notice the difference between kernel and superuser access
– Kernel processes can access anything
– Root processes can order the kernel to access anything

UNIX File Concepts
- files administered using i(index)nodes (control structure with key info of a single file (attributes, permissions, ..)
- Inode table/list for all files on a disk, copied to memory when disk mounted.
- Directories form a hirarchical tree : are a file of names and inode numbers.
- Linux treats everything as a file! Filesystem security very important.

Users and Groups
- User account: (user, uid) represents someone capable of using files (humans, processes).
- Group account: (group, gid) is a list of user-accounts

Using permissions
- A file has owner and group id (sometimes several)
- A process has owner and group id
- Kernel verifies permissions before executing system calls. (first uid and then gid are compared in this order!)

Linux Security Transactions
- A user executes a program (if it has permissions)
- When running the process normally runs as the user and group of the person or process that executed it.
- Whoever owns an object can set or change its permissions. • This is Linux DAC model's real weakness: the system superuser account ("root") can both take ownership and change the permissions of all objects in the system. It is not uncommon for processes and administrator-users to run with root privileges, in ways that provide attackers with opportunities to hijack those privileges.

5.3 File Permissions

- Permissions: r, w, x
- For: Owner(User), Group, Others
- Directory permissions: r = list contents, w = create or delete files in dir, x = use anything in or change working dir to this dir (and read files if allowed by file permissions).
- The permissions are coded into the inodes
- Sticky bit: used on dirs to limit delete. If set must own file or dir to delete, other users cannot delete even if have write. (chmod + t)
- setuid bit: means program runs as owner no matter who executes it (chmod +s).
- setgid bit: means run as a memeber of the group which owns it. Only used on executable files, not shell scripts. Very dangerous!!
- setgid on dirs: causes any file created in a dir to inherit the dirs group.
- Changing passwords: Passwords are changed using the program /bin/passwd.
- Real UID = UID of the user running the program. Effective UID = UID of user with whose privileges the program runs.
- system calls slide 81
- Major problem in Unix is that everything is dependent on root and root has access to everything. Violation of principle of least privilege.

5.4 SE Linux

Linux uses DAC security model. But Mandatory Access Controls (MAC) imposes a global security policy on all users.
- users may not set controls weaker than policy.
- Normal admin done with accounts without authority to change the global sec policy
- But MAC systems have been hard to manage.
- In SELinux all access must be explicitly granted. Allows no access by default, regardless of the Linux user/group ID's
- No default superuser in SELinux, unlike root in standard Linux.

Role Based Access Control (RBAC)
- Rules specify Roles a user may assume
- other rules specify circumstances when a user may transition from one role to another.

Multi Level Security (MLS)
- Concerns handling of classified data (no read up, no write down)
- MLS is enforced via file system labeling

Security Contexts Each individual subject object in SELinux is governed by a security context:
- User: (human or daemon) user labels on subjects specify accounts privileges. user labels on objects specify its owner.
- Role: like a group, assumed by users. Only one role per user.
- Domain (type): a sandbox being a combination of subj. and obj. that may interact with each other.

This model is called **Type Enforcement (TE)**. Subj Type can access obj. type to perform Operations on Obj.

Decision Making in SELinux
1. access decisions: subj do things to obj that already exist, or create new things in expected domain.
2. transition decisions: invocation of processes or creation of obj in different types (domains) than their parent domains. Transition must be authorized by SELinux policy.

Why use Type Enforcement?
- It enables us to ensure that only the password program can access the shadow file, regardless of the user running the program (what normal unix AC cannot do)
- Problem slide 96ff. solution domain transitions!
- Providing for secure domain transition is analogous to the concept of setuid programs, but with the strength of type enforcement.

Domain Transition Rules all following three rules are necessary alone none is sufficient:
1. The process new domain type has entrypoint access to an executable file type.
2. The process current domain type has execute access to the entrypoint file type.
3. The process current domain type has transition access to the new domain type.

5.5 Securing Commercial Operating Systems

To make an OS secure, we need at least:
- A reference monitor: system to monitor and enforce a security policy.
- Complete mediation: all security-sensitive operations must be directed to the reference monitor.
- Tamperproof: enforcement mechanism cannot be modified by untrusted processes.
- Verifiable: Small enough to audit, prove it satisfies goals.

Examples
- Compartments: Use multiple vm's to decouple different programs.
- microkernels: Make the actual kernel as small as possible.

Microkernels:

5.5.1

Building a Secure Linux Linux Security Module (LSM) Framework.
Basic Idea:
- Hook security functions and security data structures
- Allow registration and initialization of security modules (e.g. AppArmor , GRSec , SELinux)
- Limit performance overhead (especially when no module is loaded)
- SELinux: MAC for linux, using LSM.

5.5.2

Android
- Built on top of the linux Kernel
- Several Security enhancements
- Security-Usability trade off.
- Java applications with native code components.
- Typically a single user. isolate apps not users.Each app has own Linux UID. (Apps cannot read eachothers files, memory and cannot exhaust all resources.

Android Permission Model:
- Restrict access to sensitive resources.
- Only accessible through OS (complete mediation)
- Application has to request permissions in manifest.
- Permission Enforcement: During critical system call.
-

Application Signing
- Applications have to be signed by the developers.
- All Updates have to be signed with the same key.
- Applications signed by the same key can request same UID
- Signature matches app to developer who has to register with Google (Signature does not imply trustworthiniess)
- Google performs malware scan in market (bouncer) but numerous malware has made it past bouncer.

5.5.3 Windows 10

Allows different configurations to enhance security:
- Enable Memory Protection: DEP, ASLR, ..
- UEFI Secure Boot
- Device Guard (contains whitelisted applications)
- BitLocker Drive Encryption
- AppContainer: Allows app sandboxing
- Windows Defender Antivirus

6 Mobile Platform Security

6.1 History

How is mobile security evolution different from PC security?

Different Shareholders
- Mobile network operators
- End users
- Regulators

6.2 Mobile OS Security

Security-related design decisions
- Software distribution
- Application isolation
- Defining access to resources
- Sharing functionality

Android architecture Linux-base OS, open source, many variants

Software Distribution
- Android allows software from multiple sources (unlike iOS).
- Marketplace provides security scanning.
- Application signing: Developer self-signs app (public key identifies developer, but a certain key does not imply developer @ UBS)

Application isolation
- Normal process separation
- Additionally Seperate UIDs for all applications
- Binder mediates IPC (reference monitor)
- Applications run in separate vm's

Defining Access to resources Typical approach:
1. Define security principals: sec principal = app (unlike Linux: user)
2. Implement reference monitor: two places: 1) in App framework for system APIs 2) part of OS for IPC calls to other apps.
3. Define security policy: Primarily apps request permissions, access to some resources granted automatically.

Permissions
- 4 Categories: Normal, Dangerous, Signature, SignatureOrSystem.
- Permissions are declared in the manifest.
- Problem few uers can associate privacy risks with respective permissions and also habituation. Also many developers request unnecessary permissions.

Android Malware
- Often implemented by repacking popular apps
- Detection: dist channels like Google Play.
- Detection Techniques:
  - static analysis: look for known malicious code.
  - Dynamic analysis: execute in simulated environment. (But malware can detect simulation and very expensive to explore all execution paths)
- Leverage the fact that Android malware is often distributed by repackaging many popular apps. Take diffs from related Apps and identify suspicious code segments.

6.3 Mobile Hardware Security

Why hardware security?
- Ensure that correct OS security framework is run on the device requires Platform Integrity. (Secure Boot, Authenticated Boot)
- Run payment app on smartphone, requires isolated execution, secure storage and remote attestation or device authentication for provisioning of payment credential. (recall SXG enclaves and their trust model)

Trust anchors: Minimal hardware elements and functionalities needed to implement the above security requirements.
TODO look at slides

TEE (ARM TrustZone)

7 Designing Secure Systems based on Trustworthy Computing and Attestation

Trusted Computing Base (TCB): of a computer system is the set of all hardware, firmware, and/or software components that are critical to its security, in the sense that bugs or vulnerabilities occurring inside the TCB might jeopardize the security properties of the entire system.

Is my computer secure? current approaches:
- Program code in ROM: keep entire program in ROM. Is simple and noone can inject any additional software. But cannot update and can still use control-flow attack.
- Secure or Verified Boot: Only load code with valid signature. Only approved software can be loaded. Large OS almost certainly has a vulnerability, only one component needs to be compromised. SW or certificate revocation requires state (rollback to expired version attack)
- Virtual-machine-based Isolation: Isolate applications by executing them inside VMs. VMM smaller than OS assumed to be secure, smaller TCB, Isolation between applications. But VMM usually still large and part of TCB, complicates interaction between applications.
- Approach: Achieve high security for small subset of system.

Attestation for SW Integrity:
- Attestation enables veriier V to verify what software is executing on untrusted device.

General Approach: Three-step approach:
1. Establis isolated execution environment
2. Externally validate correctness of execution environment
3. Autonomous launch and operation of execution environment

Three core Mechanisms:
- Isolated execution Hardware ensures partition
- Remote attestation External validation
- Sealed storage, Enable secure local execution and fetching of secret data after local root of trust is set up.

Adversary Model:
- Remote adversary launches network-based attacks (can compromise OS and applications and can control network communication)
- Local Hardware (local hardware assumed to be trusted)
- Realistic model, as remote attacks constitute majority of threats.

7.1 Trusted Platform Module (TPM) Overview

Core TPM Goals:
Platform identity
Remote attestation: Remote verifier can validate platform configuration (BIOS, OS, runnign apps)
Sealed storage: Bind a secret to a specific platform configuration.
Secure counter
TPM is passive device that offers basic functionality to platform
TPM will store measurements (hashes) that represent platform configuration in Platform Configuaration Registers PCR
How to store long list of hashes securely sl.22 ff

Cryptographic Hash Function Properties:
- one-wayness: given h(x) cannot find x' s.t. H(x') = H(x)
- Wek collision resistance: Given x, cannot find x' s.t. H(x) = H(x')
- Strong collision resistance: cannot find any x and x' s.t. H(x) = H(x')

Basic TPM Functions sl.26

7.2 Attested Boot

TCG 1.1-Style Attestation
- Approach: Measurement of all executed software and configuration files defines platform configuration.
- Desired property: Remote verifier can verify platform state by inspecting all measurements or hashes.

Integrity measurement Architecture (IMA)

TCG 1.1 Use Cases:
- Securing corporate network access: corporation wants to ensure that all systems that connect to internal network have correct software.
- Secure online banking: User wants to ensure that her platform is malware- free before logging in to bank account.
- Secure cloud computing: verify cloud environment before executing code.

Shortcomings of TCG 1.1-Style Attestation:
- Integrity measurements are done at load-time not at run-time.
- Coarse-grained, measures entire system
- No guarantee of execution.

7.3 Dynamic Root of Trust

- Approach: Special CPU instruction creates isolated execution environment (IEE)
- Desired property: Remote verifier obtains strong assurance of code execution, achieve run-time attestation.

DRTM Computing Primitives
- Create Isolated Execution Environment (IEE)
- Remote verification/attestation of IEE
- Establish secure channel into IEE
- Externally verify that output O was generated by executing code S on input I prtected by IEE.
- Mehr infos slides 45. ff

Late Launch / DRTM
- Security properties similar to reboot, without a reboot. Removes many things from TCB.
- Allows execution of high-assurance software next to existing legacy OS, without the need to trust OS

7.4 Discussion

Secure Channel Establishement Protocols:
- Goal: Local Party wants to establish a secure channel to a IEE executing on a remote host.
- Challenges: OS on remote host is untrusted, IEE needs to be invoked several times, as OS is handling network comm., OS keeps state for IEE.
- Assumptions: Local party knows correct public key of remote host, host hardware is not compromised, strong random numbers are available to IEE.

The cuckoo attack

7.5 Software-Only Root of Trust

Achieve a Dynamic Root of Trust without HW support

Initial Setting:
- Untrusted device D, trusted verifier V
- V knows expected memory contents of D
- V wants to obtain proof of D's memory contents, obtain memory integrity
- D executes verification function VF
- Problem: malicious VF returns expected result.

Approaches:
- Reflection: Fill entire memory randomly, get checksum of entire memory.
- Genuinity: verificatio function uses randomized memory access to cause unpredictable cache misses
- Alien vs. Quine: Implement a secure loader program.
- SWATT: Software based ATTestation for embedded devices. Verification function is a pseudorandom memory traverssal to compute memory checksum. Verifier times checksum computtaion and verifies checksum.

7.5.1 SWATT

- Ned to make sure that there is not a faster way to compute checksum on the device.
- Need to make sure that response originates from the device.
- Checksum computed over entire device memory. Does not scale to large memory sizes. Solution: design checksum function that can check small memory areas (Memory area being checked includes checksum function), but introduces many

Attack on Partial Memory Verification:

Challenges on x86 Platforms:
- Out of Order execution, cache and virtual memory.
- Complex instruction set and architecture how can we ensure code is optimal?
- DMA-based attacks from malicious peripherals
- interrupt-based attacks.

7.5.2 Pioneer

Goal: provide verifier with guarantee about what code execued on device.
1. Verify code integrity through SW-only root-of-trust attestation.
2. Set up untampered code execution environment
3. Execute code

7.6 Potential Attacks

Exceptions and Interrupts:
- Attacker installs malicious exception and interrupt handlers
- Attacker generates exception or interrupt during execution of Verification Function
- Solution: Replace interrupt and exception handlers, but when?

7.6.1 Sensor Network Key Establishment

Given nodes in a sensor network, how can any pair of nodes establish a shared secret without any prior authentic or secret information? In theory this is impossible because of active MitM attack.

ICE Key Establishment:
- leverage ICE to compute checksum faster than any other node, and use checksum as a short-lived shared secret.

8 IoT Security

Conventional Security mechanisms don't work well because:
- Constrained capabilities and resources
- Diverse communication technologies

Information Leakage: Event-driven communication of the devices leads to the possibility to infer information about what is going on (e.g. user is walking from one point to other)
1. Existential Leakage: Occurs when the transmission of a single message implies a real-world event (e.g. a car enters a parking spot and this is reported by an Occupancy sensor)
   Can not be fully eliminated.
2. Statistical Leakage: Occurs when a deviation from normal implies a real-world event. (e.g a company event leads to more activity on the parking lot.)
   can be eliminated.

Other Information Leakage despite encryption:
- VoIP Phonemes analysis of encrypted transmission data.
- D

8.1 Case: Study Device Pairing