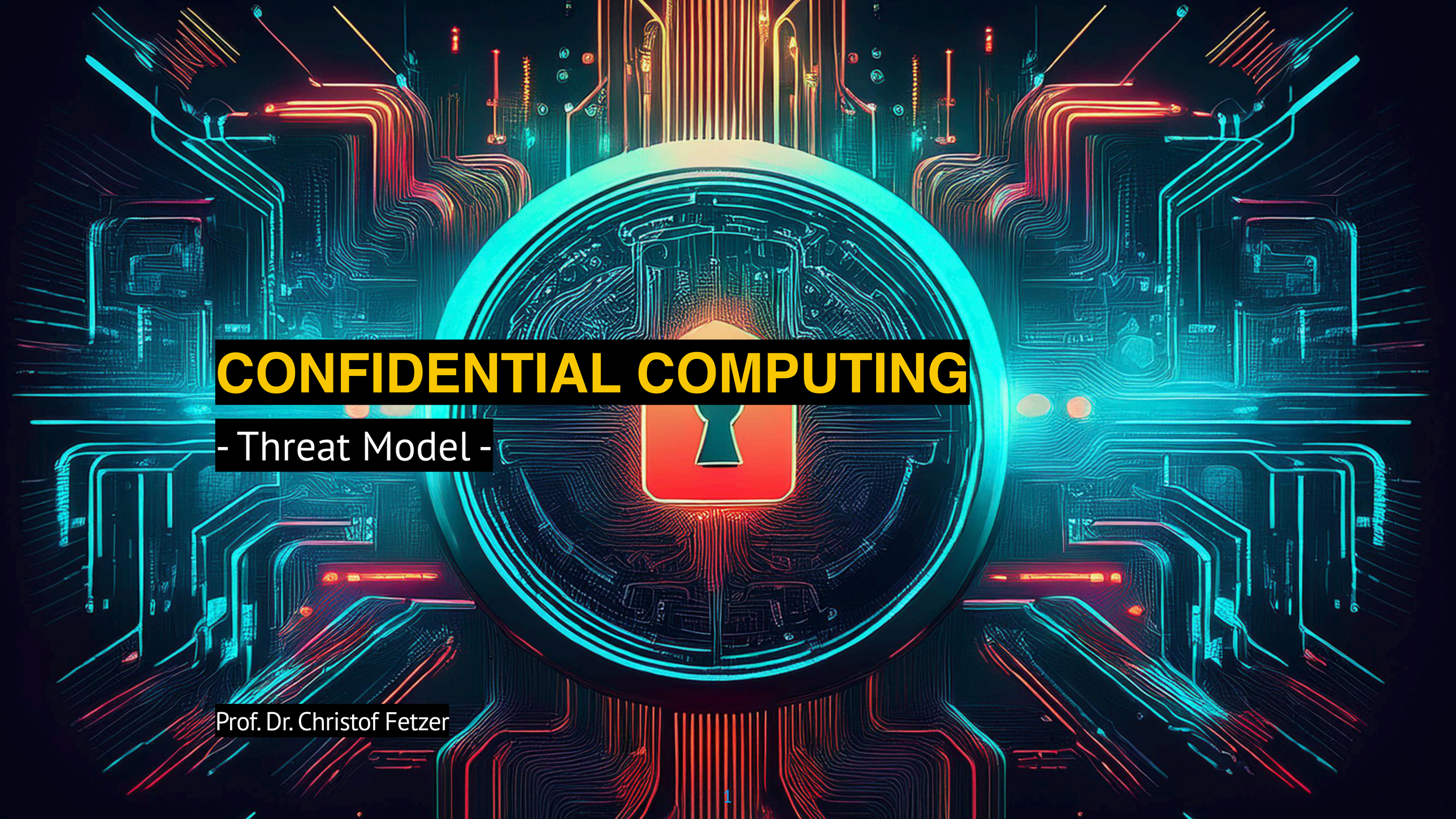


CONFIDENTIAL COMPUTING

- Threat Model -

Prof. Dr. Christof Fetzer

Definition: Threat Model

- A **threat model** is a structured representation of information that affects the security of an application or system.
- The main purposes of a **threat model** are:
  - To identify and enumerate potential threats and vulnerabilities
  - To assess risks associated with identified threats
  - To prioritize mitigation strategies
  - To inform security decisions throughout the software development lifecycle

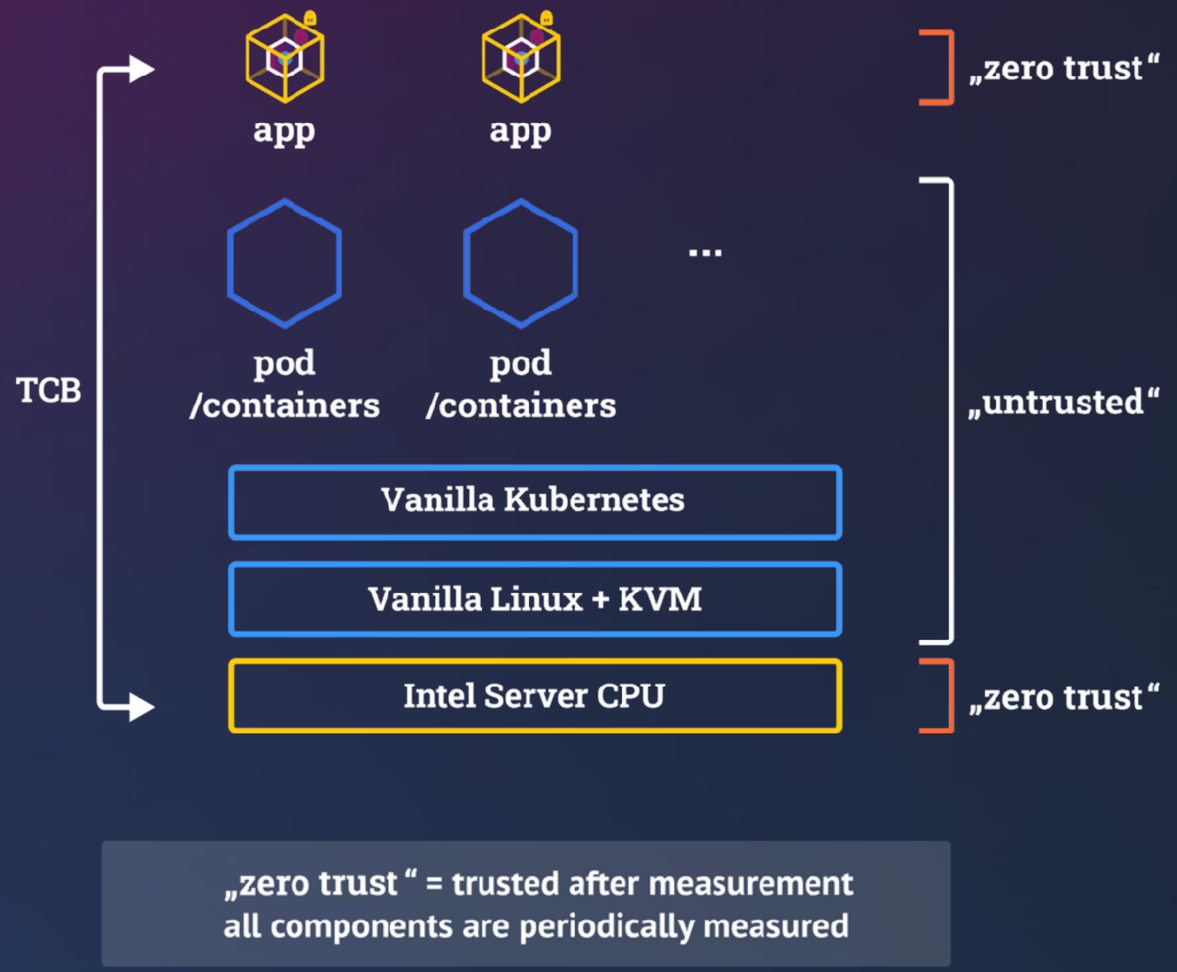
Standard Security Development Lifecycle

Problem

- We reuse lots of services, libraries
  - which were designed for a different/no threat model
- **Approach**:
  - we need to retrofit applications, services, libraries for a new threat model
    - e.g., apps move from a protected data center with trusted staff to a cloud without trusted staff members

Which components to trust?

- CPU with TEE support
  - after attestation & verification
- Services of application
  - after attestation & verification




A Threat Model
- used for confidential computing -

Threat Model

A1) **Unprivileged Software Adversary**

A2) **System Software Adversary**

A3) **Startup Code/SMM Software Adversary**

A4) **Network Adversary**

A5) **Software Side-Channel/Covert-Channel Adversary**

A6) **Simple Hardware Adversary**

A7) ***Roll-Back State Adversary***

A8) ***CVE Adversary***

A9) ***Insider attacks from Security Team***

https://www.intel.com/content/www/us/en/security/security-practices/secure-development-practices/threat-modeling.html

Unprivileged Adversary

(A1) We assume that an adversary might be able to run user code on the same server

- In a **cloud setting**:
  - an adversary can run applications by creating an account - might be collocated with other „tenants“
- **Difficulties for adversary:**
  - the adversary’s code might not run on the same server (1000s of machines)
  - the defender might rent a whole server (bare metal server) - no co-location with other „tenants“
    - **defender**: how to verify that no other application runs on the same server?
  - adversary’s and defender’s code are in different VMs (Virtual Machine, i.e., sandbox)
  - …

https://www.intel.com/content/www/us/en/security/security-practices/secure-development-practices/threat-modeling.html

System Adversary

(A2) System Software Adversary: we assume that an adversary might have administrators' credentials with root access to the system software. Or, they might coerce admins into copying or modifying data or code both at the system and the application level.

- Note:
  - adversary can read all memory (e.g., by „attaching“ to process)
  - adversary can modify memory, modify system call arguments / return values, can read/modify arbitrary files, can read/modify network traffic,...

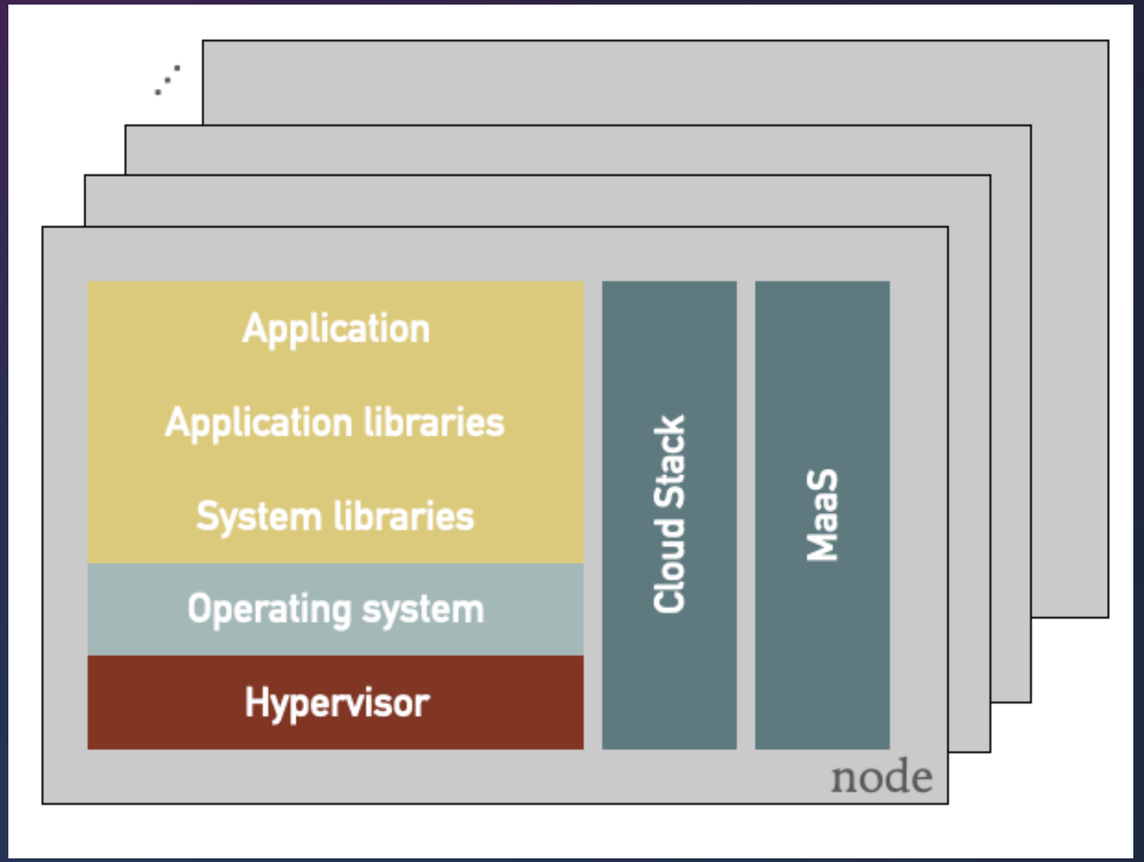
Why assume System Software Adversary?

Reasons:

- **Legal**:
  - cloud provider / cloud employee might be legally required to provide access to the data
- **Liability**:
  - too expensive to err on the threat model
- Limits of access control:
  - How do we know that we can trust individual user?
- **Software complexity**: cannot assume that software is correct (see **Defender's dilemma**)
- **Hardware complexity**: cannot assume that hardware is correct (see **BMC**, **firmware**, ...)
- ...

Defender’s Dilemma

- **Attackers**:
  - success by exploiting a single vulnerability
  - lot’s of new vulnerabilities (every week)
- **Defender**:
  - must protect against every vulnerability
  - not only in application
  - millions of lines of source code
  - not all known by service provider



Linux Kernel

https://openhub.net/p/linux/analyses/latest/languages_summary (Oct 2924)

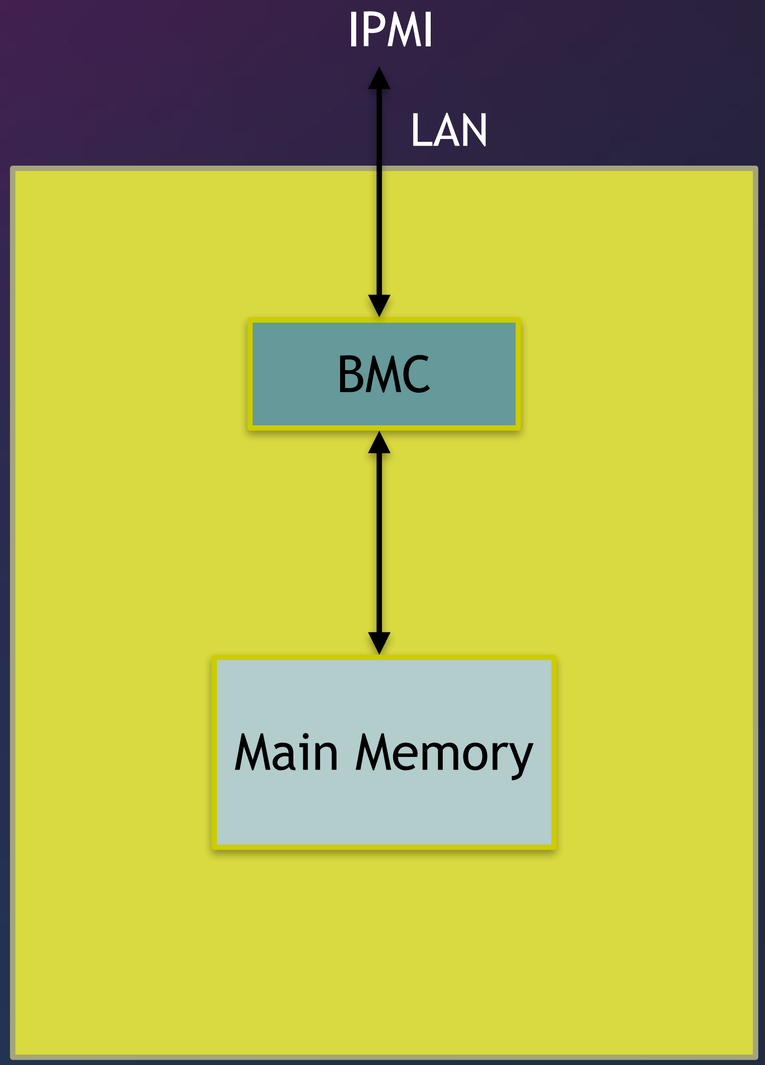
Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C	34,711,115	5,014,716	12.6%	5,609,320	45,335,151	96.2%
C++	544,550	242,660	30.8%	98,456	885,666	1.9%
Assembly	261,970	48,411	15.6%	48,589	358,970	0.8%
shell script	224,012	40,316	15.3%	59,306	323,634	0.7%
Python	82,539	15,084	15.5%	17,239	114,862	0.2%
Make	72,629	15,517	17.6%	15,118	103,264	0.2%
Perl	54,012	6,752	11.1%	10,376	71,140	0.2%
Rust	6,726	6,360	48.6%	1,116	14,202	0.0%
AWK	2,510	302	10.7%	397	3,209	0.0%
XML Schema	349	9	2.5%	46	404	0.0%
XSL Transformation	286	128	30.9%	65	479	0.0%
UnrealScript	264	17	6.0%	15	296	0.0%
CSS	262	106	28.8%	89	457	0.0%
Autoconf	91	5	5.2%	15	111	0.0%
Ruby	50	0	0.0%	8	58	0.0%
Matlab	35	37	51.4%	17	89	0.0%
Vim Script	27	12	30.8%	3	42	0.0%
Automake	25	3	10.7%	6	34	0.0%
JavaScript	6	0	0.0%	0	6	0.0%
HTML	-779	3	-	-121	-897	0.0%
XML	-59,777	1,245	-	-5,214	-63,746	-0.1%
Totals	35,900,902	5,391,683		5,854,846	47,147,431	

Considerations

- Linux Kernel: About 50 new vulnerabilities per week
- Study indicates:
  - security vulnerabilities stay for about 5 years in kernel
    - before detected and fixed
  - i.e., we always need to expect that there are undetected vulnerabilities in the kernel / any software

Host Management Service

- Servers have management support
  - via LAN using, e.g., IPMI
- **Server**
  - have separate CPU (BMC)
  - have separate management interface
- **BMC is in control, i.e.,**
  - host OS cannot switch off BMC
- **BMC can access main memory**
  - can dump secrets / modify data

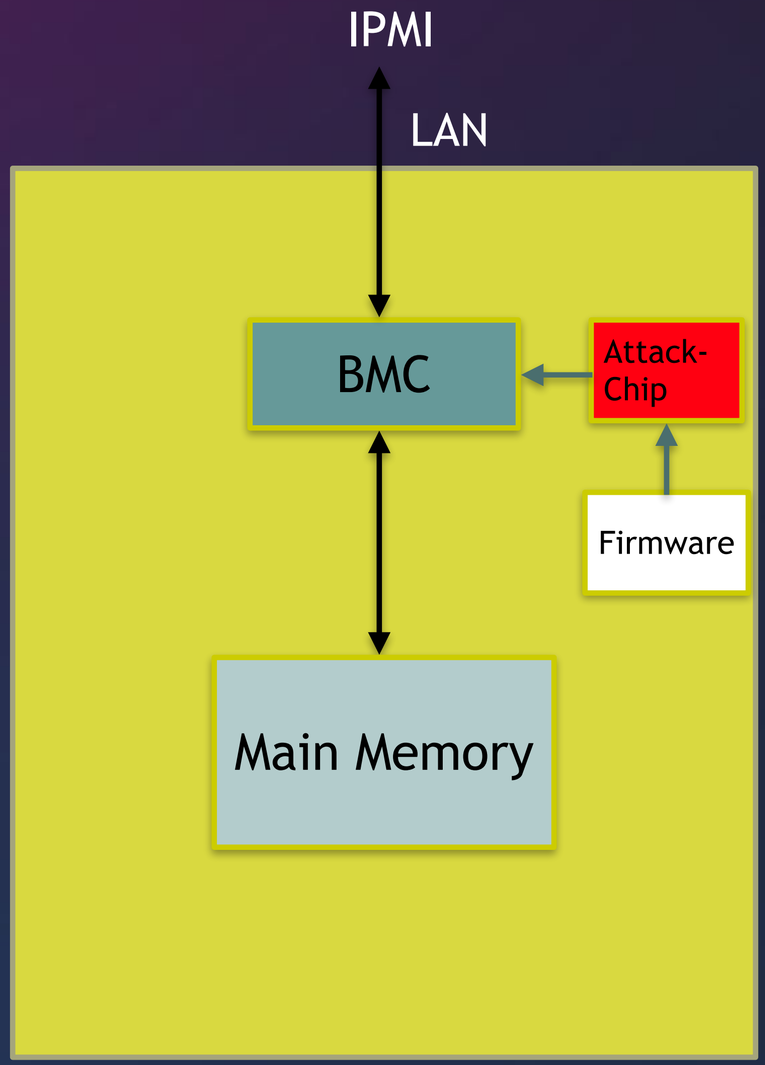


BMC = Baseboard Management Controller
IPMI = Intelligent Platform Management Interface

Supply-Chain Attack (Supermicro Boards)

- Attack-Chip added to motherboard
  - very small - looks like a passive element
- Attack-Chip extends BMC code

- News:
  - This was detected because of crashes
- (Note: Not 100% sure if this news story is true)



BMC = Baseboard Management Controller
IPMI = Intelligent Platform Management Interface

Adversaries

A3) **Startup Code/SMM Software Adversary**

We assume that an adversary can modify the code initially booted on the server and change the **system management mode** (SMM) code. **This adversary might also try to access the machine via the BMC**.

- **SMM** provides a special operating mode for handling system-wide functions like **power management**, **hardware control**, and proprietary OEM-designed code.

- SMM runs at a higher privilege level than the hypervisor and kernel mode, making it extremely powerful - e.g., can access memory without restriction.

Adversaries

A4) **Network Adversary**

We assume that an adversary with access to the network can try to communicate with app to gain access or trigger bugs in app.

- Example:
  - TLS implementations have had bugs and an adversary might use these bugs to gain access without having the right credentials

Adversaries

A5) **Software Side-Channel/Covert-Channel Adversary**

An adversary could try to extract secrets and keys from app using side channels that circumvent the protections of TEEs.

- Example:
  - the adversary might measure access times to certain memory locations. It can derive if a cache line might have been used by another application. In return, adversary might learn about the secrets of an application

Adversaries

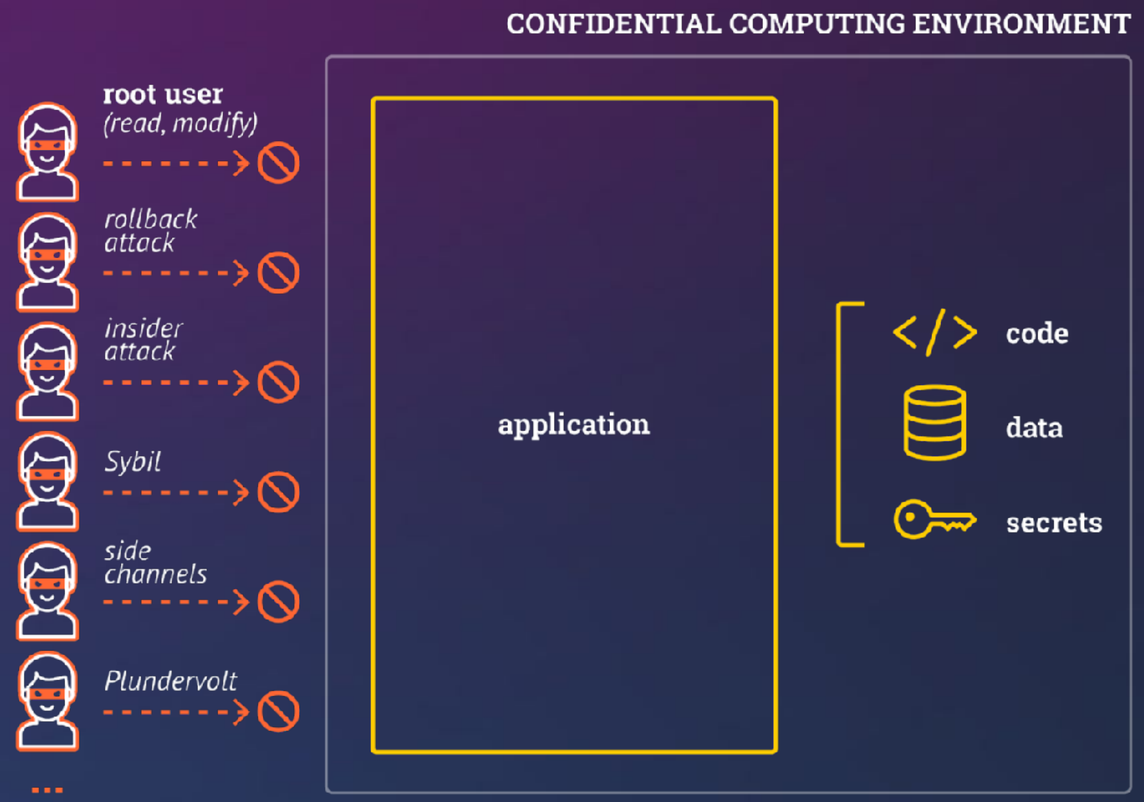
A6) Simple Hardware Adversary

- A6.1) The adversary can remove DRAMs, has tools for hardware debugging, and can listen to buses.
- A6.2) The adversary gets hold of decommissioned hardware. The adversary recovers a CPU, TPM, and database when the hardware or service is decommissioned.

A7) Roll-Back State Adversary:

- A7.1) The adversary can **roll back the disk state**. The disk state might contain an encrypted database with keys that have been updated already.
- A7.2) The adversary can **roll back the CPU firmware** or the app code. The adversary can install an old version of the CPU firmware to exploit known bugs in enclave implementation. An adversary could use this to extract information from, say, an encrypted key database.
- A7.3) The adversary can freeze the state of an existing app. This is to keep access to an earlier version of the database.

Lots of attacks known and possible…



Must check for mitigations!

A8 CVE Adversary

The adversary knows all components of an application, i.e., all dependencies and their versions of each of the application services are known.

The adversary knows all the vulnerabilities that affect these components and will exploit these vulnerabilities to attack the application.

A9) Insider Attacks

A9) Insider attack from security team. Attacker has credentials to access security policies and might try to change policies to grant access to data, code, and secrets.

Threats Not Covered: A10-A12

A1) **Unprivileged Software Adversary**

A2) **System Software Adversary**

A3) **Startup Code/SMM Software Adversary**

A4) **Network Adversary**

A5) **Software Side-Channel/Covert-Channel Adversary**

A6) **Simple Hardware Adversary**

A7) ***Roll-Back State Adversary***

A8) ***CVE Adversary***

A9) Insider Attack (from Security Team)

A10) **Skilled Hardware Adversary**

A11) **Hardware Reverse Engineer Adversary**

A12) **Authorized Adversary**

https://www.intel.com/content/www/us/en/security/security-practices/secure-development-practices/threat-modeling.html

Threat Model

Questions?