CS5430 – Systems Security Spring 2018

Lecture 28: Hardware-based Security

May 9, 2018 *Instructor: Eleanor Birrell*

1 The Case for Hardware-based Security

One problem that arises repeatedly in systems security is that a system needs to interact with another component running on a remote machine controlled by a different (potentially malicious) principal. In a typical client-server system, the server cannot be sure that the code it is actually interacting with is actually the intended client and not a malicious impostor client. In a system that depends on a network of independently operated machines (e.g., Tor), a user cannot ensure that the other principals it interacts with (e.g., Tor relays) actually behave as intended and don't contradict the purpose of the system (e.g., track or censor user connections). Systems that outsource computation to the cloud cannot be completely confident that the cloud provider actually performs the desired computation (and no other malicious behavior, e.g., leaking sensitive data and/or algorithms). Trusted hardware has been proposed as the solution to this class of problems.

The key feature of a trusted hardware is isolation; dedicated hardware components can have isolated storage (either volatile or non-volatile) that can be protected from external access. Trusted hardware can also have secrets (e.g., cryptographic keys) embedded in the chip; these secrets can be used enforce and attest to the confidentiality and integrity of stored values. In recent years, there has been a push towards building secure hardware components that implement these features and using this hardware as a root of trust for building new, more secure systems.

2 Trusted Platform Module (TPM)

Trusted Platform Modules or TPMs are devices that satisfy an international standard defined by the Trusted Computing Group, an industry consortium. The current standard (TPM 2.0) was released in 2014. TPMs are most commonly implemented as dedicated, special-purpose chips; TPMs are available from most major hardware vendors.

TPMs include several different components: embedded keys (or seeds for keys), Platform Configuration Registers (PCR), general-purpose volatile and non-volatile memory, an execution engine, cryptographic engines, and modules for random number generation and key generation. Discrete TPM chips are typically wrapped in tamper-resistant packaging.

The primary feature of a TPM is platform attestation. This is achieved by means of the PCRs. PCRs have an initial value that is reset when the machine is restarted.

They can be modified only by *extending* their value, that is, updating the value to be $PCR_{new} = H(PCR_{old}||m)$, where m is a measurement of the software executing on the platform. Platform attestation was intended to achieve secure boot: the first software to execute (known as the Root-of-Trust-for-Measurement) measures the second software, stores the measurement by extending the PCR, and then executes the second software (if the measurement matches a whitelisted value). The second software measure the third, extends the PCR, and then executes the third (if the measurement checks). This continues until the operating system is instantiated. Alternatively, PCRs can be used to implement *measured boot*, which extends the PCR without checking the next measurement against an authorized whitelist. Principals can subsequently check that the boot process was not compromised by asking for the signed PCR value—known as obtaining a *quote*—verifying the signature and comparing it to the expected value.

In general, TPMs are not used to attest user-level applications because the sequence of programs (and the range of possible arguments) renders measurement evaluation impossible. TPMs can, however, generate and attest to other values using platform-specific *endorsement keys* (EKs) derived from the endorsement seed.

The other key feature of a TPM is data sealing. Data can be encrypted under a TPM-derived key for secure storage; this *sealed data* can be tied to the current PCR values such that the data can only be decrypted when the PCRs contain the matching values. Sealing can be used to ensure that only the intended software can access that software's secrets. This technology is used by Bitlocker, Microsoft's full-disk encryption software, to store its encryption keys.

3 TrustZone

TrustZone is an ARM technology designed to provide hardware isolation for trusted software. Effectively, it partitions the world into Non-Secure—which includes the standard hardware, rich OS, and existing applications—and Secure—which includes hardware secure resources (keys, memory, crypto engines, etc), a trusted OS, and trusted applications. TrustZone protects secure components from non-secure components by integrating protective measures into the processor, bus fabric, and system peripherals. This measures include the addition of Non-Secure (NS) bits to control signals and cache tags. For example, hardware logic present in the modified bus fabric ensures that no Secure resources can be accessed by Non-Secure applications.

Software, like any other component, runs in either the secure world or the non-secure world. These processes are run in separate virtual cores; which world the processor is running in is indicated by the NS bit in the Secure Configuration Register. Context switching is handled by a new monitor mode; mechanisms for entering monitor mode from the non-secure world are tightly controlled, e.g., dedicated instructions.

TrustZone enable devices can implement secure boot by storing the public key of the trusted vendor (or a hash of the key) in one-time-programmable hardware (e.g., poly-silicon fuses) during manufacture. A TrustZone enabled processor always starts in the Secure world; secure boot proceeds by iteratively checking the signature of each bootloader until the trusted OS is running, and then it starts up the normal OS.

Secure systems can be split into trusted and untrusted components that communicate using the TrustZone API. However, only signed and verified components can run in the secure world. Signed binaries are verified using a X.509 signing certificate hierarchy. Since there is no hardware-enforced isolation between trusted components, signing keys are strictly controlled. This limits the practical use of TrustZone technology to develop secure systems.

Although TrustZone is designed to run secure applications at any time, the most well-known example of TrustZone in use is probably Android's full disk encryption system. This system relies on a RSA signing key generated by the KeyMaster module, an application that runs in the secure world; this signing key is only available in the non-secure world when encrypted under a hardware-backed encryption key. The disk is stored encrypted under a randomly chosen 128-bit decryption key, this decryption key is stored under a PBE key derived from the user PIN. The PBE scheme used is a form of scrypt modified to incorporate an RSA signature during an intermediate state; this modification is designed to tie decryption of the decryption key (and thus decryption of the disk) to both the user's PIN and the physical hardware device.

4 Secure Guard Extension (SGX)

Intel's Secure Guard Extension (SGX) is a set of extensions to the Intel instruction architecture designed to enable a trusted execution environment. SGX enables the construction of a secure container called an *enclave*; enclaves are isolated and support both sealing and remote attestation.

Isolation is enforced by introducing a subset of memory called *processor reserved memory* that is accessible only to Intel enclaves; this address range includes an enclave page cache. Pages in the enclave page cache are associated with a particular enclave and can only be accessed by that enclave.

SGX enclaves are defined by their SGX Enclave Control Structure. This structure is used to implement the isolation features described above. It can also be used to produce an identifier or *measurement* for the enclave. This measurement is used by the key generation function (along with secrets embedded in the hardware) to derive keys, including sealing keys. Since sealing keys can only be re-derived if both the enclave attributes and the hardware secrets match the values at the time the key was originally derived, this mechanism enables enclave-based sealing. Certain enclave attributes can optionally be masked, so for example a key might either be unique to a particular enclave or might be derivable by an enclave with the same author.

SGX, unlike other hardware security tools, also enables remote attestation. Local attestation (between enclaves) can be achieved by using the EREPORT instruction, which produces a signed (HMAC'd) copy of the enclave measurement. Since the (symmetric) signing key can be constructed by any enclave running on the same hardware, the enclave measurement can be verified by other enclaves. This is extended to remote attestation using a pair of Intel-defined enclaves: the provisioning enclave and the quoting enclave. The provisioning enclave requests an attestation key from Intel and stores it sealed under a key that can only be derived by Intel-authored enclaves. The quoting enclave retrieves the attestation key, verifies the measurement using local attestation, and signs the measurement; the resulting signed measurement, called a *quote* can be verified using Intel's Attestation Service. Replay attacks are prevented by careful use of nonces.

Use of SGX in practice is limited by Intel's key management; enclave launch in production mode requires a production key controlled by Intel, and remote attestation (quote verification) requires a request to Intel. This ensures that Intel controls when and how enclaves are used in real-world systems and is in-the-loop for all such systems. This approach has naturally deterred some companies from depending on SGX technology.

Since SGX-enabled hardware has only been released in the last year and since some companies are reluctant to depend on a technology so closely tied to Intel, SGX is not currently in use in any major systems. However, prototype systems applying SGX to solve problems related to cloud computing, TOR networks, smart contracts, and many other applications have been designed and prototyped.

5 Vulnerabilities of Hardware-based Security

Secure hardware offers the potential to build new secure systems using that hardware as a root of trust. However, these hardware tools are not magical solutions to all security problems. In particular, hardware-based security is subject to two classes of vulnerabilities: untrustworthy trusted code and side channels. A summary of the vulnerabilities of the various secure hardware solutions is given in Table 1.

If trusted code, for example an application running in TrustZone's secure world or inside an SGX enclave, contains bugs, these can introduce vulnerabilities that undermine the security of the system. For example, a recent version of Android's full-disk encryption system was compromised by a privilege-escalation vulnerability in Qualcomm's TrustZone kernel. In the case of TrustZone, a malicious (or coerced) original equipment manufacturer (OEM) could sign a Trusted application that attempts to extract secrets from the secure world; hardware isolation is designed to protect the secure world from components in the non-secure world, such attacks would bypass TrustZone's hardware-based isolation.

Trusted execution environments like TrustZone's secure world and SGX enclaves are also potentially vulnerable to side-channel attacks, including page fault attacks and cache attacks. Such attacks have been demonstrated in research contexts, but there feasibility in the real-world is not yet well understood.

Adversary	Attack	TPM	TrustZone	SGX
OS	direct probing	n/a	access checks on TLB misses	Access checks on TLB misses
OS	page faults	n/a	secure world page tables	X
OS	cache timing	n/a	X	X
Another container	direct probing	n/a	n/a (secure world trusted)	access checks on TLB misses
Another container	cache timing	n/a	n/a (secure world trusted)	X
Peripheral	DMA	X	bus bounces accesses	IOMMU rejects DMA
Physical attacker	Physical	X	n/a (on-chip SRAM only)	memory encryption engine

Table 1: A summary of the vulnerabilities of current trusted hardware.