ECE/CS230
Computer Systems Security

Charalambos (Harrys) Konstantinou

https://sites.google.com/view/ececs230kaust

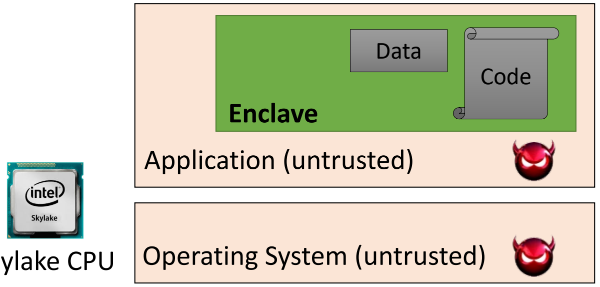
TEEs

Trusted Execution Environment (TEE)

- Hardware technologies for trusted computing
  - Isolated execution: integrity of code, confidentiality
  - To protect application from untrusted platform
- Practical limitations of TEEs
  - Trusted Platform Module (TPM) : Poor performance
  - ARM TrustZone : Compatibility (only for embedded devices)

Intel SGX

- An extension of x86 Instruction Set Architecture (ISA)
  - Offers native performance, Compatibility with x86
  - Application keeps its data/code inside the “**enclave**”



Data

Code

Enclave

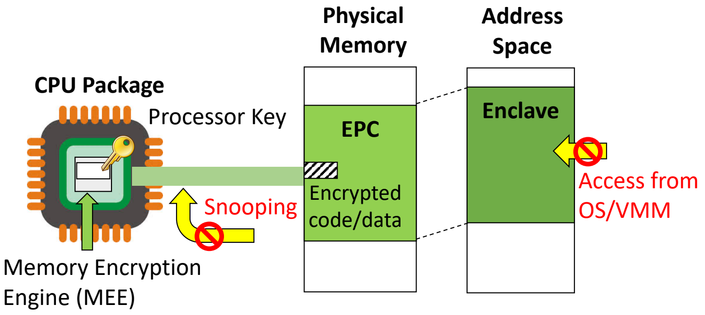
Application (untrusted)

Operating System (untrusted)

Skylake CPU

Intel SGX 101: Isolated Execution

- Smallest attack surface by reducing TCB (App + processor)
- Protect app’s secret from untrusted privilege software



Physical Memory

Address Space

CPU Package

Processor Key

EPC

Encrypted code/data

Enclave

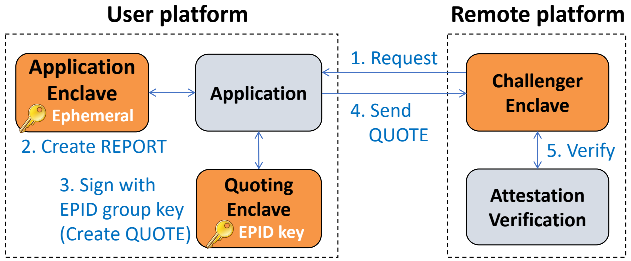
Snooping

Access from OS/VMM

Memory Encryption Engine (MEE)

Intel SGX 101: Remote attestation

- Attest an application on remote platform
  - Check the **integrity of enclave** (hash of code/data pages)
  - Verify whether **enclave is running on real SGX CPU**
  - Can establish a **“***secure channel***”** between enclaves



Least privilege in popular OSs

- Pretty poor
- Windows pre-NT: any user process can do anything
- Windows pre-Vista: fine-grained access control. However, in practice, many users just ran as administrators, which can do anything
  - Some applications even required it
- Windows Vista
  - Easier for users to temporarily acquire additional access rights (“User Account Control”)
  - Integrity levels, e.g., Internet Explorer is running at lowest integrity level, which prevents it from writing up and overwriting all a user’s files

Least privilege in popular OSs (cont.)

- Traditional UNIX: a root process has access to anything, a user process has full access to user’s data
- SELinux and AppArmor provide Mandatory Access Control (MAC) for Linux, which allows the implementation of least privilege
  - No more root user
  - Support both confidentiality and integrity
  - Difficult to set up
- Other, less invasive approaches for UNIX
  - Chroot, compartmentalization, SUID (see next slides)

Chroot

- Sandbox/jail a command by changing its root directory
  - `chroot /new/root command`
- Command cannot access files outside of its jail
- Some commands/programs are difficult to run in a jail
- But there are ways to break out of the jail

Containers

- Files (as in chroot) are not the only thing you might want to isolate from one process to another
- Some OSes (e.g., Linux) support namespaces for various resources
  - process IDs, user IDs, network configuration, filesystem mounts, ...
- A container can run processes in a set of namespaces isolated from other containers on the same physical (“host”) machine
- Example container systems: lxc, docker
- Having a privilege inside a container does not imply having the privilege in other containers, or on the host machine

Compartmentalization

- Split application into parts and apply least privilege to each part
- OpenSSH splits SSH daemon into a privileged monitor and an unprivileged, jailed child
  - Confusingly, this option is called UsePrivilegeSeparation. But this is different from Separation of Privileges (see earlier)
- Child receives (maybe malicious) network data from a client and might get corrupted
- Child needs to contact monitor to get access to protected information (e.g., password file)
  - Small, well-defined interface
  - Makes it much more difficult to also corrupt monitor
- Monitor shuts down child if behavior is suspicious

setuid/suid bit

- In addition to bits denoting read, write and execute access rights, UNIX ACLs also contain an suid bit
- If suid bit is set for an executable, the executable will execute under the identity of its owner, not under the identity of the caller
  - /usr/bin/passwd belongs to root and has suid bit set
  - If a user calls /usr/bin/passwd, the program will assume the root identity and can thus update the password file
- Make sure to avoid “confused deputy” attack
  - Eve executes /usr/bin/passwd and manages to convince the program that it is Alice who is executing the program. Eve can thus change Alice’s password

Assurance

- How can we convince others to trust our OS?
- Testing
  - Can demonstrate existence of problems, but not their absence
  - Might be infeasible to test all possible inputs
  - Penetration testing: Ask outside experts to break into your OS
- Formal verification
  - Use mathematical logic to prove correctness of OS
  - Has made lots of progress recently
  - Unfortunately, OSs are probably growing faster in size than research advances
- Validation
  - Traditional software engineering methods
  - Requirements checking, design and code reviews, system testing

Evaluation

- Have trusted entity evaluate OS and certify that OS satisfies some criteria
- Two well-known sets of criteria are the "Orange Book" of the U.S. Department of Defence and the Common Criteria
- Orange Book lists several ratings, ranging from "D" (failed evaluation, no security) to "A1" (requires formal model of protection system and proof of its correctness, formal analysis of covert channels)
  - See text for others
  - Windows NT has C2 rating, but only when it is not networked and with default security settings changed
  - Most UNIXes are roughly C1

Common criteria

- Replace Orange Book, more international effort
- Have Protection Profiles, which list security threats and objectives
- Products are rated against these profiles
- Ratings range from EAL 1 (worst) to EAL 7 (best)
- Windows XP has been rated EAL 4+ for the Controlled Access Protection Profile (CAPP), which is derived from Orange Book's C2
  - Interestingly, the continuous release of security patches for Windows XP did not affect its rating
- Windows 7, Red Hat Enterprise Linux 6 were also rated EAL 4+