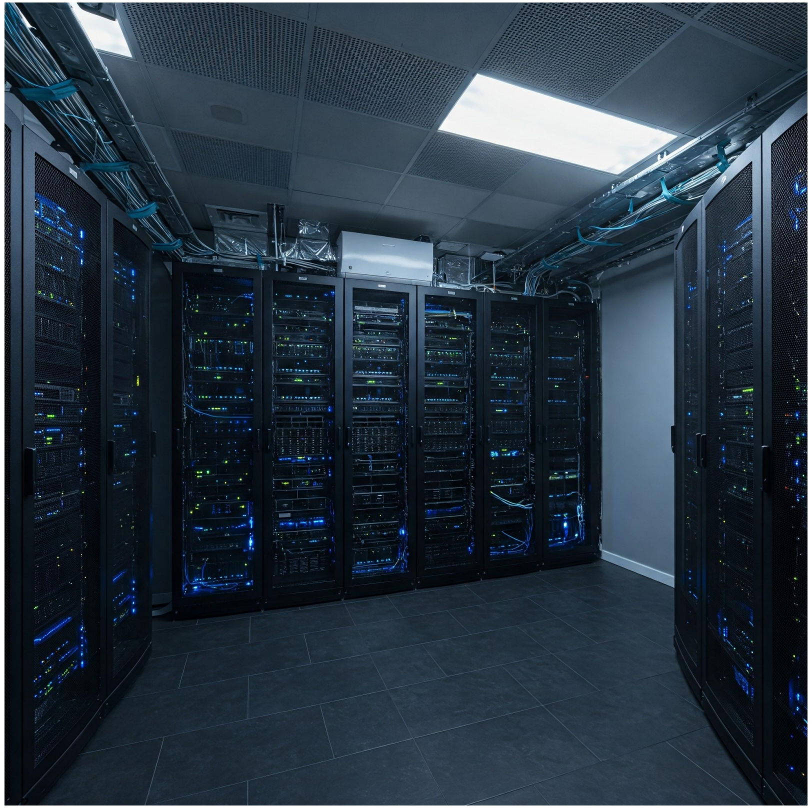
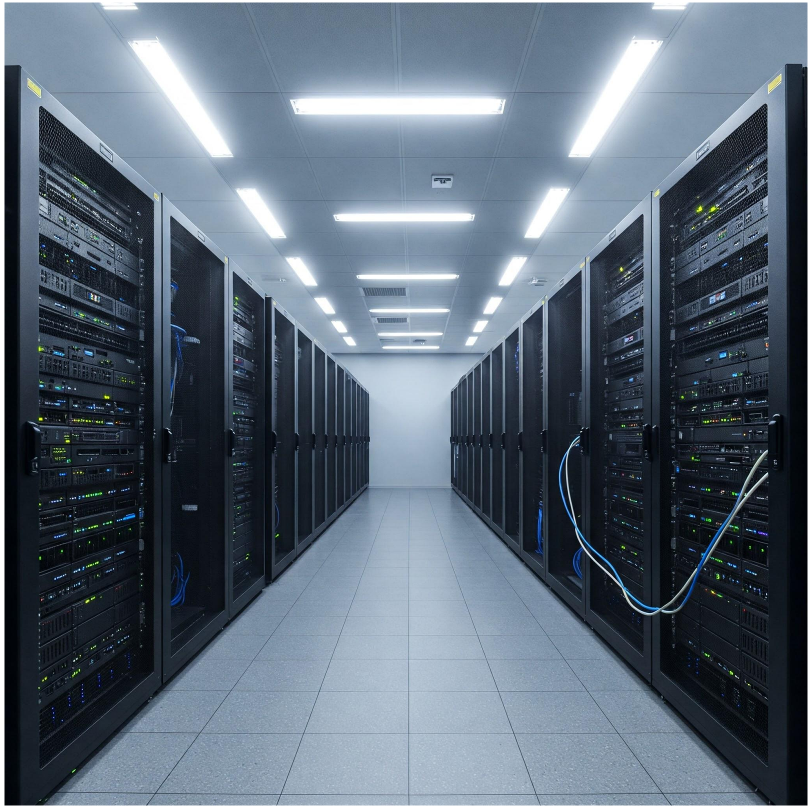
Security at (Hyper) Scale

Challenges and Opportunities

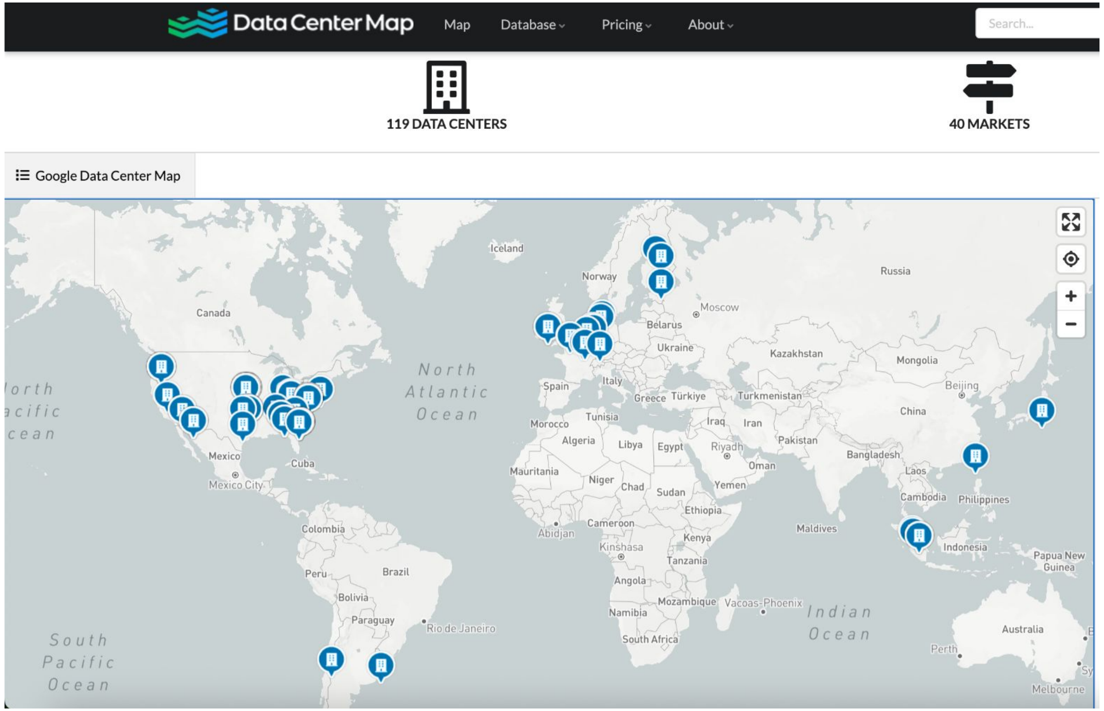
CS-115 Invited Lecture

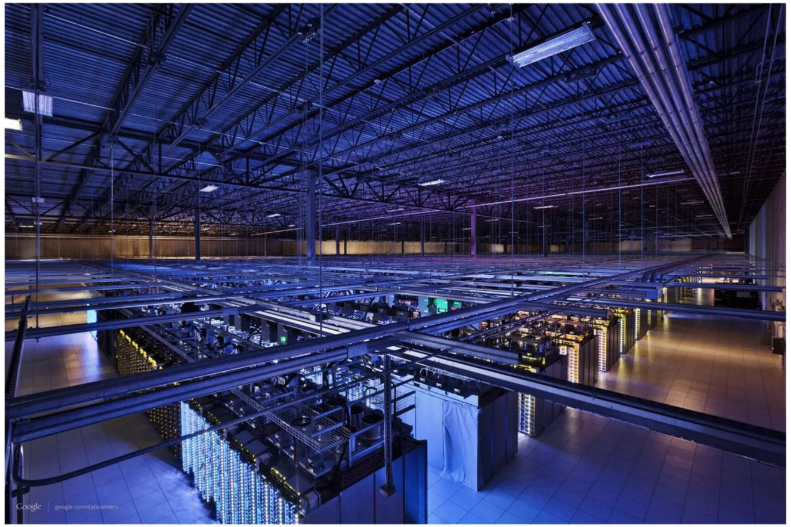
Not this….





This..

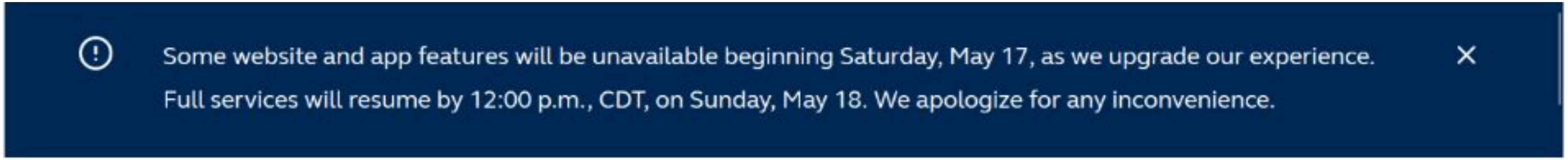


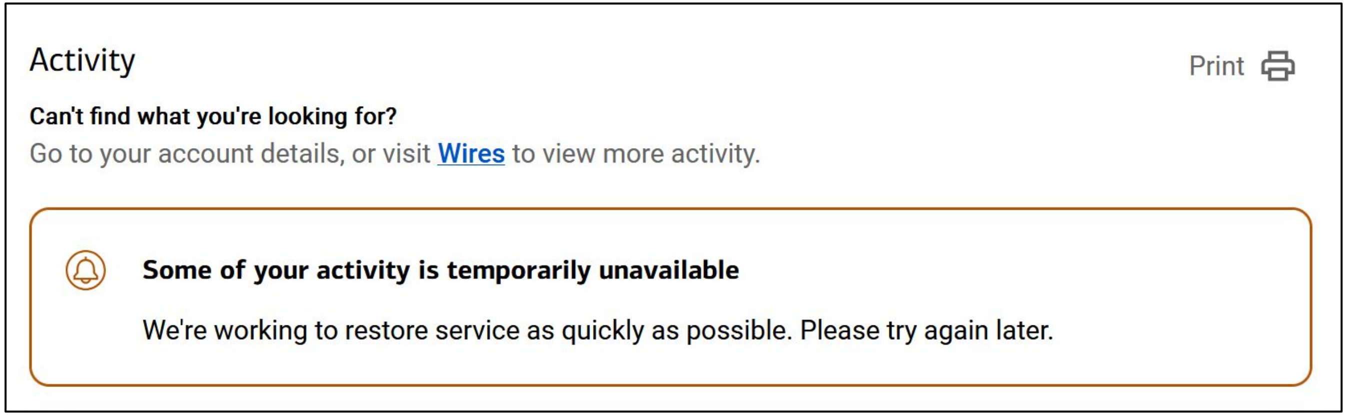


One 30MWatt server hall in a multi-GigaWatt datacenter

Source: https://www.datacentermap.com/c/google/datacenters/

For high usage consumer services: Not this…


Some website and app features will be unavailable beginning Saturday, May 17, as we upgrade our experience. Full services will resume by 12:00 p.m., CDT, on Sunday, May 18. We apologize for any inconvenience.


Activity

Print

Can't find what you're looking for?

Go to your account details, or visit Wires to view more activity.

Some of your activity is temporarily unavailable

We're working to restore service as quickly as possible. Please try again later.

This..

What’s unique about the environment

Foundational Design Criteria

- Resilience/remediation of failures without visible application level disruption
  - Application level outage SLOs << mean time to failure of individual systems, components.
  - Application SLOs are set based on SLOs of hardware and software dependencies
    - Foundational services/dependencies have extremely high reliability needs
  - **Invert:** Unused outage SLOs is a **valuable resource** to proactively clean up systems, disaster testing etc.

- Failures/Incidents
  - 6+sigma events and failures will occur and be noticed
  - Even hardware manufacturers have not tested their equipment at this scale
  - Your testing will always be incomplete, your changes will often cause failures

Foundational Design Criteria (cont)

- Core security services may operate at extremely high QPS
    - E.g, authorization checks operate at O(10 Billions)/s
    - Broad based failure during an update will quickly bring down large number of services

- Extremely difficult to justify fast, global changes
    - Fast global change -> high risk of outage across multiple services
    - Slow changes is the norm
        - Multiple ongoing rolling changes to the infrastructure at any point in time
    - **Design criteria:** Security must operate in a rolling change environment and deal with not tripping over dependencies between changes

Foundational Design Criteria (cont.)

- Extremely good visibility, measurements and telemetry essential
  - Entire teams devoted to this
- Need to operate with extremely high machine to human ratio
  - Automation, Automation, …..
  - Powerful fleet level management tooling, ….,
  - Avoid need for physical human intervention as much as possible
  - **Conversely:** your budget for requiring human intervention is capped by human availability bandwidth.
    - E.g., requiring human access to a server in a datacenter.

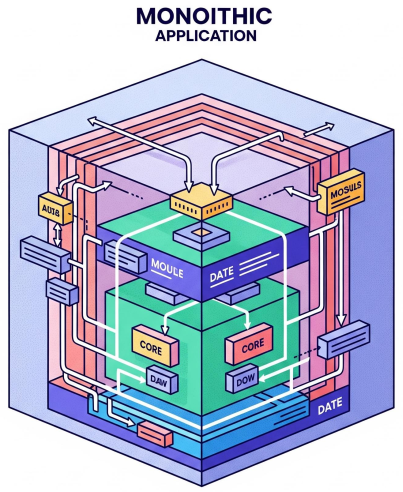
Interesting Hyperscale Security Challenges

- Communication Security, Workload Authentication/Authorization
- Key Management
- Maintaining infrastructure integrity
- Access Management
- Application Security
- Monitoring, Detection and Response

Workload Communication Security, Authentication/Authorization

Background

- Workloads: Microservices win over Monolithic Applications
  - Scaling out with standard hardware, intelligent load balancing, handling system failures and migration



VS

MICROSERVICES
based APPLICATION

Hyperscale software environment

- Huge number of microservices running on a large distributed infrastructure
- Extremely high #RPCs/second in the infrastructure
  - RPCs need communication security, authentication, authorization.
- Loose binding between services endpoints and actual physical machines and network identities
  - Autoscaling, load balancing
- Can't depend on network layer for communication security,
  - Huge TCB: Huge number and diversity of networking devices, complex network control planes
  - Many network level attacks (covered in lectures 15/16: Internet protocols/Security)
  - But **could provide another layer for communication security**, segmentation etc if no performance impact
- Network security still needed for availability
  - **Internet protocols/Internet Security Lectures** (DNS, BGP, DDoS protections, Firewalls, Network level intrusion detection)

Concepts

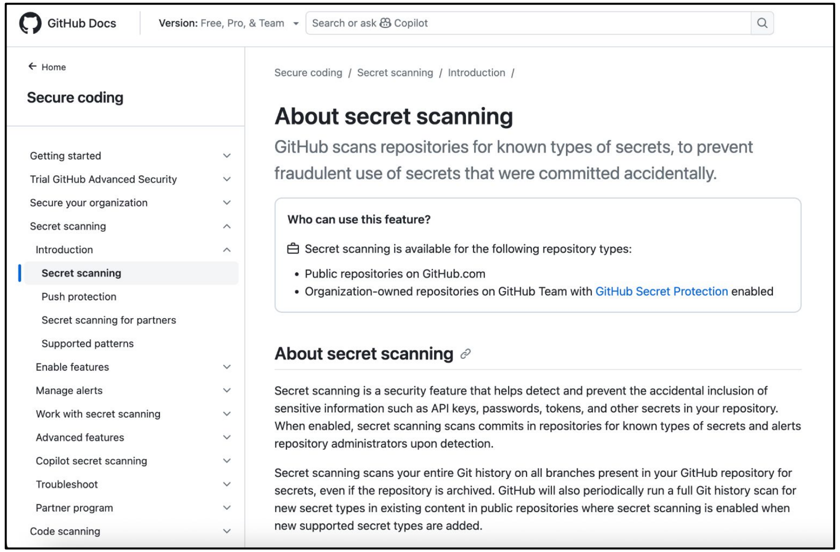
- Communication security @scale
  - Private PKI with subjects being (RPC-capable) service identities, machines, control/management services etc.
  - Service to service RPCs require confidentiality, integrity, mutual authentication and authorization.
  - Authentication/Communication security protocol: ALTS (Application Layer Transport Security) or mTLS
    - mTLS needs X.509 certs with service identity not hostname or IP
    - SPIFEE in Subject Alternative Name (SAN)
  - Each endpoint authenticates the peer in incoming RPCs and performs authorization check based on authenticated peer identity.
    - **Zero Trust** – all RPCs needs authentication and authorization based on peer identity, no implicit trust based on being in same machine, cluster, network segment etc.

For more information on ALTS: see https://en.wikipedia.org/wiki/ALTS

Bootstrapping service credentials

How services get credentials/keys ?

- Not left to app developers – keys may be left unprotected and compromised (e.g., GitHub).
- Keys (securely) pushed to workloads as part of dispatching workloads to the machine by the scheduler
    - **Protected by keys tied to machine level keys that are released by a Root-of-Trust hardware upon secure boot of the machine**
- Compromise of a machine only give attackers keys of workloads on that machine, no other identity or trust relationship beyond that.



About secret scanning

GitHub scans repositories for known types of secrets, to prevent fraudulent use of secrets that were committed accidentally.

Who can use this feature?

Secret scanning is available for the following repository types:

- Public repositories on GitHub.com
- Organization-owned repositories on GitHub Team with GitHub Secret Protection enabled

About secret scanning

Secret scanning is a security feature that helps detect and prevent the accidental inclusion of sensitive information such as API keys, passwords, tokens, and other secrets in your repository. When enabled, secret scanning scans commits in repositories for known types of secrets and alerts repository administrators upon detection.

Secret scanning scans your entire Git history on all branches present in your GitHub repository for secrets, even if the repository is archived. GitHub will also periodically run a full Git history scan for new secret types in existing content in public repositories where secret scanning is enabled when new supported secret types are added.

Managing Keys at Scale

Challenges with Key Management @ Scale

- Some hard truths in a large scale environment
  - Huge numbers of keys used by a huge number of applications/microservices for a large number of purposes.
  - **Likely impossible to keep all keys secure**
    - Some application keys will get inadvertently leaked, e.g., bugs, core dumps, logs
    - Some applications will get compromised from attacks and expose the keys before detection/response.
      - Example: Lecture 1 - control hijacking attacks, Lecture 7: microarchitecture attacks
    - Only a few keys can be very strongly protected (e.g., offline in secure hardware in safes in multiple locations with multi-party human controls to activate to perform certain operations).
    - In some situations losing a key is worse than leaking a key
      - E.g, Storage encryption key, Key-encrypting key etc.
  - **Relying on application developers to securely handle key material is a bad idea.**
  - **Relying on training application developers about secure and insecure ways to perform cryptography is a bad idea.**

Handling crypto related challenges

- Unburden developers from cryptography pitfalls and complexity
- Cryptography expertise is limited so:
  - Have a highly opinionated cryptographic library (e.g, Tink) for developers, that provides a safe and limited interface to a small set of commonly needed cryptographic operations
    - Implemented correctly by experts
  - **Expert Reviewers** can then focus on new protocols or uncommon cryptographic constructs to make sure they are secure
    - Refer back to Lecture 11: Brief overview of Cryptography: BREACH attack.
- Handling and securing key material used by applications
  - A key management system (KMS) that stores key material securely, manages rotations, lifecycle and access and delivers them to the applications that need them
    - Needed keys are delivered to workload using secure communications based on Workload identity and credentials
      - **Actually delivered to the linked opinionated cryptographic library**

Tink reference: https://developers.google.com/tink/what-is

Assuming inevitability of key exposure

- Traditional key-usage
  - (Relatively) long-lived keys, certs, credentials, long validity dates
  - Revocation capability upon known exposure incident
- What tends to happen
  - Dealing with key and credential expiry not on developers top of mind
  - Changing keys and credentials is hard and potentially disruptive, often public keys are baked in hardware with no way to update
  - Pressure to towards having longer and longer validity
  - Key exposure may remain unnoticed
    - Aligns with common attacker incentive to not get noticed.
  - When incident happens many services and systems are unprepared.
    - Fire drill: All hands on deck to revoke keys
      - Applications break in unexpected ways when keys or credentials updated.
      - Hardware may never fully recover from key compromise
- Mitigating key exposure is about building muscle, practice and playbooks for rotating/changing keys across the infrastructure

Mitigations for key exposure

- Regular key rotation and regular credential revocation/invalidation **should be part of normal operations**
  - NOTE
    - For web-PKI: short lived certificates is the way to go (Refer to lecture 12, short-lived certificates)
    - For hyperscaler private PKIs for workloads and system level credentials:
      - Time based credential expiry is highly discouraged compared to explicit invalidation
      - Why ?
- If key exposure is unnoticed, damage is time-limited as key/credentials rotate out
  - Adversary would need to maintain persistence without detection to get subsequent keys.
    - Persistence is more detectable
- In case of incident, applications and site reliability engineers are prepared, playbooks exist, few applications break when rotation/revocation performed quickly
  - Most dependencies and issues on rolling out key/credential rotations and revocations are well understood as this is practiced often.

Enabling Key Rotation: Keysets vs. Key

- **Work with well established KMSes**
- **Use Keysets, Not Single Keys:** A keyset is a list of keys with unique IDs and statuses. One key is designated as "primary."
    - Example: Primary key used for encrypting new data.
- **Handles and Blobs:** Use handles for keys and operate with cryptographically protected blobs.
- **Enables Key Management:**
    - Disable keys without removal.
    - Track which key protected specific data.
    - Transparently rotate keys and algorithms in conjunction with the KMS.
- **Long-lived Data Encryption:**
    - Option 1: Keep old data encryption keys in keyset till the data encrypted with them exists, newer data encrypted with newer keys.
    - Option 2 (much better):
        i. Data encrypted by Data Encryption key(s) (DEKs). DEKs encrypted using Key Encrypting Key(s) (KEKs) managed by KMS
        ii. Rotate KEKs and re-encrypt DEKs to avoid re-encrypting large amounts of data unless needed due to DEK exposure.

Regular rotation

1. **New Key Addition:**
   A new key is added to the existing keyset, but it's not immediately promoted to be the primary key (the key currently in use).
2. **Keyset Distribution:**
   The updated keyset, including the new key, is distributed to all systems or applications that use it. This ensures that all parts of the system can utilize the new key before it becomes active.
3. **Primary Key Promotion:**
   Once all applications are updated and can utilize the new key, it is promoted to be the primary key.
4. **Transition Period:**
   During the transition, **while promotion is underway,** both the old and new keys can be used for encryption and decryption, allowing for a seamless switchover.
5. **Key Retirement:**
   After the new key is fully adopted, the old key can be retired or removed from the keyset.

PRACTICAL NOTE: Mandatory telemetry and monitoring of applications is critical to know for certain that all applications are updated. In practice, there will be some binaries that have stopped providing telemetry – they may be malfunctioning for example and in a microservices environment, terminating them would be OK.

Emergency revocation

- An emergency revocation or rotation is designed similarly, but moves faster to reach key retirement, and focuses on the compromised keys
    - More tolerance for minor failures in case new keys not fully deployed
- For some public keys that are used by limited number of applications it may be easier to push out a faster and explicit global revocation flow to prevent usage of a compromised private key
    - E.g., RPC Authentication keys used by services originating from one compromised machine.
    - Care needs to be exercised that this revocation does not revoke anything more that this one key/credential – otherwise can result in widespread outage.
- A risk based decision still needs to be made – wait till next rotation or perform emergency revocation that increases likelihood of having failures.

Maintaining Infrastructure Integrity

Inevitability of (Security) Failures

Security Folklore – A secure computer

- No such thing as a secure and *working* computer, component or device
- Given sufficient time, effort, cost *ALL computing systems, devices or components can be attacked and broken.
- "*Attacks always get easier, they never get harder*" - Security Folklore

Attacks keep coming, they only get easier…

- Lecture 2: Control hijacking attacks, exploits
- Lecture 7:Processor and microarchitecture security: Intel TDX and the Spectre attack
- Lecture 9: Web Attacks
- Lecture 12: HTTPs Goals and pitfalls
- Lecture 13: Security of AI systems - Multi-modal prompt injection
- Lecture 15/16: Internet Protocols/Security
- Lecture 17: DoS attacks

How to deal with this constant incoming stream of vulnerabilities, attacks etc ?

Key concepts

- Principle of multiple security layers
- Workload trustworthiness and workload security rings
- Machine Integrity
  - Challenges and solutions in securing modern system bootstrap and Root-of-Trust(s)
- Constant vigilance to maintain integrity during operations
  - *Red teaming, Vulnerability management, etc.*
  - *Monitoring of machine and infrastructure state*
  - Detection/Response

Principle of Multiple Security Layers

- Ensure that attackers have to compromise multiple independent security protections (or layers) to achieve their goal
  - No-one layer is invincible
  - Maximize chances of detection and response to a multi-step/stage attack
  - Rolling out a security fix or mitigation to a vulnerability at a layer will take time – additional layers buy valuable time
  - Refer to “Defense in Depth” in Lecture 4
- Example:
  - VMM, Sandboxes, hardened OS, Zero-trust across machines, **Workload Security Rings,**
  - Refer to Lecture 5: Isolation and sandboxes

Dealing with varying Workload Trustworthiness

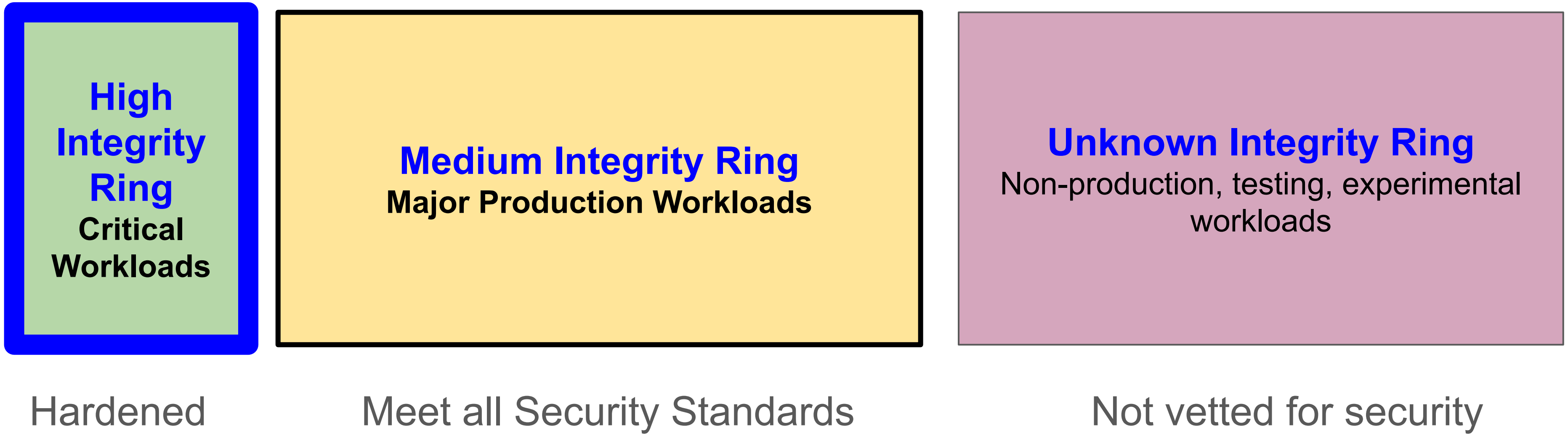
- Many external and internal attacks on applications will start by an attacker getting a foothold in some running workload and privilege escalate to the desired target

- Unwise to expect all internal workloads to be trustworthy or to try to make them so:
    - “*For should the enemy strengthen his van, he will weaken his rear; should he strengthen his rear, he will weaken his van; should he strengthen his left, he will weaken his right; should he strengthen his right, he will weaken his left.* ***If he sends reinforcements everywhere, he will everywhere be weak”*** - Sun Tzu
    - Better to assume the full range of trustworthiness exists amongst workloads

- Dealing with varying workload trustworthiness
    - Figure out what really needs to be protected and what compromises are acceptable:
    - *"Those generals who have had but little experience attempt to protect every point, while those who are better acquainted with their profession, having only the capital object in view, guard against a decisive blow, and acquiesce in small misfortunes to avoid greater."* - Frederick the Great.

Example: Workload Security Rings

INFRASTRUCTURE PARTITIONS



* In addition to Zero trust: Application level mutual Authentication/Authorization for RPC. Security Standards and API hardening for higher integrity rings.

https://www.usenix.org/publications/loginonline/workload-security-rings

Machine Integrity: Securing Modern System Bootstrap and Root-of-Trust(s)

Securing System Bootstrap (TCG Traditional View)

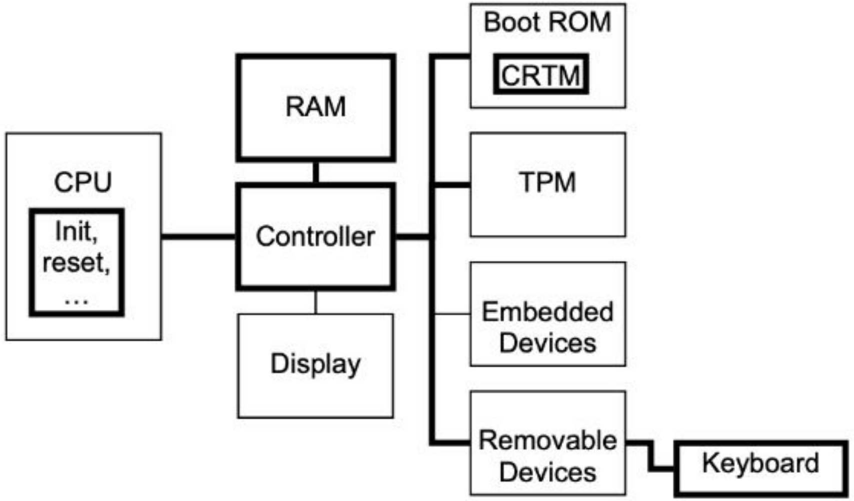


Figure 4:b – Bold indicates part of Trusted Building Block components of a trusted platform

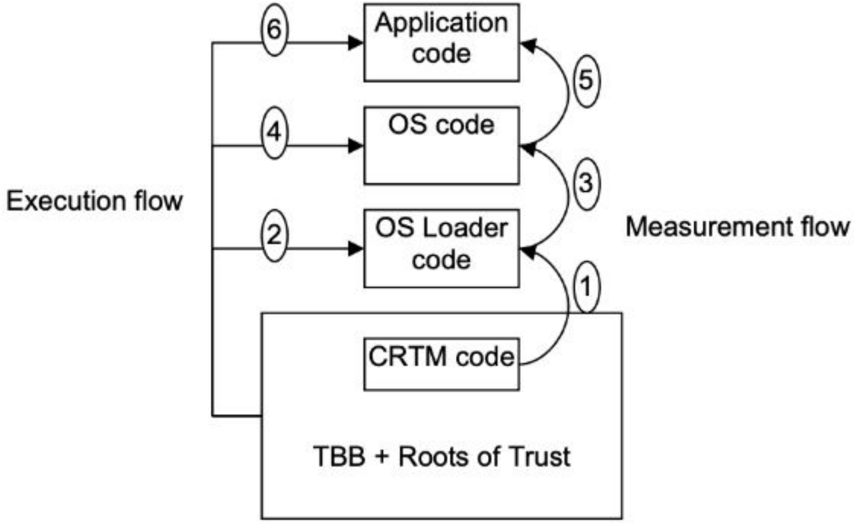
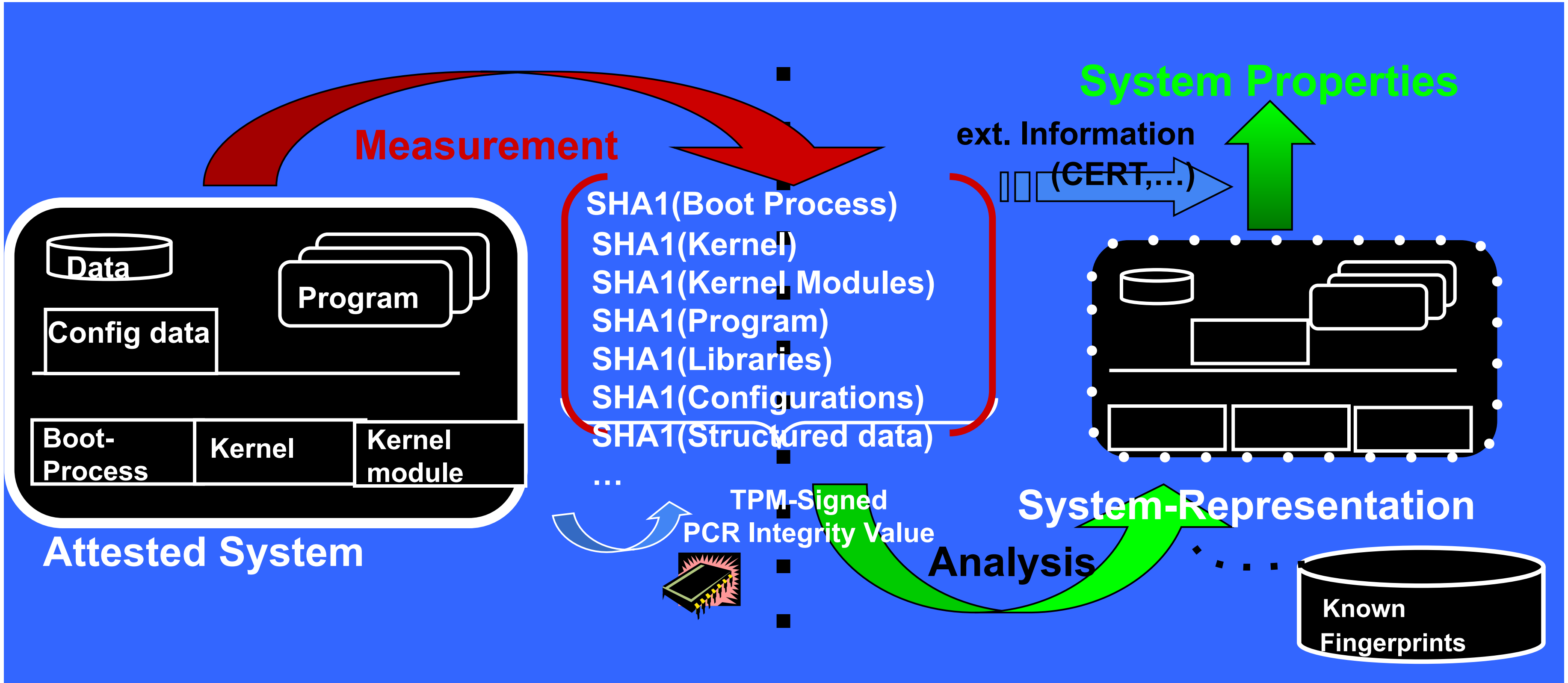


Figure 4:c – Transitive trust applied to system boot from a static root of trust.

Integrity Measurement Architecture



- Available since Linux 2.6.30

Example: Rootkit compromise example

```
                  Measurement  List         |    Fingerprint DB
==============================================+===============
000 : D6DC07881A7EFD58EB8E9184CCA723AF4212D3DB |    boot_aggregate
001 : 84ABD2960414CA4A448E0D2C9364B4E1725BDA4F |    init
002 : 194D956F288B36FB46E46A124E59D466DE7C73B6 |    ld-2.3.2.so
003 : 7DF33561E2A467A87CDD4BB8F68880517D3CAECB |    libc-2.3.2.so
...                                            |    ...
110 : F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2 |    syslogd
...
```

(a) THE GOOD CASE

```
...
110 : F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2 |    syslogd
... :                                          |    ...
525 : 4CA3918834E48694187F5A4DAB4EECD540AA8EA2 |    syslogd-LRK5
...
```

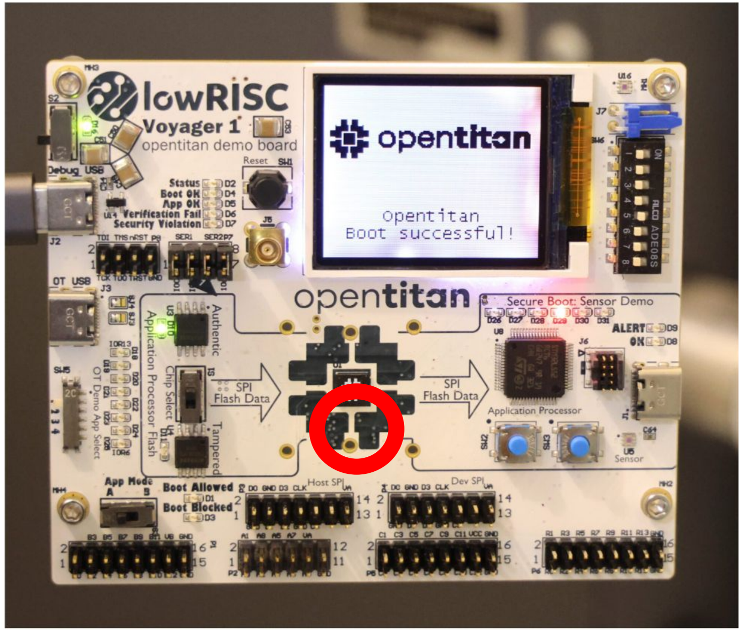
(b) LRK5-COMPROMISED SYSLOGD

Challenges Applying Traditional Secure Boot @Hyper-Scale

- Large gap: what is traditionally measured vs. what needs to be trusted
  - No “Boot ROM”, but Flash
  - Large custom BIOS/UEFI blobs in flash from vendors – not measured.
    - Dynamic Root of Trust helps but not enough – e.g., SMM attacks before DTRM, CPU bugs (e.g, Spectre: Lecture 7, EntrySign)
  - CPU is just part of what needs trust, others devices and peripherals that have boot flash may be equally or more important.
    - Smart NICs, GPUs/AI-accelerators, storage:SSDs, other devices, memory …
    - In modern server - Hardware bringup, patch management of all devices, and server remote management handled by separate full featured BMCs (Baseboard Management Controller) chip, with its own boot flash.
- Secure vs. Trusted Boot: Huge pressure to automate recovery from failures
  - Hardware failure or rare race-condition failures in a large collective of complex machines more frequent than attacks.
  - Manual physical recovery impractical
    - Bricking infrastructure due to insufficient manual fixing capacity is an attack vector.

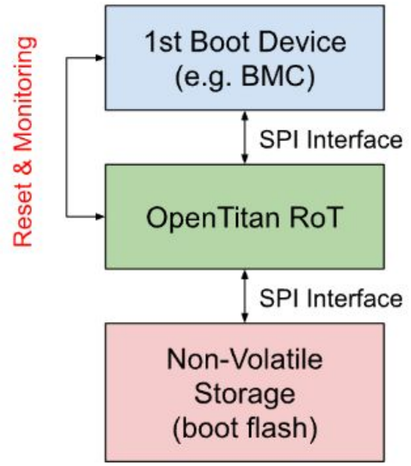
Addressing Measurement Gap

- Concepts
  - **First Instruction Integrity**
  - **Manifest of attestations**
- SOC with external flash and external Root-of-Trust
  - Example: OpenTitan (open source and commercial)
- SOC with Internal Root-of-Trust
  - Example: Caliptra (open source)

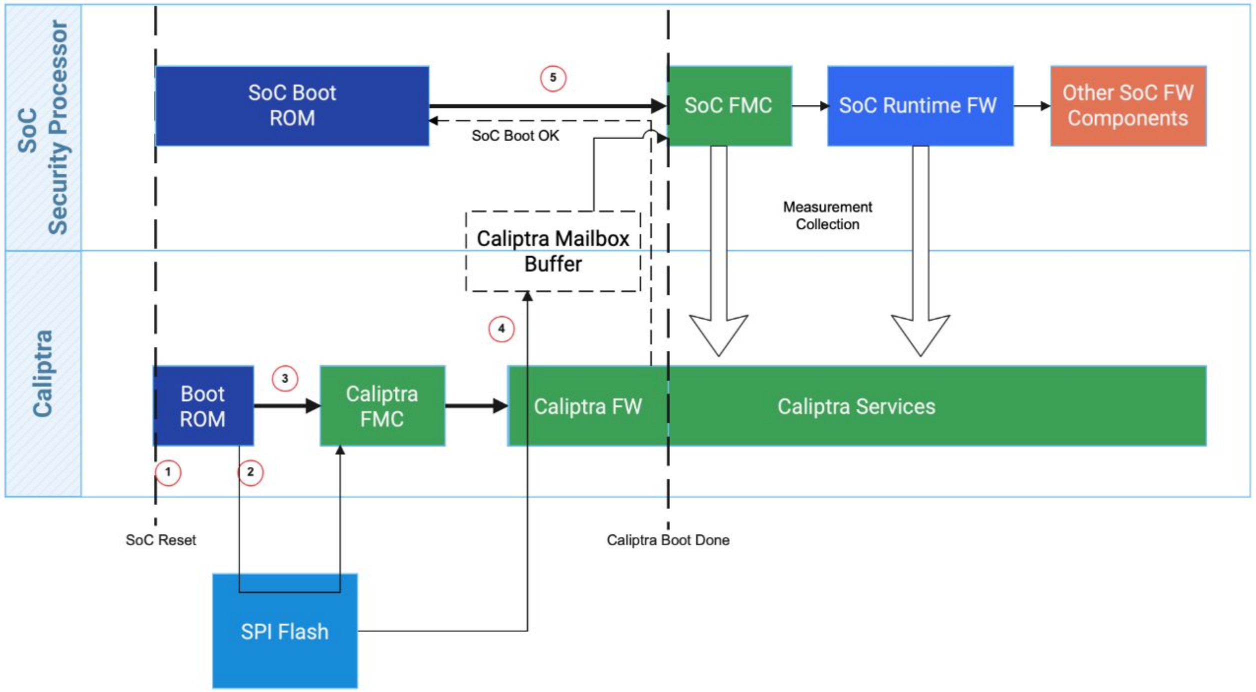


OpenTitan as Platform Integrity Module

- **Interposes** on the SPI interface between the Platform's boot flash and it main boot device (e.g., BMC)
- Implements the following security properties:
  - Measure integrity of first boot firmware stages before bringing the boot devices out of reset, accessing boot flash via SPI
  - Monitor resets and heartbeat of downstream boot devices.
  - Enforce runtime boot device access policies, and **manage A/B firmware updates for software stored in boot flash.**
  - Provides root key store and attestation flows as part of the platform integrity secure boot implementation.
  - Uses key-ladders that include secrets + its firmware measurements and hardware/OTP state to derive its keys.
    - *The presence of the correct attestation keys confirms integrity of OpenTitan and allows recovery from vulns/compromises*



Caliptra: Internal Root of Trust

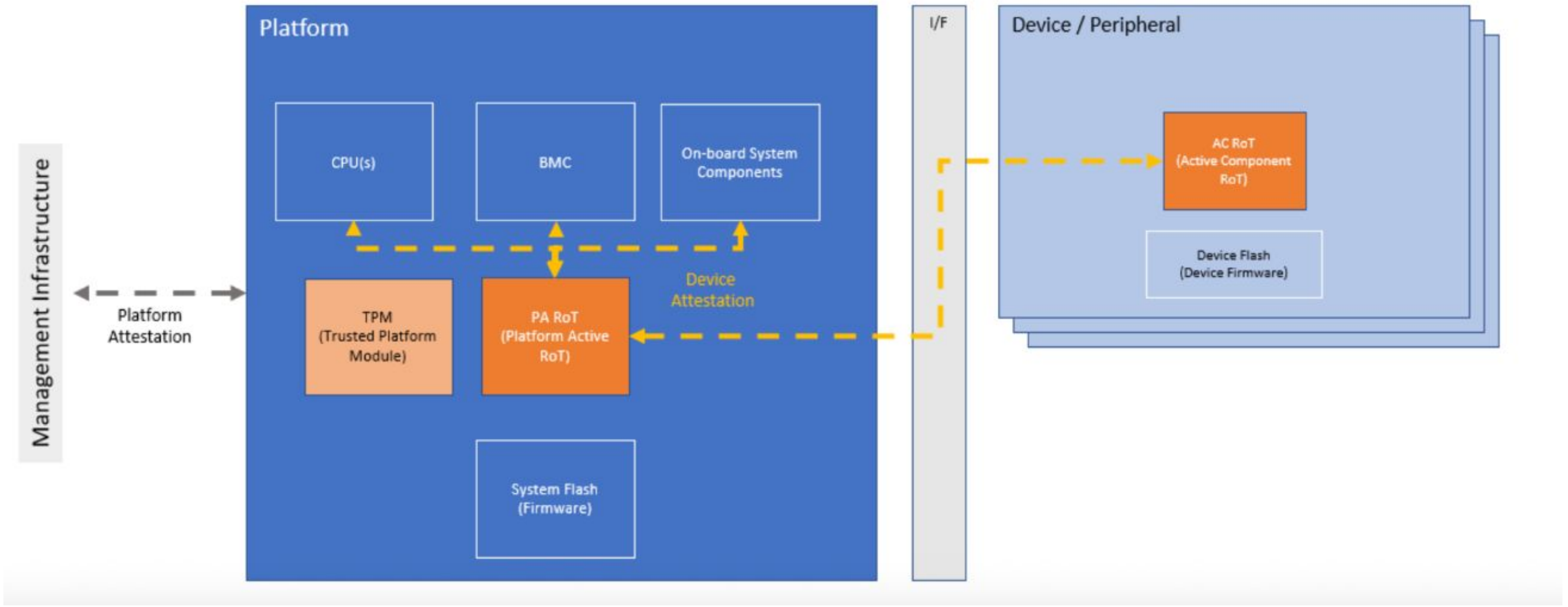


1. Hardware SOC power-on reset logic.
2. Caliptra ROM executes first and **performs cryptographic identity generation,** reads in Caliptra firmware from flash.
3. Caliptra ROM measures and verifies its firmware before loading/executing it.
4. After loading its own firmware, Caliptra copies the SOC First Mutable Code (FMC) into an SOC internal SRAM mailbox buffer and measures that firmware.
5. Caliptra signals to SOC ROM and SOC core to continue power-on reset

Caliptra cryptographic identity generation

- Based on TCG DICE (Device Identifier Composition Engine) standard
  - Designed to provide basic integrity services with minimum silicon, suitable for IoT.
- Caliptra Features
  - Unique device identity and **secret** (best effort to protect against attacks)
    - Keys derived from this are certified by manufacturer, only visible to Caliptra ROM
  - Mixed with **owner** supplied, field programmable/renewable entropy (in case of device secret is compromised) to generate the **logical device key**, certified by owner CA.
    - This key is accessible only by ROM.
  - Other keys and certs are generated based on measurement of **Caliptra First Mutable Code + other hardware parameters certified by logical device key** and once control is passed to FMC, **subsequent keys are derived based on measurements of the caliptra run-time**, **certified by the owner.**
    - Owner can update mutable firmware and update the certs.
  - **Any unauthorized modification of the Caliptra mutable code, will result in Caliptra not having the secret keys corresponding to its owner supplied certificates.**

Addressing system complexity: attestation manifests



Management Infrastructure
Platform Attestation
Platform
CPU(s)
BMC
On-board System Components
Device Attestation
TPM (Trusted Platform Module)
PA RoT (Platform Active RoT)
System Flash (Firmware)
I/F
Device / Peripheral
AC RoT (Active Component RoT)
Device Flash (Device Firmware)

- Different options on how platform is attested to management infrastructure and how updates as made based on what component(s) are considered trustworthy at different stages the system life cycle

From: Open Compute Project

Recovery from bootstrap and RoT integrity check failures

- Needs automation for hyperscale environments
- Out-of-Band management channel to BMC, accessible even when server is powered down and unable to boot.
  - BMC's own firmware/flash protected by an external Root-of-Trust that minimizes chances of BMC boot flash from getting corrupted, e.g., using A/B updates.
    - Physical access needed if the corruption happens
  - BMC has the ability to reflash other firmware/flash in the system that may become corrupted.
    - Reflash done either directly or preferably via the hardware root of trust managing that flash.
  - Doing so securely:
    - BMC can verify the integrity of what's being loaded into flash or the Hardware Root of Trust can do so.

Access Management @Scale

Recall Lecture 4: Principle of least privilege, access control, and operating systems security

Infrastructure: microservices level access/authorization checks

- Extremely high QPS – O(billions-10s of billions)/s.
- Extreme reliability/availability needs
  - Unavailability of authorization capability will quickly bring down applications
  - Mistakes in authorizations can easily cause outages.
- Handling this tradeoff
  - Separate management of authorization matrix (human time-scales and QPS) from managing permissions in infrastructure (workload access time-scales and QPS)
    - Workload identify vs. host based ACLs help
  - Push snapshot of authorization matrix to clusters everywhere and updates on a regular schedule.
  - **Failsafes in authorization updates**
    - **Don’t lock out critical components or recovery capabilities**
  - Authorization snapshot can be broadly distributed and cached
    - Reliability and availability
  - Slow rollout of updates to authorizations, canarying and monitoring of health of applications to catch any failures due to a faulty update
    - Application update rollout have to sync with authorization update rollouts

General (Human) Access Management challenges@scale

- O(millions) of ACLs, permissions etc
- Does principal X have specific access to resource Y can be a hard problem
  - Direct accesses are easy to check for.
  - Indirect, multi-step access through a chain of actions including using roles that allow modifying ACLs and permissions:
    - Theory: Harrison-Ruzzo-Ullman (HRU): determining whether a safety property holds generalized access matrix model could be undecidable
    - Heuristics are pretty good, can search up to several steps.
- Access Control Ideal: Principle of Least Privilege (lecture 4)
- Reality: 2nd Law of Thermodynamics applies to Access Control Systems
  - All access controls systems naturally move towards increasingly overprovisioned access
    - One sided (human) pressure to get work done: increase access rights till necessary work gets unblocked.
      - In practice giving/getting overly-broad permissions easiest way to unblock
      - Some resources can become progressively valuable over time.

Managing permissions @scale

- Creation of backpressure to reduce overprovisioned access towards least privilege
  - Automated provisioning/deprovisioning of permissions based on roles
    - Follows lifecycle of employment
  - Add friction on unusual/risky permission requests
    - Require some changes to have a second approver (2 party control) who does not have a conflict of interest - also useful for meeting compliance obligations.
    - Surface risks and unusualness of request to requester and approver of a permission change.
      - Requires building risk profiles human access to resources
  - Use it or lose it to continuously prune stale permissions in the access matrix
- Continuous risk management for critical accesses/access to critical resources,
    - Regular heuristic based indirect path analysis for critical accesses
      - Pruning access paths that permit non-appropriate principals to get access.
    - Controls (e.g, 2 party approvals, or 2nd party oversight) on operations on critical resources
- Access Intelligence on current state is a required capability to do this

Application Security at Scale

Application Security at different Scales

O(100) major features/year
O(1K) developers

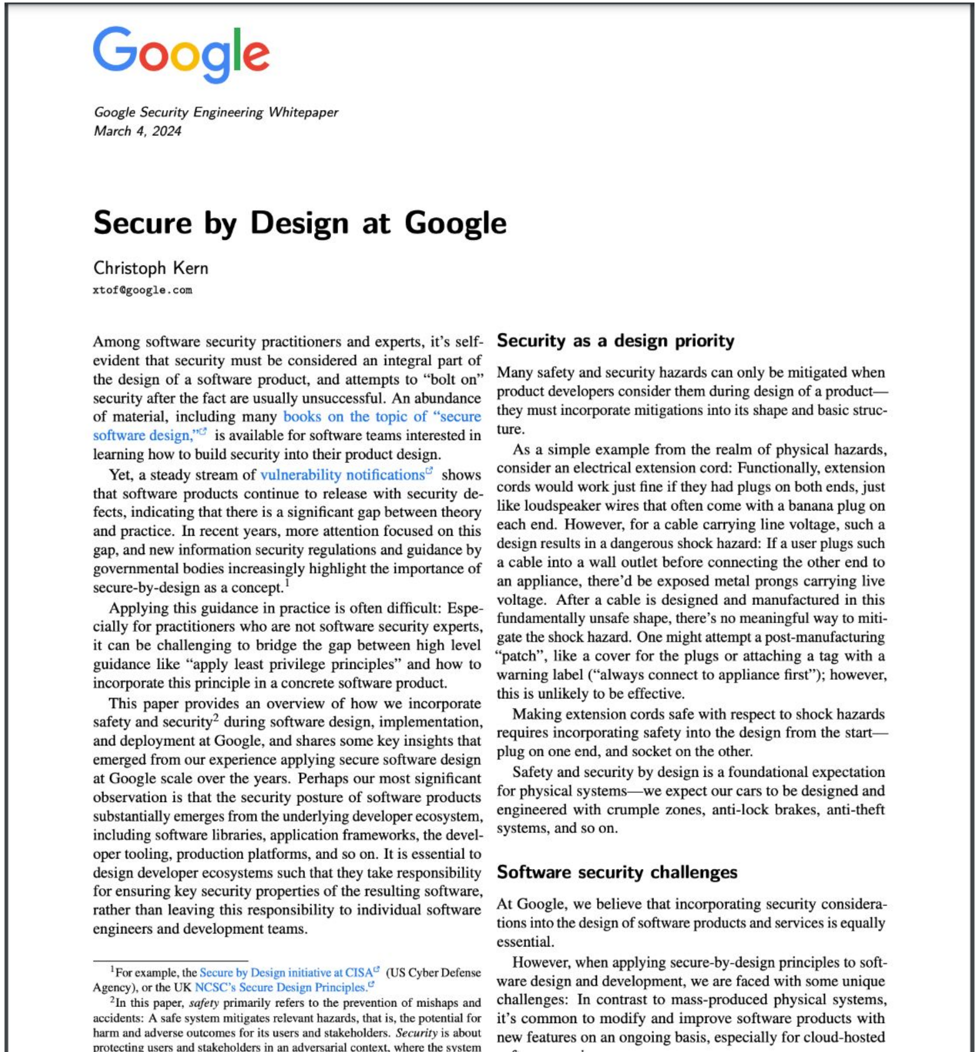
Security Reviews

Train Developers about vulnerabilities and secure coding

Security Certifications for Developers

O(10K) major features/year
O(100K) developers

?



Google Security Engineering Whitepaper
March 4, 2024

Secure by Design at Google

Christoph Kern
xtof@google.com

Among software security practitioners and experts, it's self-evident that security must be considered an integral part of the design of a software product, and attempts to "bolt on" security after the fact are usually unsuccessful. An abundance of material, including many books on the topic of "secure software design," is available for software teams interested in learning how to build security into their product design.

Yet, a steady stream of vulnerability notifications shows that software products continue to release with security defects, indicating that there is a significant gap between theory and practice. In recent years, more attention focused on this gap, and new information security regulations and guidance by governmental bodies increasingly highlight the importance of secure-by-design as a concept.[1]

Applying this guidance in practice is often difficult: Especially for practitioners who are not software security experts, it can be challenging to bridge the gap between high level guidance like "apply least privilege principles" and how to incorporate this principle in a concrete software product.

This paper provides an overview of how we incorporate safety and security[2] during software design, implementation, and deployment at Google, and shares some key insights that emerged from our experience applying secure software design at Google scale over the years. Perhaps our most significant observation is that the security posture of software products substantially emerges from the underlying developer ecosystem, including software libraries, application frameworks, the developer tooling, production platforms, and so on. It is essential to design developer ecosystems such that they take responsibility for ensuring key security properties of the resulting software, rather than leaving this responsibility to individual software engineers and development teams.

[1]For example, the Secure by Design initiative at CISA (US Cyber Defense Agency), or the UK NCSC's Secure Design Principles.

[2]In this paper, *safety* primarily refers to the prevention of mishaps and accidents: A safe system mitigates relevant hazards, that is, the potential for harm and adverse outcomes for its users and stakeholders. *Security* is about protecting users and stakeholders in an adversarial context, where the system

Security as a design priority

Many safety and security hazards can only be mitigated when product developers consider them during design of a product—they must incorporate mitigations into its shape and basic structure.

As a simple example from the realm of physical hazards, consider an electrical extension cord: Functionally, extension cords would work just fine if they had plugs on both ends, just like loudspeaker wires that often come with a banana plug on each end. However, for a cable carrying line voltage, such a design results in a dangerous shock hazard: If a user plugs such a cable into a wall outlet before connecting the other end to an appliance, there'd be exposed metal prongs carrying live voltage. After a cable is designed and manufactured in this fundamentally unsafe shape, there's no meaningful way to mitigate the shock hazard. One might attempt a post-manufacturing "patch", like a cover for the plugs or attaching a tag with a warning label ("always connect to appliance first"); however, this is unlikely to be effective.

Making extension cords safe with respect to shock hazards requires incorporating safety into the design from the start—plug on one end, and socket on the other.

Safety and security by design is a foundational expectation for physical systems—we expect our cars to be designed and engineered with crumple zones, anti-lock brakes, anti-theft systems, and so on.

Software security challenges

At Google, we believe that incorporating security considerations into the design of software products and services is equally essential.

However, when applying secure-by-design principles to software design and development, we are faced with some unique challenges: In contrast to mass-produced physical systems, it's common to modify and improve software products with new features on an ongoing basis, especially for cloud-hosted

"Over the past decades, security and reliability engineers at Google arrived at the key insight that the risk of such implementation and configuration defects being introduced during development and deployment ***is an emergent property*** of the design of development and deployment environments"

Christoph Kern @Google.

Secure by Design Example: Addressing DOM based XSS vulnerabilities

Also Refer to Lectures 8, 9, 10 on Web Security

Applying secure by design to web applications

Problem: Any JavaScript code can introduce DOM-based cross-site scripting (XSS) vulnerabilities, the 2024 CWE Most Dangerous Software Weakness.

Over 60 common, unsafe JS APIs which result in XSS if used on untrusted data: `innerHTML`, `document.write()`, `window.open()`, `location.assign()`, ...

```
document.getElementById('foo').innerHTML = location.hash.slice(1); // XSS bug
```

Exploit: `https://victim.com/#<img src=x onerror=alert('XSS')>`

Challenge at Google: Thousands of engineers, millions of JS changes each day.
How can we robustly prevent this class of vulnerabilities at scale?

Example JS APIs vulnerable to DOM XSS

window.open
HTMLFrameElement.srcdoc
HTMLMediaElement.src
HTMLScriptElement.innerText
HTMLInputElement.formAction
document.write
location.href
HTMLSourceElement.src
Element.innerHTML
HTMLAreaElement.href
HTMLInputElement.src
HTMLFrameElement.src
HTMLBaseElement.href
HTMLTrackElement.src
HTMLButtonElement.formAction
HTMLScriptElement.textContent
HTMLImageElement.src
HTMLFormElement.action
location.assign
HTMLEmbededElement.src

Applying secure by design to web applications

Problem: Any JavaScript code can introduce DOM-based cross-site scripting (XSS) vulnerabilities, the 2024 CWE Most Dangerous Software Weakness.

Over 60 common, unsafe JS APIs which result in XSS if used on untrusted data: `innerHTML`, `document.write()`, `window.open()`, `location.assign()`, ...

```
document.getElementById('foo').innerHTML = location.hash.slice(1); // XSS bug
```

Exploit: `https://victim.com/#<img src=x onerror=alert('XSS')>`

Challenge at Google: Thousands of engineers, millions of JS changes each day.
How can we robustly prevent this class of vulnerabilities at scale?

Trusted Types prevent DOM-based XSS by default

Web security feature (HTTP header) which ensures that all dangerous JS APIs reject plain strings and require special typed objects, safe by construction.

```
Content-Security-Policy: require-trusted-types-for 'script';
```

```
myPolicy = trustedTypes.createPolicy('my-policy', {
   createHTML: (input) => { return DOMPurify.sanitize(input); }});

document.getElementById('foo').innerHTML = value; // Throws error
document.getElementById('foo').innerHTML = myPolicy.createHTML(value) // OK
```

Secure by Design: Web attack protections built into popular Web development frameworks

Only the use of non-standard web development frameworks or code inside `createPolicy` can cause XSS bugs and **needs security review**.

To enable scaled deployments, **supports a *report-only* mode** that notifies application owners about incompatible patterns without enforcing security restrictions.

Enabled in 600+ Google services, **on by default in frameworks.** Handful of XSS-es in 5 years.

Monitoring, Detection and Response

Key concepts

- Measurements and monitoring is a necessity to operating a huge infrastructure reliability
  - Can’t manage what you don’t know
  - Critical to know if something is going wrong, systems are getting corrupted, or some changes or events are beginning to cause failures
- Rolling out security related changes without externally observable outages is completely dependent on having great monitoring
  - **Additional enforcement** or **privilege changes** likely to trigger unforeseen failures
  - Many iterations and rollbacks before some of these changes can be fully rolled out
  - Complex security changes usually rolled out in multiple well designed stages
    - Follow the dependency chain – e.g., key rollout before application key usage rollout
    - **Monitor-only-mode** critical to suss out unforeseen issues when enforcing
    - **The curse of the “Fat tail”**
      - How many issues to fix before failures can be handled by human level intervention
    - **The Race: Who gets in earlier**
      - Race between security enforcement and thousands of developers changing and adding (non-compliant) code running in production.
    - **Story: PSP protocol (https://cloud.google.com/blog/products/identity-security/announcing-psp-security-protocol-is-now-open-source)**

Detection and Response

Detection

- Sensors spread across the infrastructure + use of existing monitors (e.g, traffic volume for DDoS) to detect signs of attack.
- **Sensors on corporate assets to detect signs of attack**
    - **Many TTPs go this route**
- Enormous volume of data generated by sensors
- High performance detection pipeline to analyze humongous amount of sensor data to distill it down to potential threats detected
    - Many sources feed into detection logic, e.g, threat intelligence, TTPs of known attacks and attackers, etc
    - **Threat intelligence:** *"If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle."*

      — Sun Tzu, The Art of War

Detection & Response

- Detection has always been very false positive heavy
  - First level of automated analysis to prioritize detection to hone in on the top findings that need human analysis
    - Limited by # of human experts
  - In smaller environments the first level is manual
- Post identification of an attack/incident, Response function activated
  - Full organization/team specializing in incident management and response
    - Unique skillset(s).
  - Perform further investigation, forensics
  - Also includes
    - legal, law-enforcement. communications to impacted customers etc
  - Determine and take action, monitor progress till incident is closed

Thank You!

Why choose a Cybersecurity Career ?

- A field with constant demand
- Continuous learning and growth
- Making a real difference and serve the greater good
    - *“He who wishes to secure the good of others, has already secured his own.” - Confucius*
    - Supporting Research
        - S.G. Post: *“Altruism, Happiness, and Health: It's Good to Be Good”,* Book Chapter in “*An Exploration of the Health Benefits of Factors That Help Us to Thrive, A Special Issue of the International Journal of Behavioral Medicine”, Editors: Gail Ironson, Lynda H. Powell*

Q&A