Computer and Network Security

Lecture 14:
OS & Hardware Security

COMP-5370/6370
Fall 2025

Do you trust other people's code?

- Do you use an HP computer?
  - Android OS or Google's Android Apps?
  - Google services (GMail, Docs, Calendar, etc)?

Finster Professor

Shady Professor

Profesor Turbio

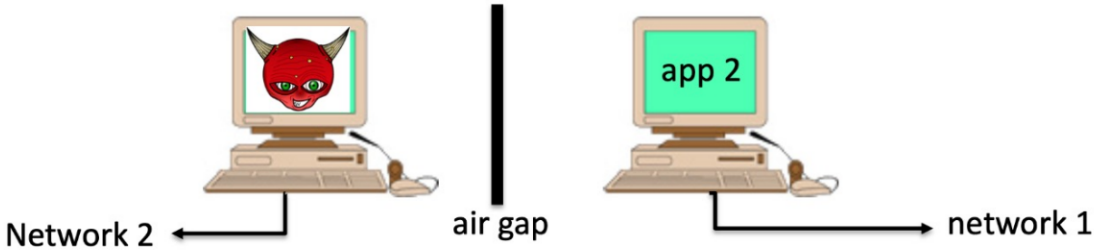
Architectural Defenses

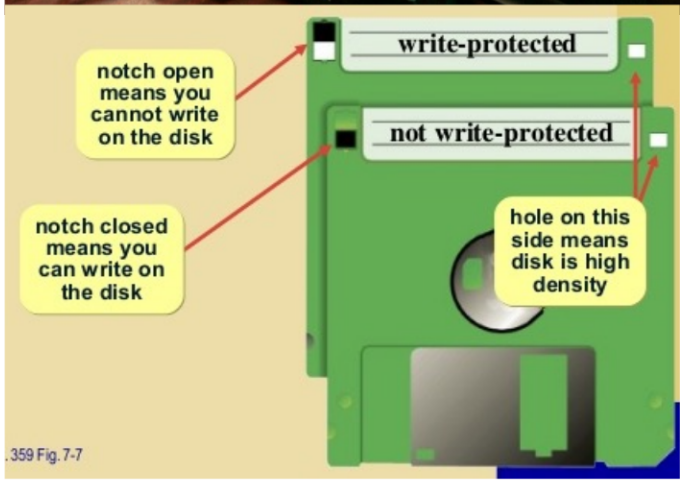
An **architectural defense** is one which is both generic in terms of implementation and focused on *isolating* a potential misbehaving application or process.

- Not tied to a specific application or attack
- ***Can defend against unknown attacks in the future due to generality***

Defense: Air Gap Hardware

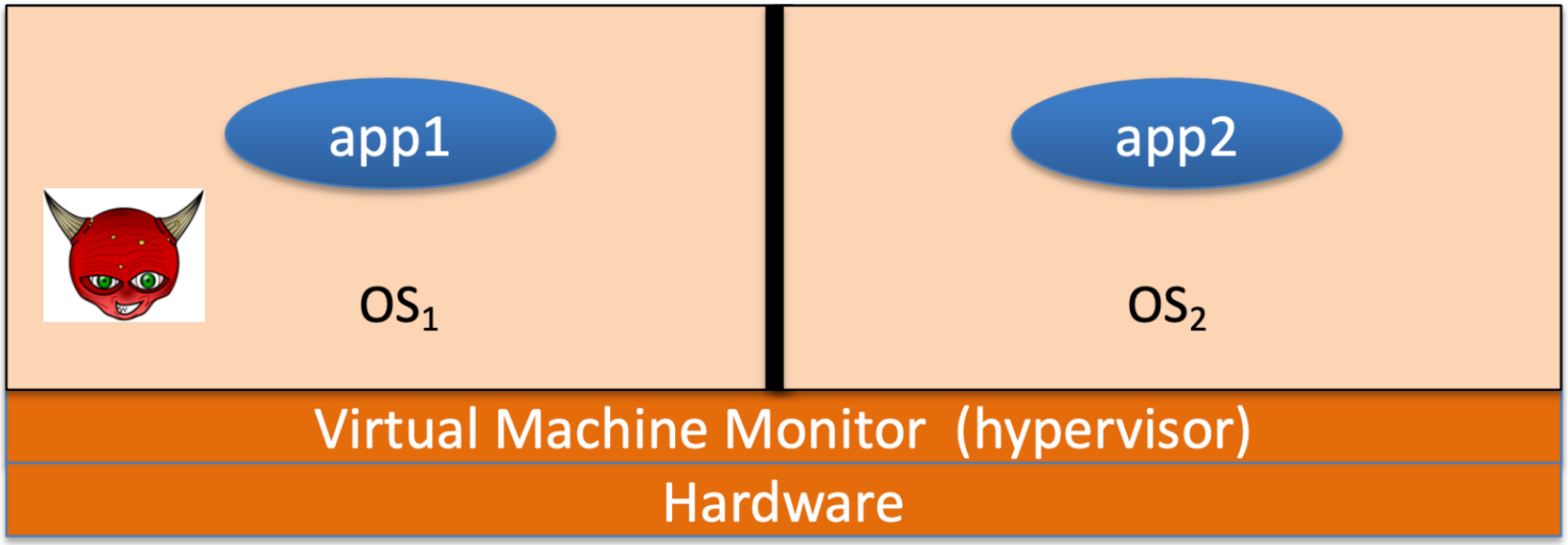
- Physically separated systems/networks
- No logical interaction across boundary
- Physical data transfer across boundary





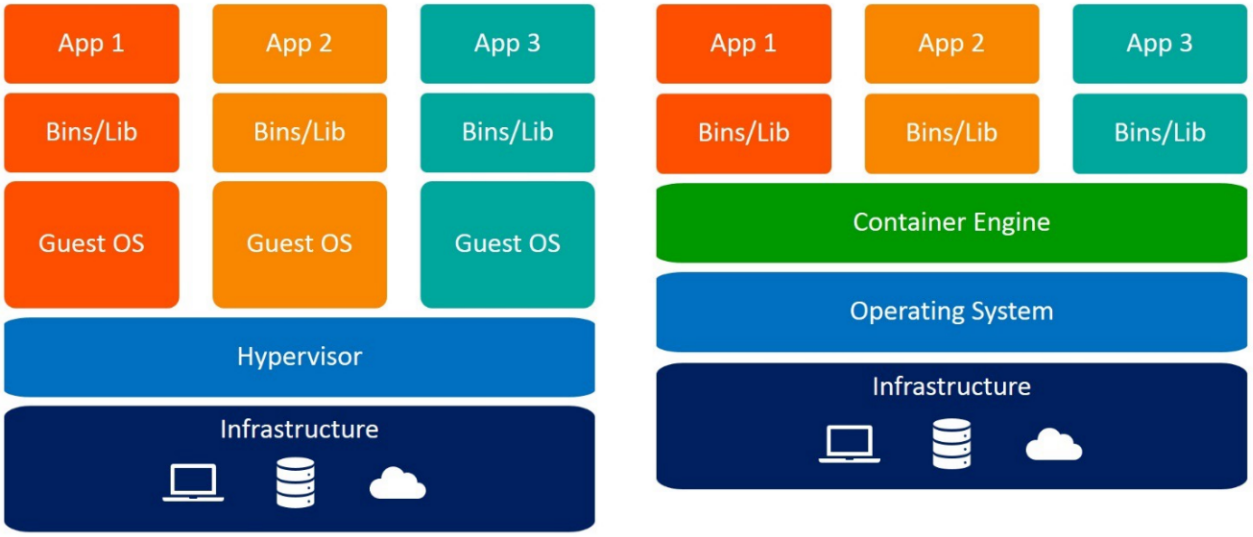
Defense: Virtual Machines

- OS-level isolation
  - Similar to dual-boot but simultaneous
- A “hypervisor” coordinates hardware access
- OSes are isolated (Linux VM on Windows)



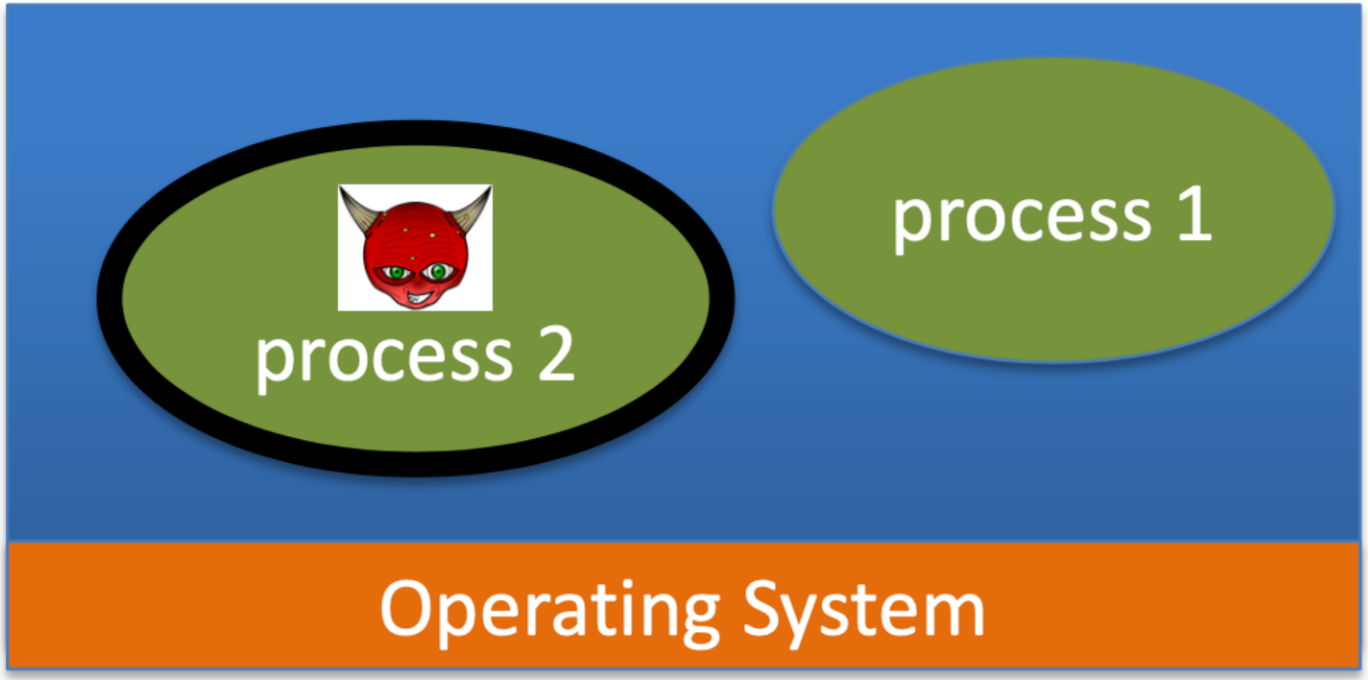
Defense: Containers

- Perspective-level isolation
- Shared kernel w/ isolated perspective
  - Each container thinks it's the only thing running on the entire computer



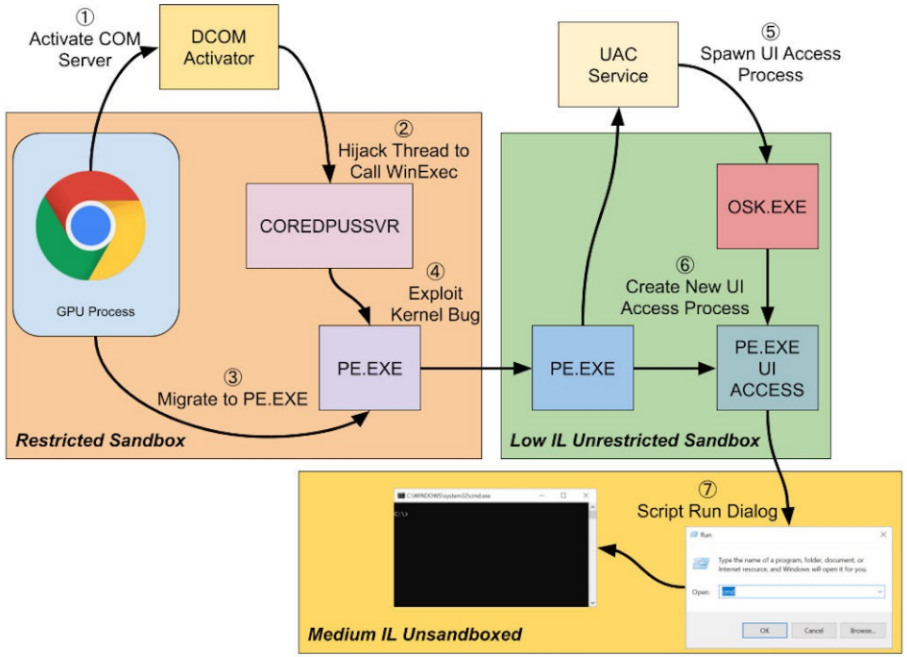
Defense: Process Isolation

- Memory-level isolation
- Managed via OS scheduler
- Extremely efficient due to virtual memory



Defense: Thread Sandbox

- Logic-level isolation (code logic)
- Threads interact via memory & IPCs
- Tainted threads can be killed and restarted

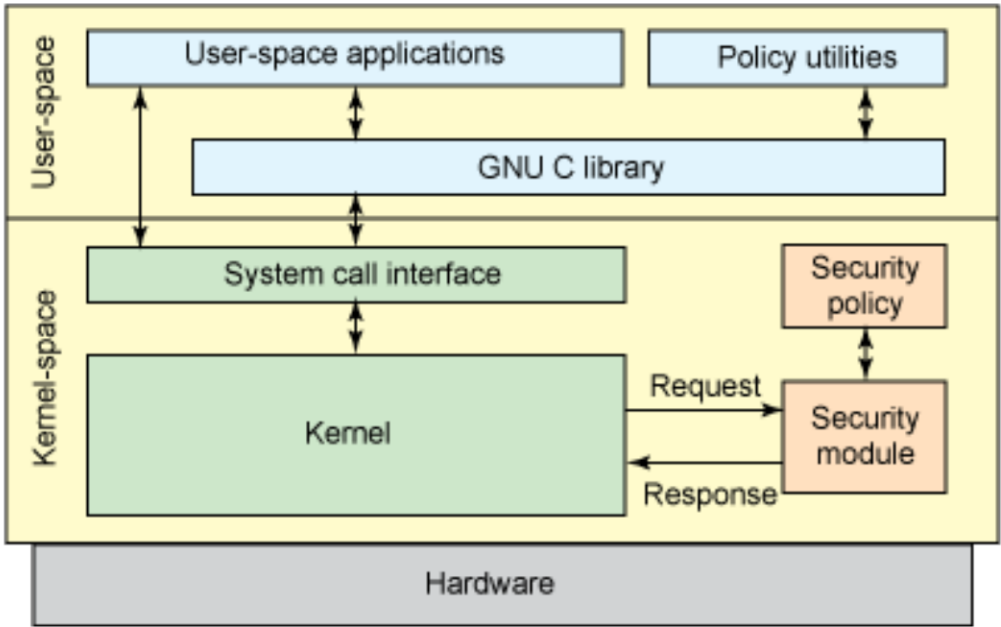


Defense: Thread Sandbox

- “Do one thing and do it safely”
- A “policy engine” can blindly enforce data interactions and data exchanges
- Very useful for web browsers
  - Request content, run JS, render image, etc are *very* different things with predictable inputs and outputs
- Better than nothing but is exceptionally difficult to internally isolate behavior

Defense: SELinux/App Armor

- Permission-level isolation
- **HEAVILY** patched set of kernel modules
- "Know what an application is suppose to do and don't let it do anything else."
  - `ls` doesn't need network access
  - Print driver doesn't need keystrokes



Defense: Use-Specific HW

- Required to install a shady VTC program?
  - Get a $50 tower from 2012 and $20 monitor
- Need to test a really sketchy app?
  - Get a $30 Android phone with Wifi
- Need to install a known-spyware extension?
  - Chromebooks are $100 on-sale




What do you think?

avast

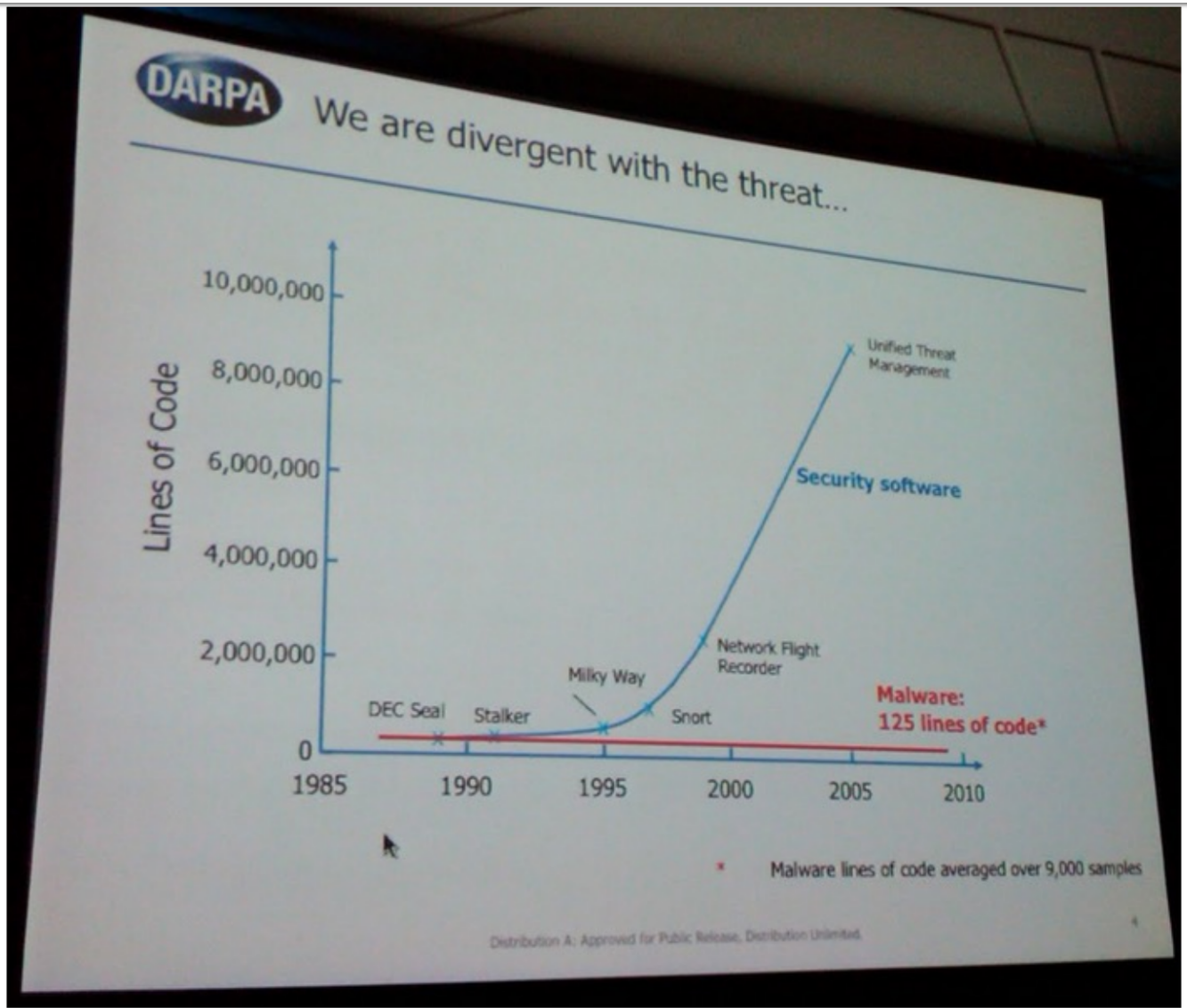
McAfee™

Norton™

Defense: Anti-Virus

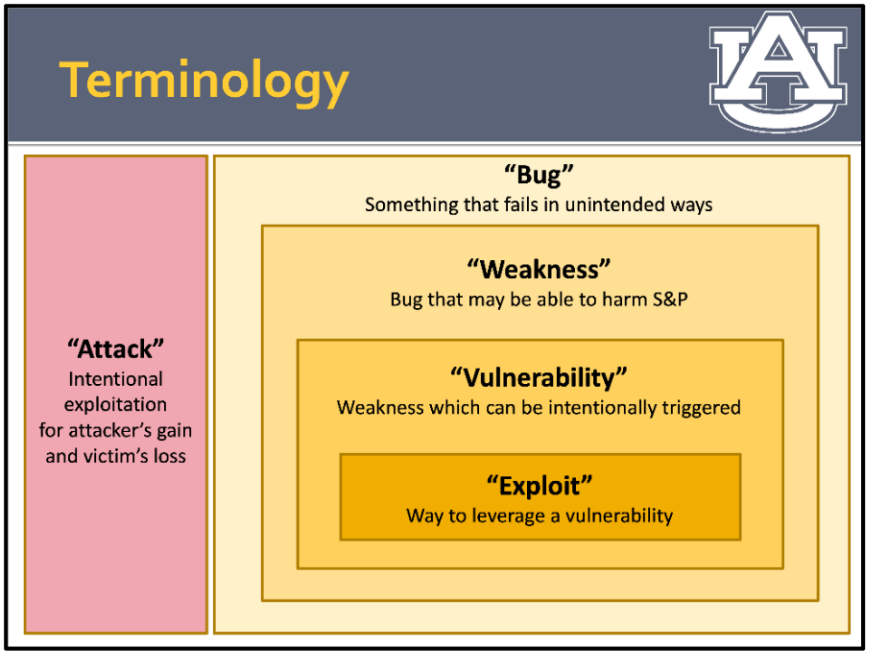
- Anti-Virus and “host-based defense systems” are … complicated…
- **Good** – They can quarantine and alert you when there’s known malware.
- **Maybe** – They can only tell you about known malware and the naïve versions
- **Bad** – They love to quarantine dev-tools and known-benign security tools/apps
- **Worse** – They have to touch **everything**

Defense: Anti-Virus

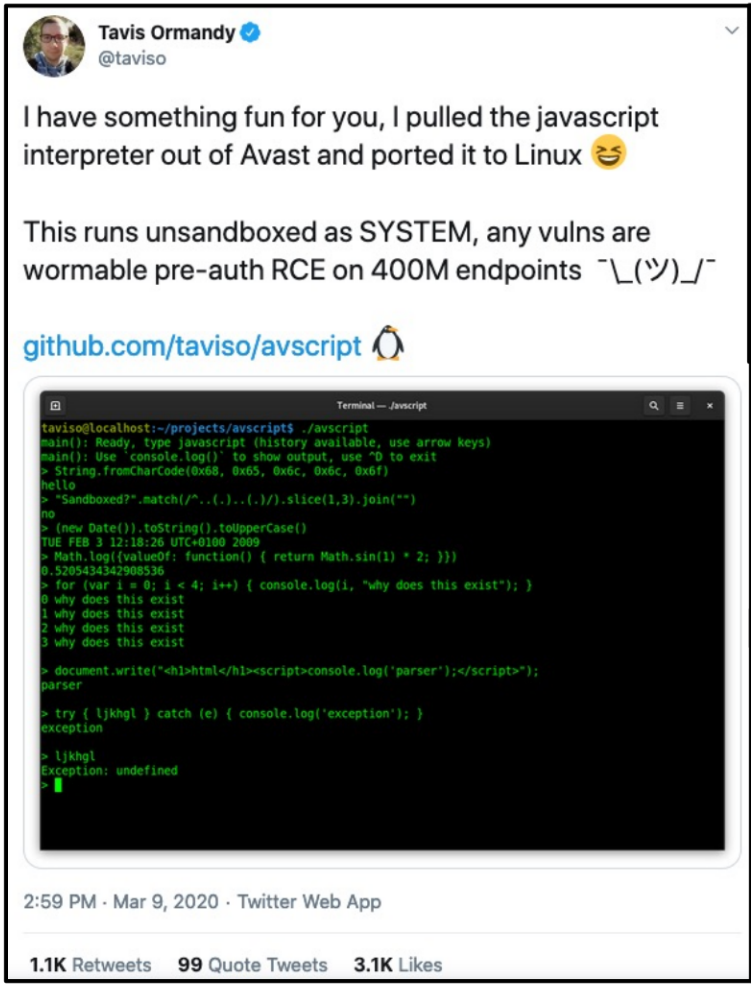


Software Facts of Life

- Software has bugs
- Some bugs are weaknesses
- Some weaknesses are vulnerabilities
- Some vulnerabilities can be exploited
- Someone has an interest in exploiting others for gain



Defense: Anti-Virus



Tavis Ormandy
@taviso

I have something fun for you, I pulled the javascript interpreter out of Avast and ported it to Linux 😆

This runs unsandboxed as SYSTEM, any vulns are wormable pre-auth RCE on 400M endpoints ¯_(ツ)_/¯

github.com/taviso/avscript 🐧

```
Terminal — ./avscript
taviso@localhost:~/projects/avscript$ ./avscript
main(): Ready, type javascript (history available, use arrow keys)
main(): Use `console.log()` to show output, use ^D to exit
> String.fromCharCode(0x68, 0x65, 0x6c, 0x6c, 0x6f)
hello
> "Sandboxed?".match(/^..(.)..(.)/).slice(1,3).join("")
no
> (new Date()).toString().toUpperCase()
TUE FEB 3 12:18:26 UTC+0100 2009
> Math.log({valueOf: function() { return Math.sin(1) * 2; }})
0.5205434342908536
> for (var i = 0; i < 4; i++) { console.log(i, "why does this exist"); }
0 why does this exist
1 why does this exist
2 why does this exist
3 why does this exist

> document.write("<h1>html</h1><script>console.log('parser');</script>");
parser

> try { ljkhgl } catch (e) { console.log('exception'); }
exception

> ljkhgl
Exception: undefined
>
```

2:59 PM · Mar 9, 2020 · Twitter Web App

1.1K Retweets **99** Quote Tweets **3.1K** Likes

ars TECHNICA SUBSCRIBE SIGN IN

BIZ & IT —

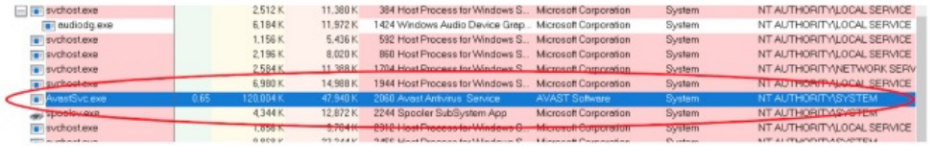
This Windows Defender bug was so gaping its PoC exploit had to be encrypted

Is there a fuzzer in the house?

DAN GOODIN - 6/26/2017, 7:10 PM

Avast Antivirus JavaScript Interpreter

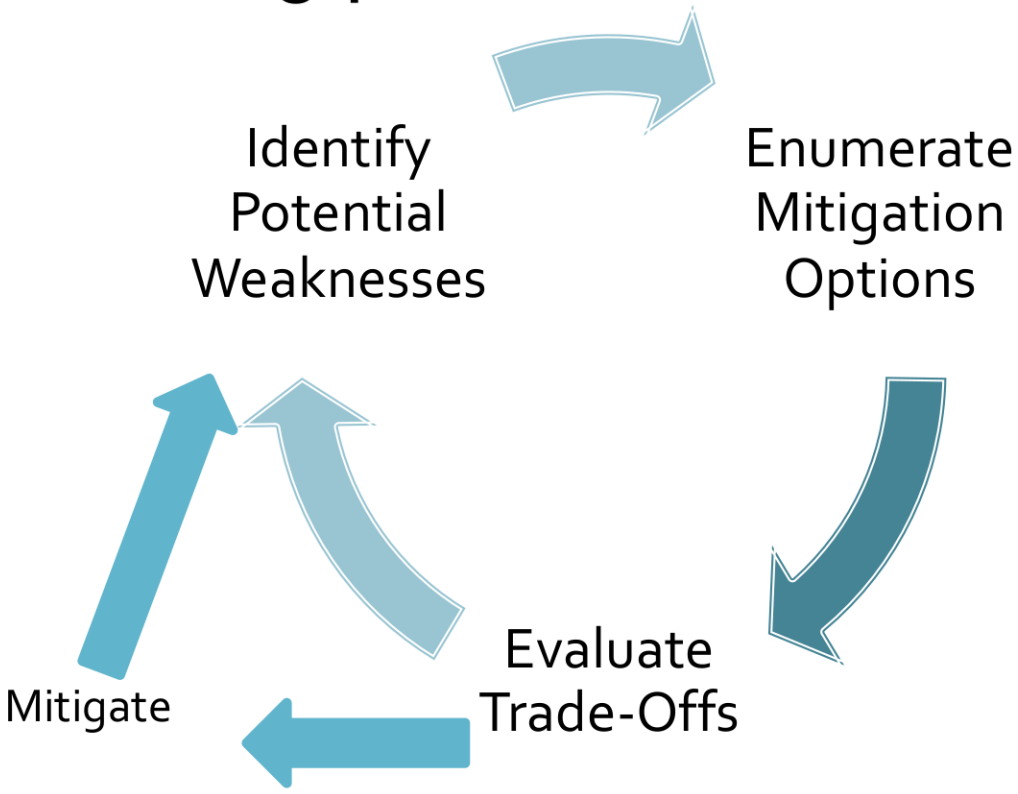
The main Avast antivirus process is called AvastSvc.exe, which runs as SYSTEM.



TLS Security	Increased Security	Same Security	Decreased Security	Severely Broken
Client Security Products	0/20	2/20	18/20	10/20
Middleboxes	0/12	1/12	6/12	5/12

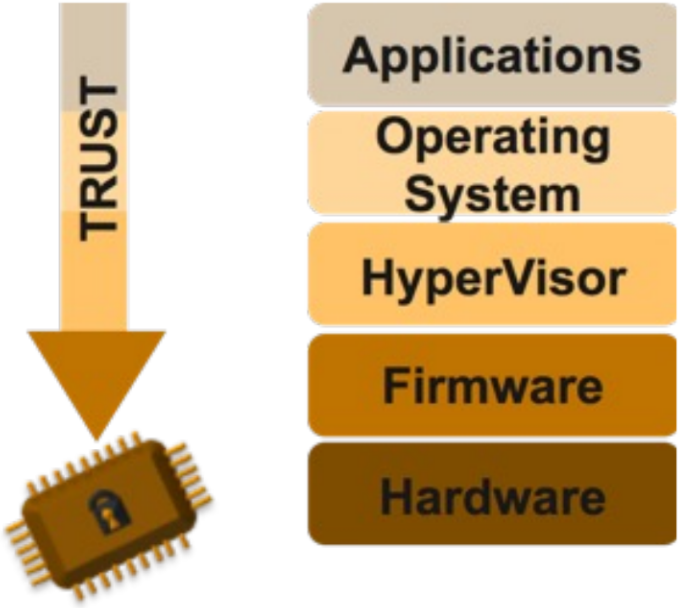
Threat Modeling

A systematic approach to analyzing and understanding potential weaknesses.



Trusted Computing Base

The **Trusted Computing Base (TCB)** is the collection of all components within a system critical to providing security properties.



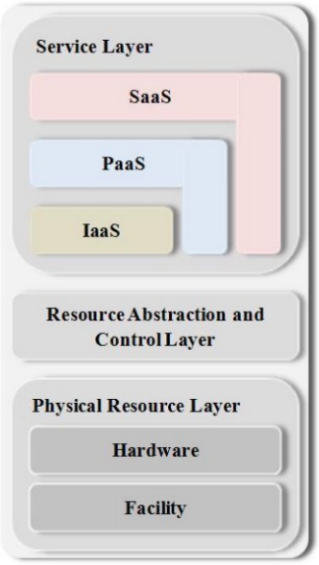
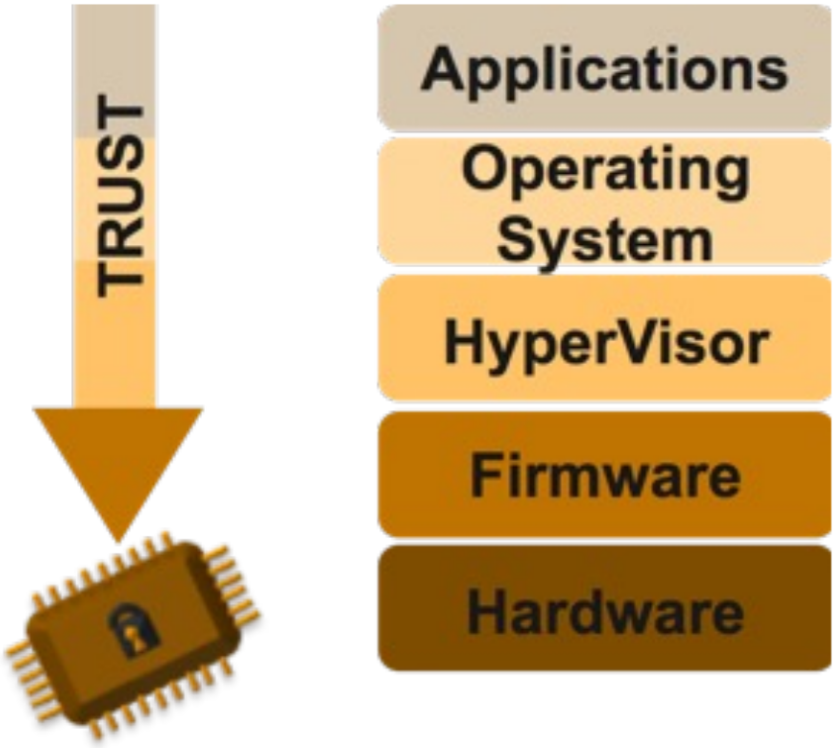


Figure 15: Cloud Provider - Service Orchestration

Local TCB



- Everything needed to run the application safely
- Each layer relies on the layers below it to behave correctly

Turtles All The Way Down

- Attack surface is exponentially larger b/c malicious lower-levels
- Level N bug means levels >=N are untrustworthy
- Bugs and vulns mimic each other due to abstraction

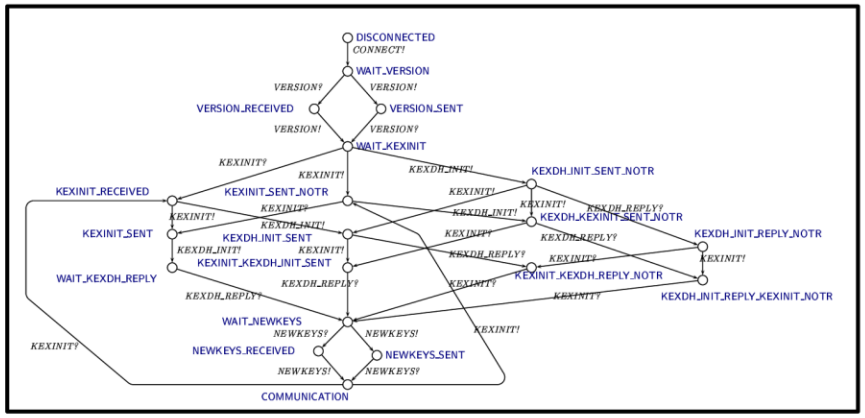
Let's Connect to a Server

Let's Connect to a Server

- Bash Environment

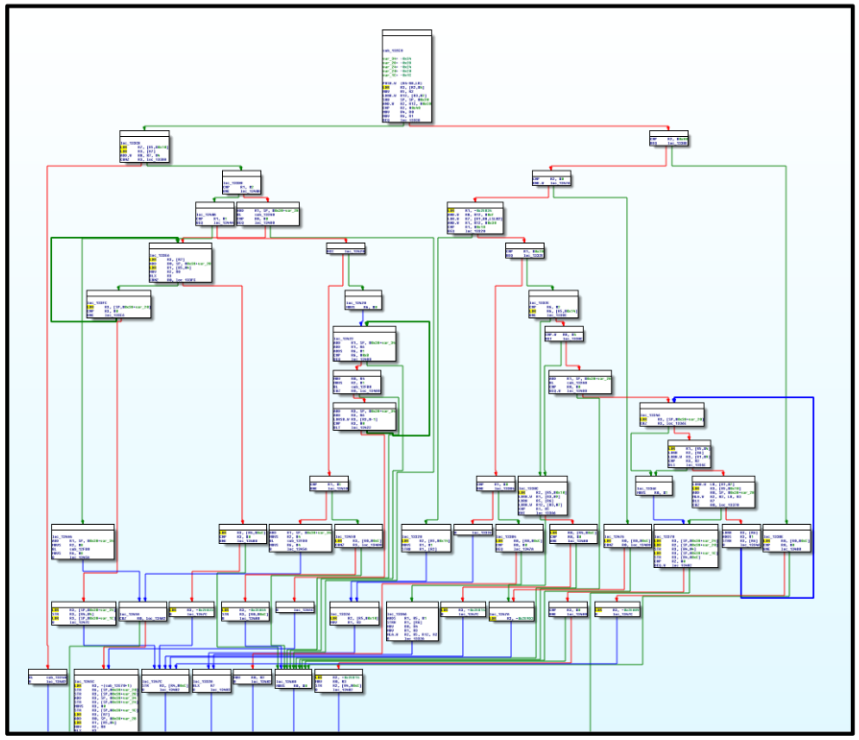
```
$>
$> ssh username@host
```

Let's Connect to a Server

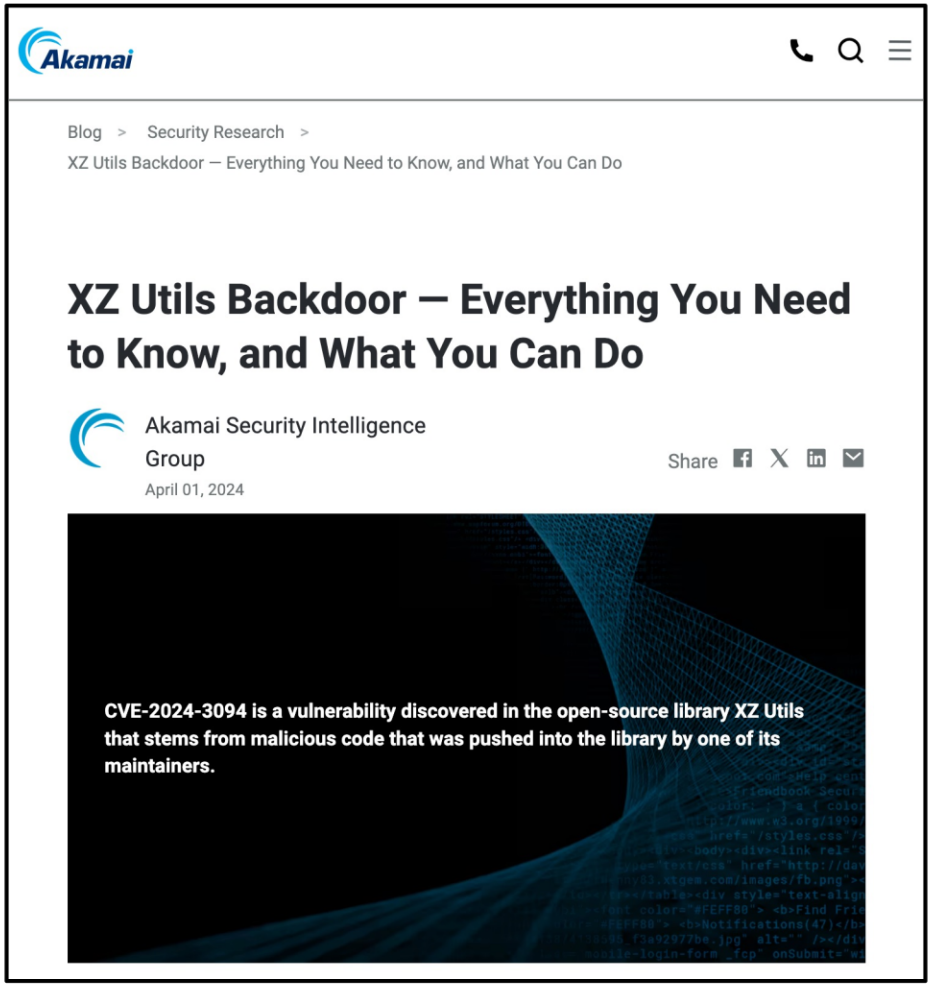


- Bash Environment
- SSH protocol design

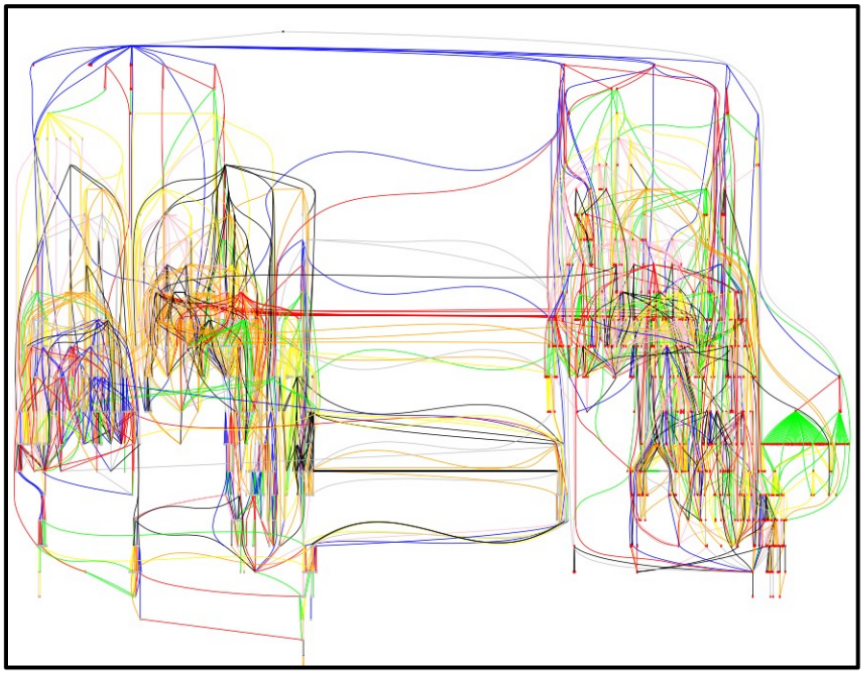
Let's Connect to a Server



- Bash Environment
- SSH protocol design
- SSH app architecture


Akamai
Blog > Security Research >
XZ Utils Backdoor — Everything You Need to Know, and What You Can Do
XZ Utils Backdoor — Everything You Need to Know, and What You Can Do
Akamai Security Intelligence Group
April 01, 2024
Share
CVE-2024-3094 is a vulnerability discovered in the open-source library XZ Utils that stems from malicious code that was pushed into the library by one of its maintainers.

Let's Connect to a Server



- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic

Let's Connect to a Server

$$
\begin{array}{l}
SEC_k : \mathsf{Hon}(\hat{C},\hat{K}) \supset (\mathsf{GoodKeyAgainst}(X,k) \vee \hat{X} \in \{\hat{C},\hat{K}\}) \\
SEC_{akey} : \mathsf{Hon}(\hat{C},\hat{K},\hat{T}) \supset (\mathsf{GoodKeyAgainst}(X,AKey) \vee \hat{X} \in \{\hat{C},\hat{K},\hat{T}\}) \\
SEC_{skey} : \mathsf{Hon}(\hat{C},\hat{K},\hat{T},\hat{S}) \supset (\mathsf{GoodKeyAgainst}(X,SKey) \vee \hat{X} \in \{\hat{C},\hat{K},\hat{T},\hat{S}\}) \\
AUTH_{kas} : \exists \eta.\ \mathsf{Send}((\hat{K},\eta), Cert_K.SIG[sk_K](\text{“}DHKey\text{”}.gy.\tilde{n_1}).E_{sym}[k^{t\to k}_{T,K}](AKey.\hat{C}). \\
\qquad E_{sym}[k](AKey.n_1.\hat{T})) \\
AUTH_{tgs} : \exists \eta.\ \mathsf{Send}((\hat{T},\eta), \hat{C}.E_{sym}[k^{s\to t}_{S,T}](SKey.\hat{C}).E_{sym}[AKey](SKey.n_2.\hat{S}))
\end{array}
$$

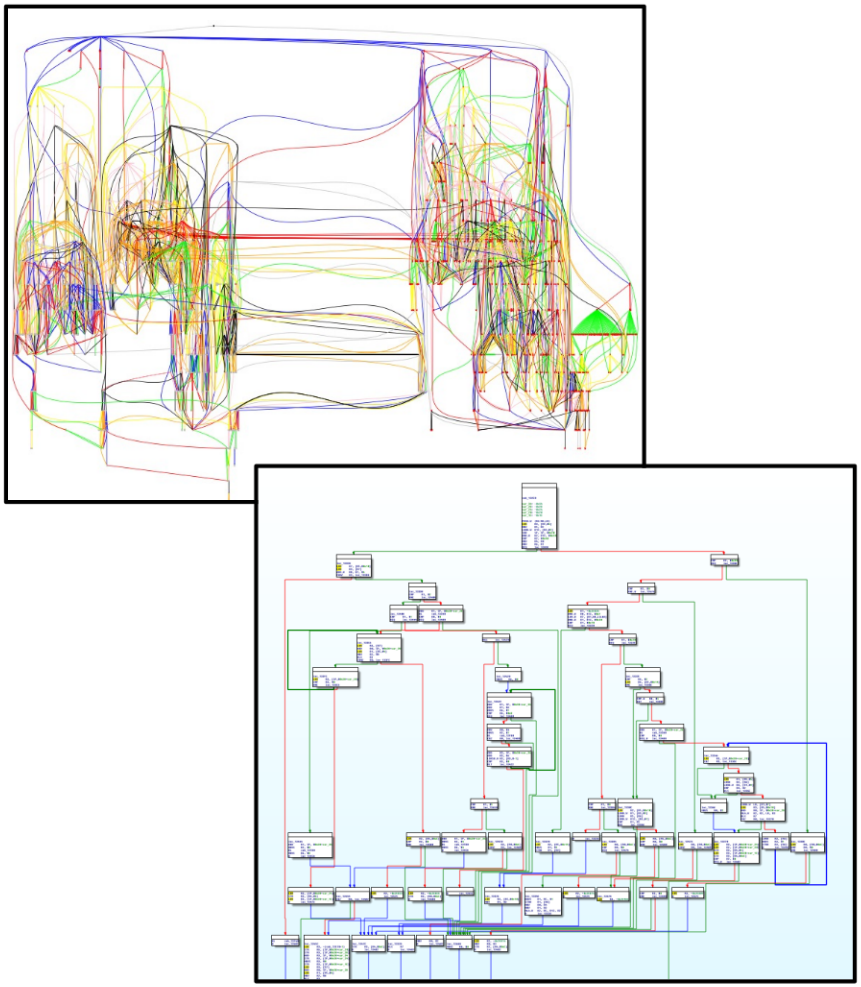
$$
\begin{array}{ll}
SEC^{client}_{k} : [\mathbf{Client}]_C\ SEC_k & SEC^{kas}_{k} : [\mathbf{KAS}]_K\ SEC_k \\
\\
SEC^{client}_{akey} : [\mathbf{Client}]_C\ SEC_{akey} & AUTH^{client}_{kas} : [\mathbf{Client}]_C\ \mathsf{Hon}(\hat{C},\hat{K}) \supset AUTH_{kas} \\
SEC^{kas}_{akey} : [\mathbf{KAS}]_K\ SEC_{akey} & AUTH^{tgs}_{kas} : [\mathbf{TGS}]_T\ \mathsf{Hon}(\hat{T},\hat{K}) \\
SEC^{tgs}_{akey} : [\mathbf{TGS}]_T\ SEC_{akey} & \qquad \supset \exists n_1,k,gy,\tilde{n_1}.\ AUTH_{kas} \\
 & AUTH^{client}_{tgs} : [\mathbf{Client}]_C\ \mathsf{Hon}(\hat{C},\hat{K},\hat{T}) \supset AUTH_{tgs} \\
SEC^{client}_{skey} : [\mathbf{Client}]_C\ SEC_{skey} & AUTH^{server}_{tgs} : [\mathbf{Server}]_S\ \mathsf{Hon}(\hat{S},\hat{T}) \\
SEC^{tgs}_{skey} : [\mathbf{TGS}]_T\ SEC_{skey} & \qquad \supset \exists n_2, AKey.\ AUTH_{tgs}
\end{array}
$$

Table 1. DHINIT Security Properties

$$
\begin{array}{rl}
C \longrightarrow K(I) : & Cert_C.SIG[sk_C](\text{“}Auth\text{”}.HASH(\hat{C}.\hat{T}.n_1).\tilde{n_1}.gx).\hat{C}.\hat{T}.n_1 \\
I \longrightarrow K : & Cert_I.SIG[sk_I](\text{“}Auth\text{”}.HASH(\hat{I}.\hat{T}.n_1).\tilde{n_1}.gx).\hat{I}.\hat{T}.n_1 \\
K \longrightarrow I \longrightarrow C : & Cert_K.SIG[sk_K](\text{“}DHKey\text{”}.gy.\tilde{n_1}). \\
 & \qquad E_{sym}[k^{t\to k}_{T,K}](AKey.\hat{I}).E_{sym}[k](AKey.n_1.\hat{T})
\end{array}
$$

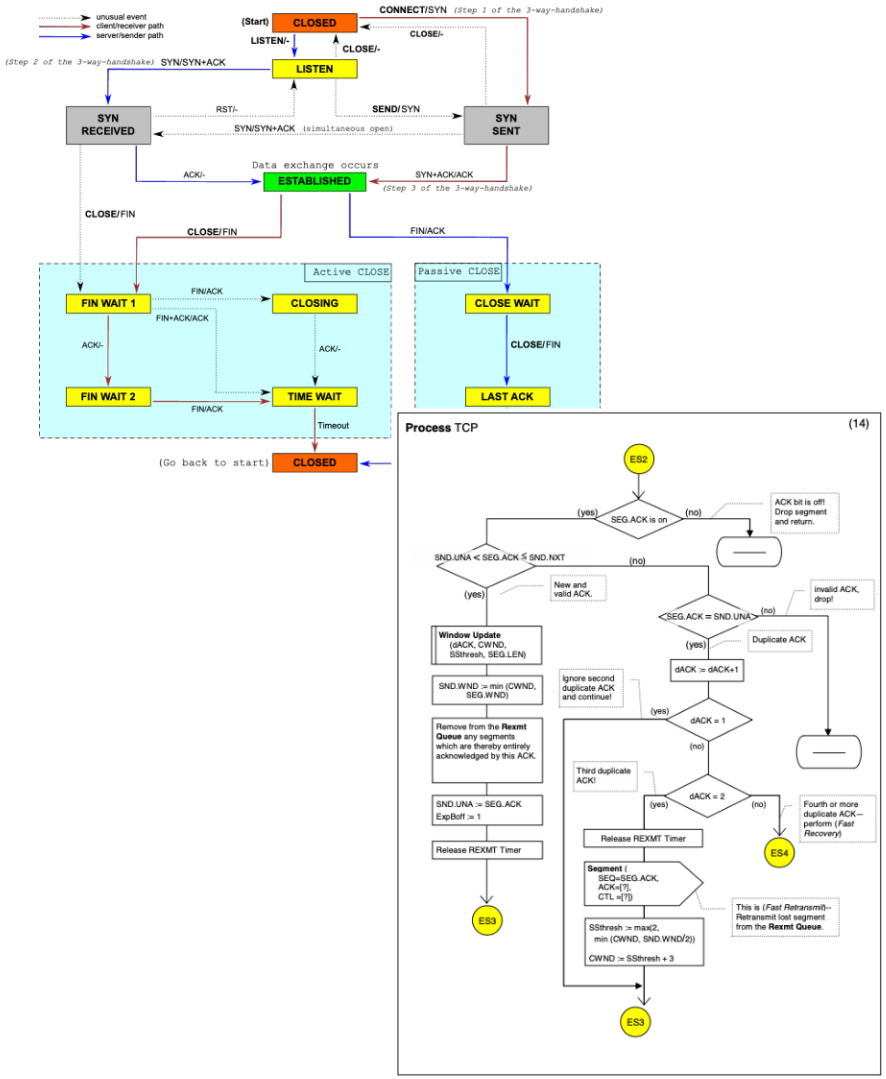
- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math

Let's Connect to a Server



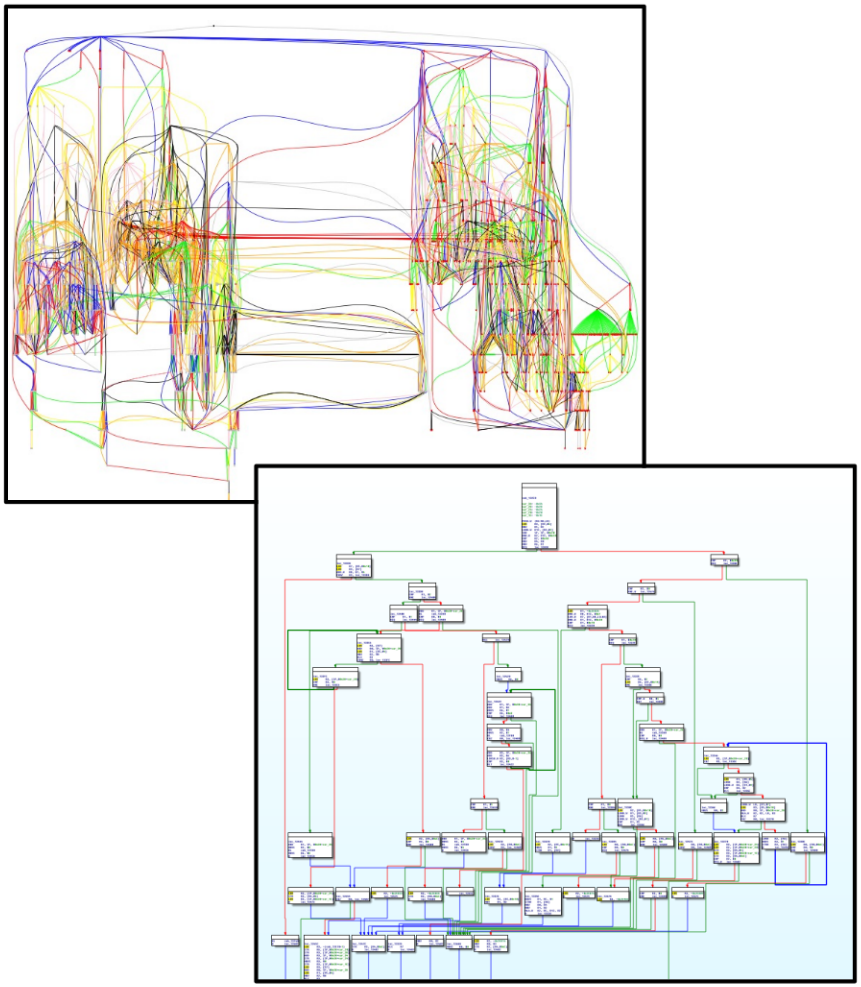
- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math
- Crypto implementation

Let's Connect to a Server



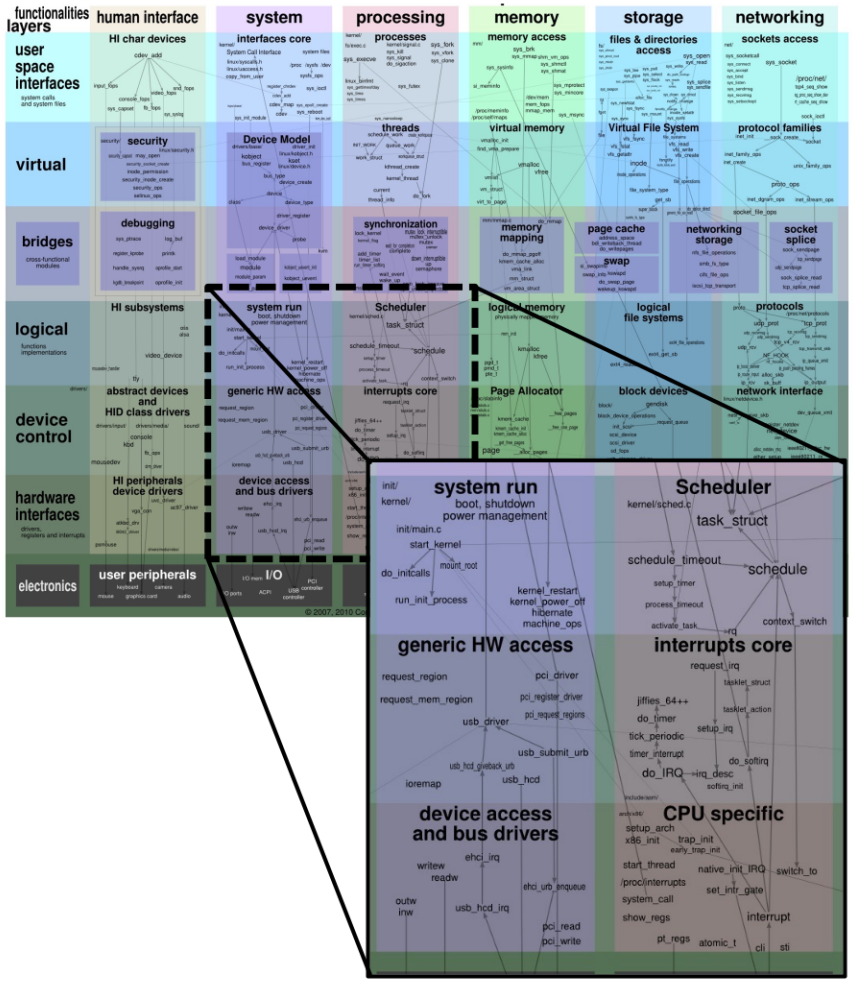
- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math
- Crypto implementation
- Channel design

Let's Connect to a Server



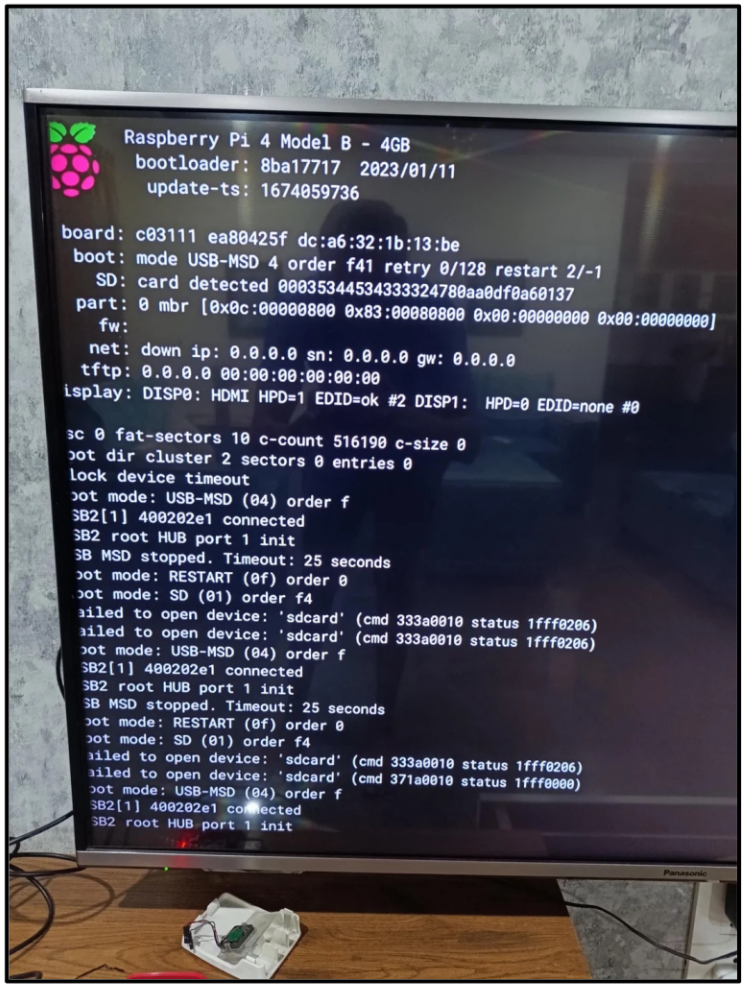
- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math
- Crypto implementation
- Channel design
- Channel implementation

Let's Connect to a Server



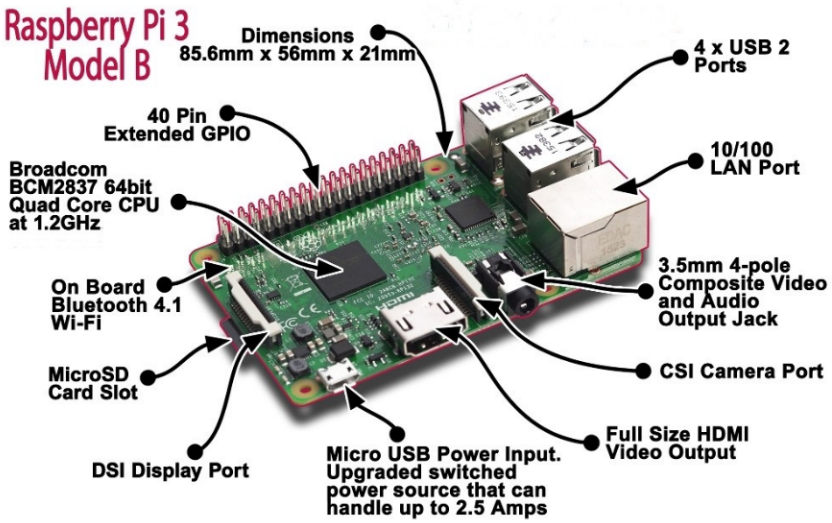
- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math
- Crypto implementation
- Channel design
- Channel implementation
- OS implementation

Let's Connect to a Server



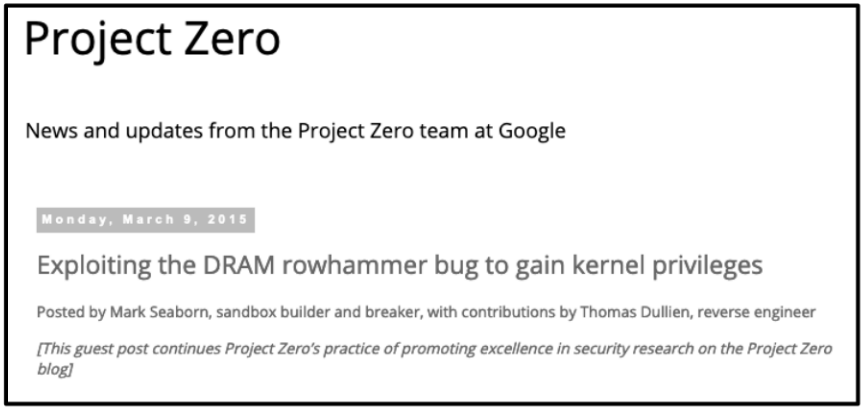
- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math
- Crypto implementation
- Channel design
- Channel implementation
- OS implementation
- Firmware for HW

Let's Connect to a Server



- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math
- Crypto implementation
- Channel design
- Channel implementation
- OS implementation
- Firmware for HW
- HW components

Rowhammer



Project Zero

News and updates from the Project Zero team at Google

Monday, March 9, 2015

Exploiting the DRAM rowhammer bug to gain kernel privileges

Posted by Mark Seaborn, sandbox builder and breaker, with contributions by Thomas Dullien, reverse engineer

[This guest post continues Project Zero's practice of promoting excellence in security research on the Project Zero blog]

ars TECHNICA SUBSCRIBE SIGN IN

RAMBLEED —

Researchers use Rowhammer bit flips to steal 2048-bit crypto key

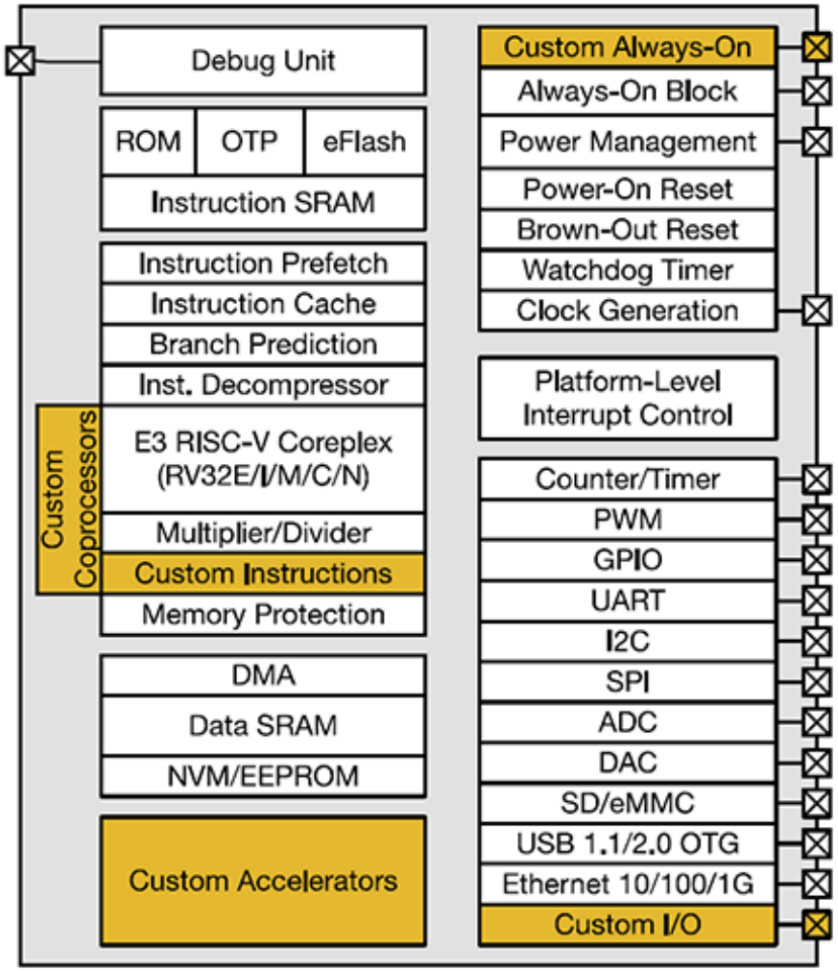
RAMBleed side-channel attack works even when DRAM is protected by error-correcting code.

DAN GOODIN - 6/11/2019, 12:00 PM

- Allows arbitrary process to flip bits in physical memory
- Predictable but not 100% perfect

Think of all the things you can do with 1 bit.

Let's Connect to a Server

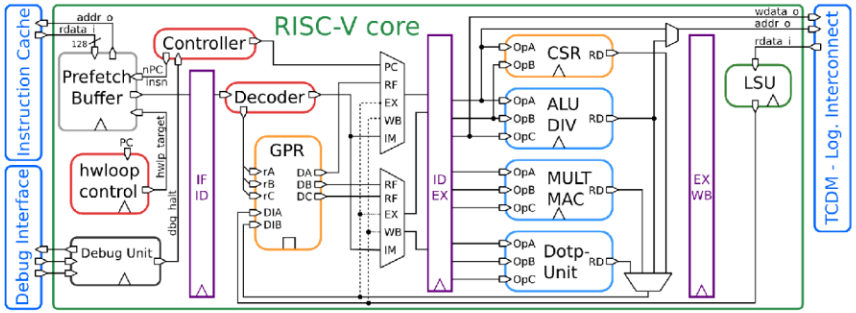


- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math
- Crypto implementation
- Channel design
- Channel implementation
- OS implementation
- Firmware for HW
- HW components
- SoC design/impl.
- Chip design/impl.

CPU Management

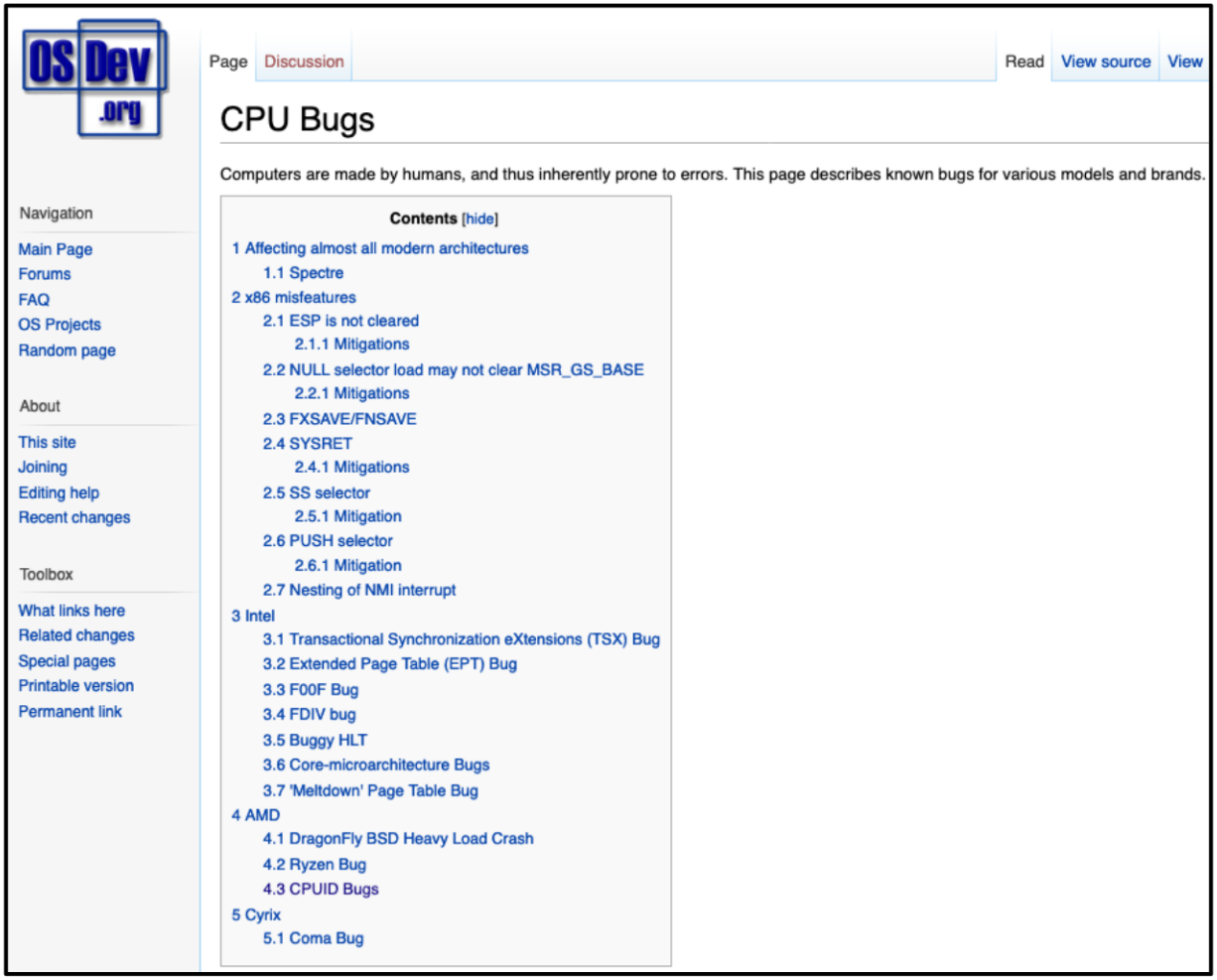
- Completely separate co-processor built into Intel CPUs
- AMD Platform Security Processor (AMD PSP)
  - All chips since 2013
- Intel Management Engine (Intel ME)
  - All chips since 2008

Let's Connect to a Server



- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math
- Crypto implementation
- Channel design
- Channel implementation
- OS implementation
- Firmware for HW
- SoC design/impl.
- CPU design/arch

CPU Bugs



OSDev.org

Page | Discussion | Read | View source | View

CPU Bugs

Computers are made by humans, and thus inherently prone to errors. This page describes known bugs for various models and brands.

Navigation

- Main Page
- Forums
- FAQ
- OS Projects
- Random page

About

- This site
- Joining
- Editing help
- Recent changes

Toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link

Contents [hide]

CPU Bug Examples

Pentium FDIV Bug

$c = 4195835 / 3145727$

appeared to be an instance of the worst-case error. Coe did his analysis without actually using a Pentium—he doesn't own one. To verify his prediction, he had to bundle his year-and-a-half-old daughter into his car, drive to a local computer store, and borrow a demonstration machine.

The true value of Coe's ratio is

$c = 1.33382044 \ldots .$

But computed on a Pentium, it is

$c = 1.33373906 \ldots .$

Cyrix Coma Bug

```
unsigned char c[4] = {0x36, 0x78, 0x38, 0x36};
int main()
{
  asm (
  "        movl    $c, %ebx\n"
  "again:  xchgl   (%ebx), %eax\n"
  "        movl    %eax, %edx\n"
  "        jmp     again\n"
  );
}
```

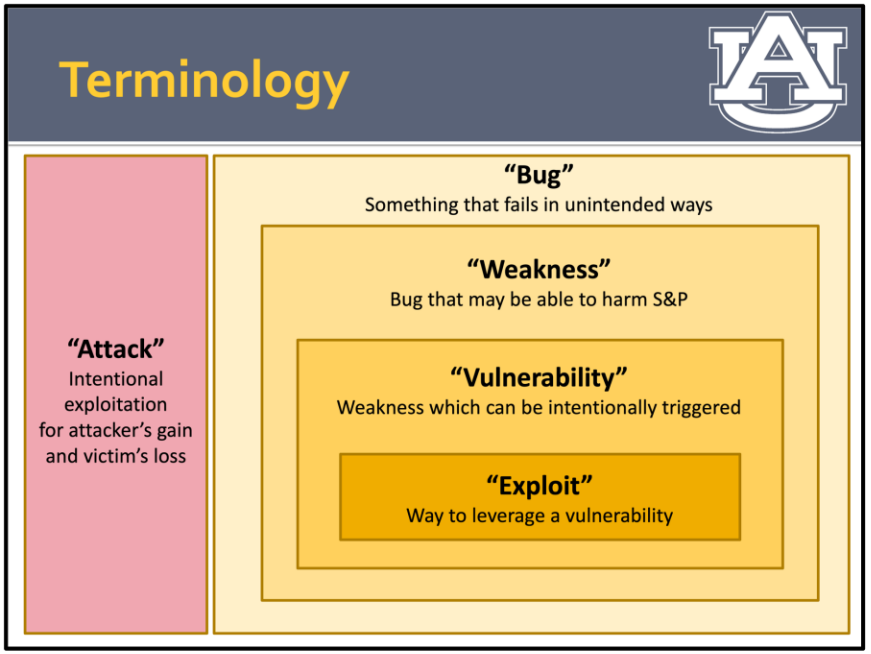
CPU Microcode

Microcode is firmware that acts as a translator for CPUs and turns *opcodes* into *micro-operations* which are executed.

- RISC won the *RISC vs. CISC War*
  - RISC-style is much more efficient
- Allows manufacturers to patch w/o recall
  - Just install a microcode patch
- **Completely unknown to almost everyone**

Software Facts of Life

- Software has bugs
- Some bugs are weaknesses
- Some weaknesses are vulnerabilities
- Some vulnerabilities can be exploited
- Someone has an interest in exploiting others for gain
- Malware is a different breed of software



Microcode Vulnerabilities

Side Channel Vulnerabilities:
Microarchitectural Data Sampling and
Transactional Asynchronous Abort

ars TECHNICA SUBSCRIBE SIGN IN

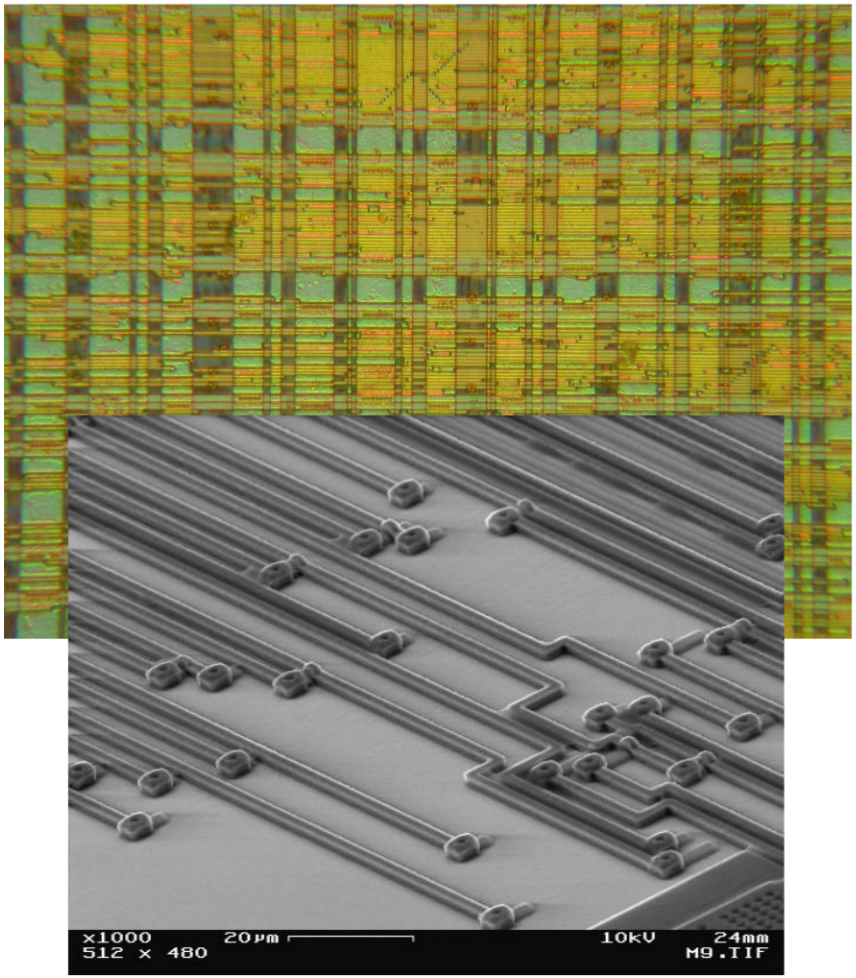
SPECULATIVE EXECUTION STRIKES AGAIN —

Intel SGX is vulnerable to an unfixable flaw that can steal crypto keys and more

Just when you thought it was secure again, Intel's digital vault falls to a new attack.

DAN GOODIN - 3/10/2020, 5:40 PM

Let's Connect to a Server

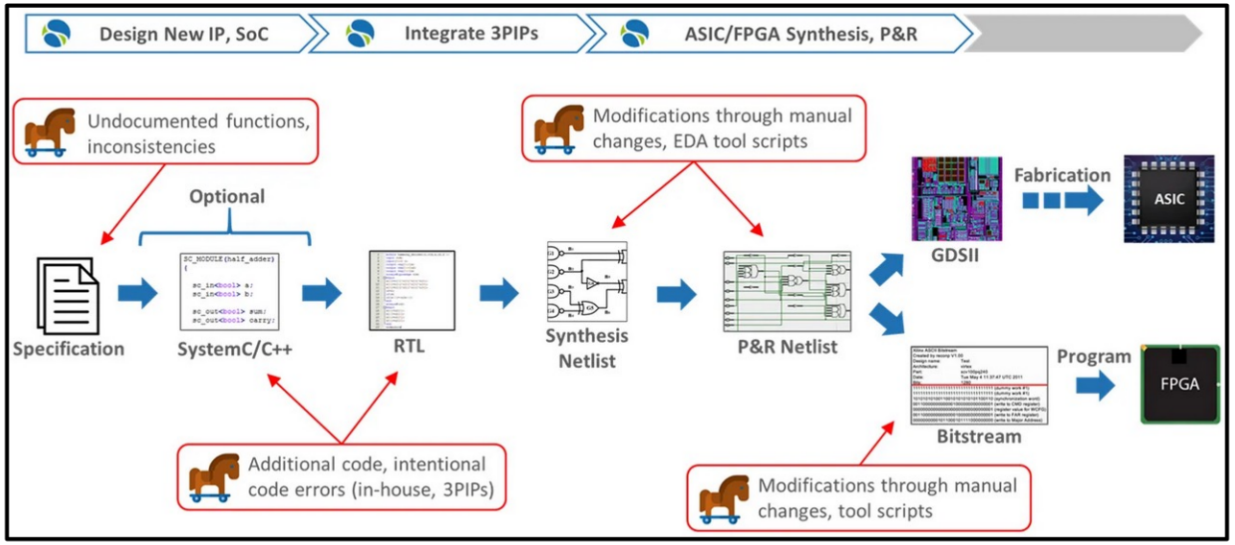


- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math
- Crypto implementation
- Channel design
- Channel implementation
- OS implementation
- HW components
- SoC design/impl.
- CPU design/arch
- Sub-component impl.

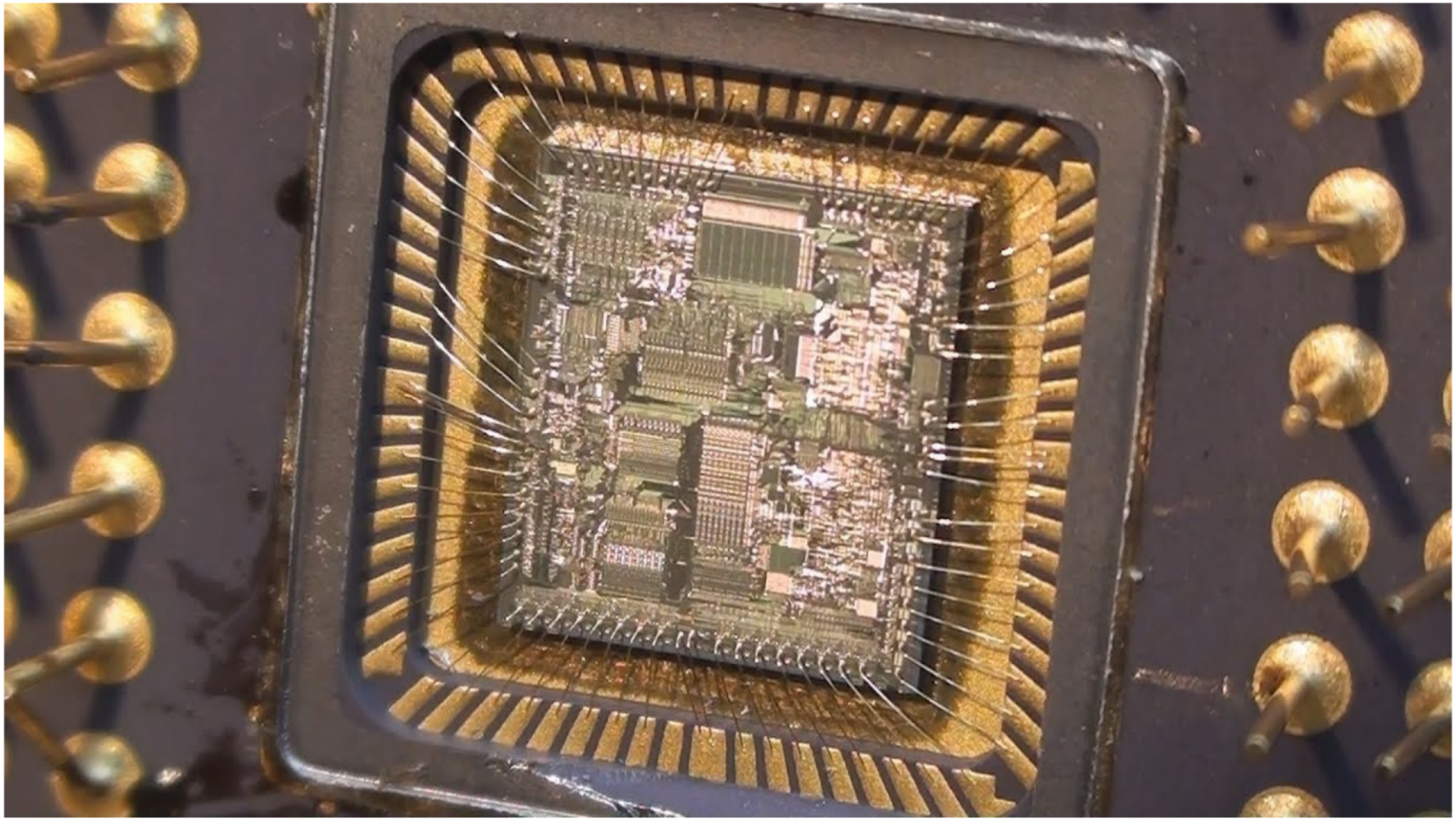
Hardware Trojans

Hardware trojans are just like trojan horse malware except implemented in hardware from the manufacturer.

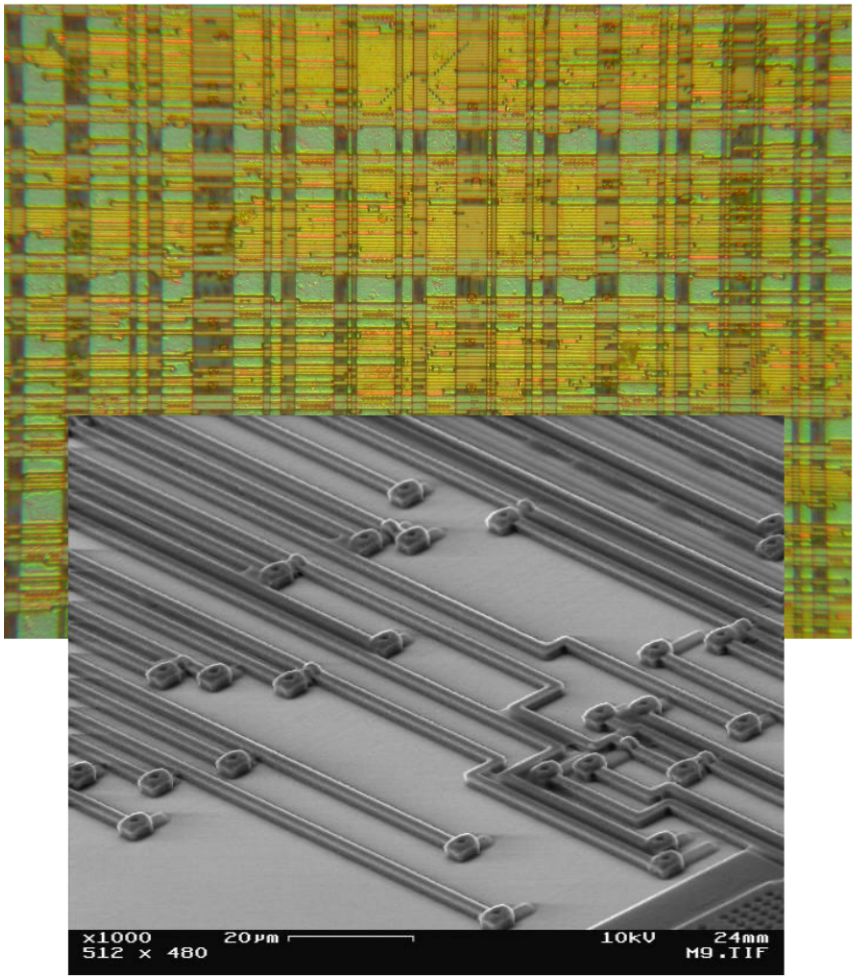
- Known capability but never seen (AFAIK)



Hardware Trojans



Let's Connect to a Server



- Bash Environment
- SSH protocol design
- SSH app architecture
- SSH app logic
- Crypto math
- Crypto implementation
- Channel design
- Channel implementation
- OS implementation
- HW components
- SoC design/impl.
- CPU design/arch
- Sub-component impl.
- Silicon traces

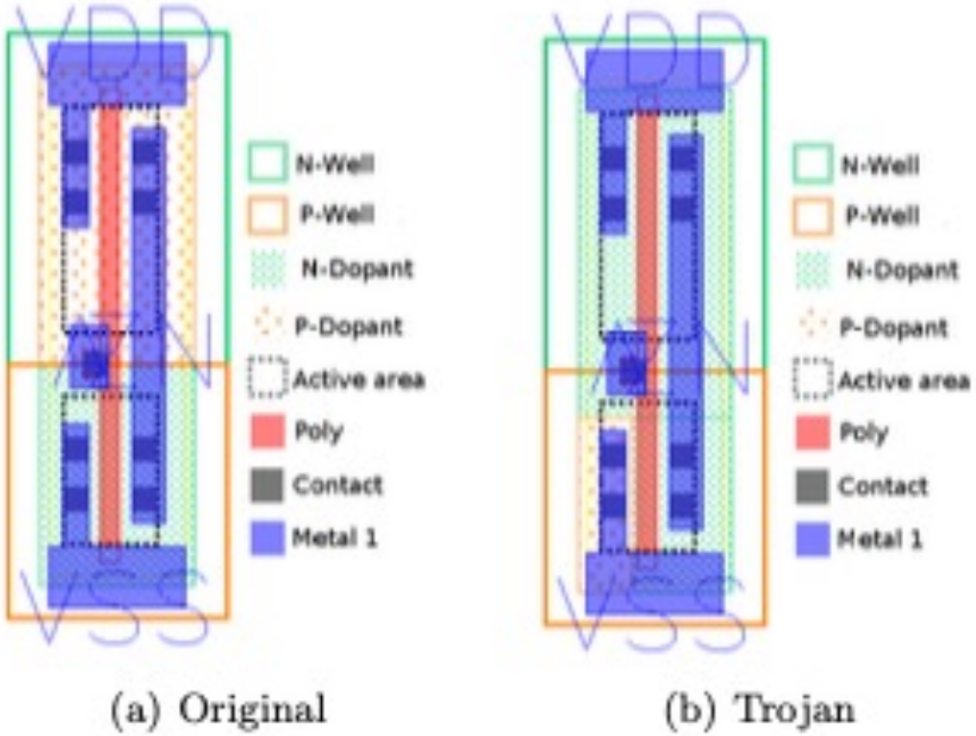
Stealthy Dopant Trojans

Stealthy Dopant-Level Hardware Trojans *

Georg T. Becker[1], Francesco Regazzoni[2], Christof Paar[1,3], and Wayne P. Burleson[1]

[1]University of Massachusetts Amherst, USA
[2]TU Delft, The Netherlands and ALaRI - University of Lugano, Switzerland
[3]Horst Görtz Institut for IT-Security, Ruhr-Universität Bochum, Germany

Stealthy Dopant Trojans



(a) Original (b) Trojan

Fig. 1. Figure of an unmodified inverter gate (a) and of a Trojan inverter gate with a constant output of V_{DD} (b).

Stealthy Dopant Trojans

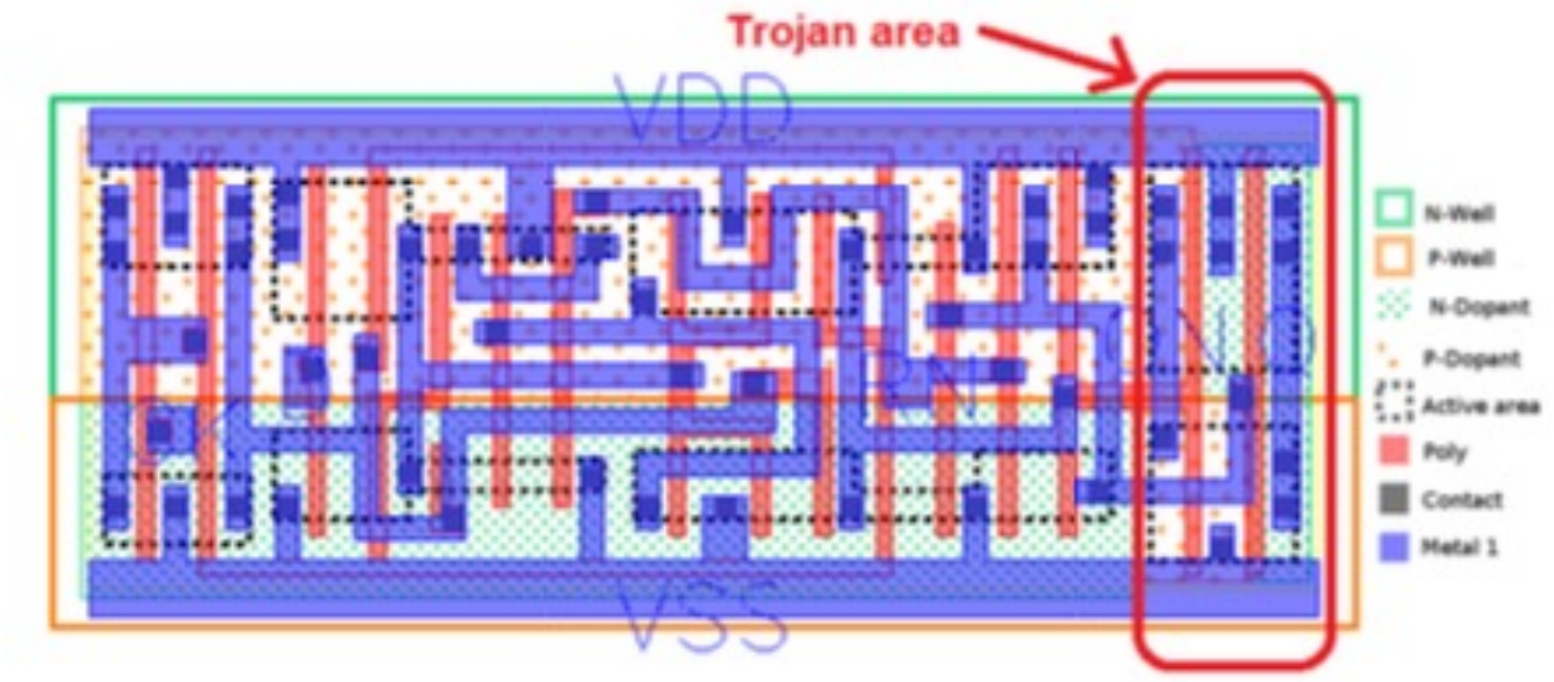
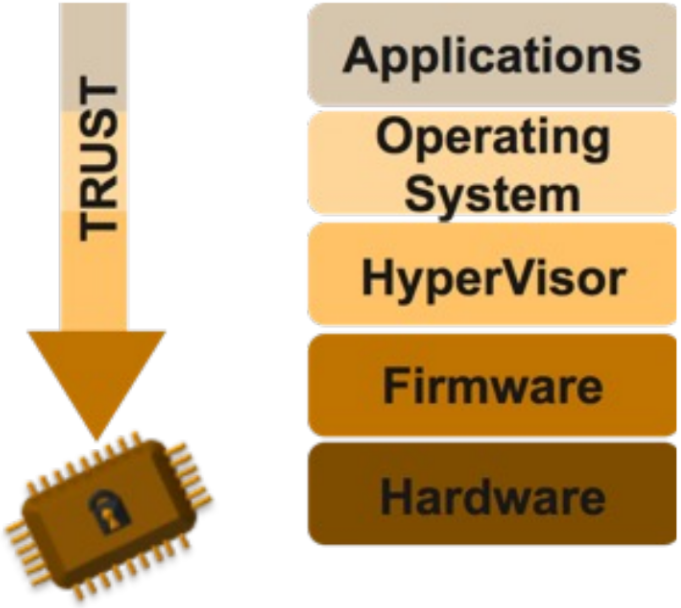


Fig. 2. Layout of the Trojan DFFR_X1 gate. The gate is only modified in the highlighted area by changing the dopant mask. The resulting Trojan gate has an output of $Q = V_{DD}$ and $QN = GND$.

Let's Connect to a Server

Trusted Computing Base

The **Trusted Computing Base (TCB)** is the collection of all components within a system critical to providing security properties.



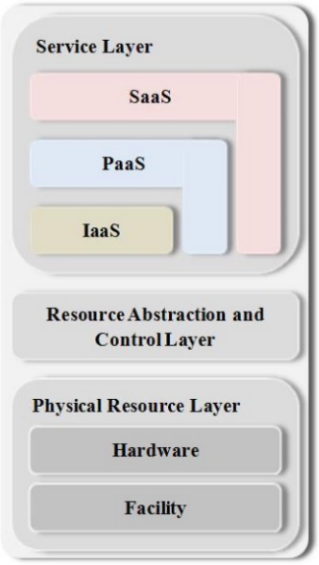


Figure 15: Cloud Provider - Service Orchestration

Hardware Security Primitives

Similar to cryptographic primitives, **Hardware Security Primitives** provide discrete functionality at the silicon-level which can be used as building-blocks.

Security CPU Instructions

AES-NI

- Perform specific AES operation in HW
  - Encrypt/Decrypt round
  - Generate round key

Instruction	Description[2]
`AESENC`	Perform one round of an AES encryption flow
`AESENCLAST`	Perform the last round of an AES encryption flow
`AESDEC`	Perform one round of an AES decryption flow
`AESDECLAST`	Perform the last round of an AES decryption flow
`AESKEYGENASSIST`	Assist in AES round key generation
`AESIMC`	Assist in AES Inverse Mix Columns
`PCLMULQDQ`	Carryless multiply (CLMUL)[3]

Security CPU Instructions

AES-NI

- Perform specific AES operation in HW
  - Encrypt/Decrypt round
  - Generate round key

Instruction	Description[2]
AESENC	Perform one round of an AES encryption flow
AESENCLAST	Perform the last round of an AES encryption flow
AESDEC	Perform one round of an AES decryption flow
AESDECLAST	Perform the last round of an AES decryption flow
AESKEYGENASSIST	Assist in AES round key generation
AESIMC	Assist in AES Inverse Mix Columns
PCLMULQDQ	Carryless multiply (CLMUL)[3]

RDRAND

- Read random value from HW and store in given register

RDRAND—Read Random Number

Opcode*/ Instruction
NFx 0F C7 /6 RDRAND r16
NFx 0F C7 /6 RDRAND r32
NFx REX.W + 0F C7 /6 RDRAND r64

RDRAND Leakage

Special Register Buffer Data Sampling (SRBDS) Hardware Vulnerability in Intel CPUs (CVE-2020-0543, aka Crosstalk)

It was discovered that special register buffer data of certain Intel CPUs may be exposed to a malicious process executing on the same CPU. Particular processor operations (e.g RDRAND, RDSEED) use data from outside the physical processor core - this can be done via an internal microarchitectural operation called a special register read. This uses part of a shared staging buffer which may not be completely zero'd on subsequent uses by other threads. As such, a local attacker may be able to infer stale values which were previously returned from special register reads to other processors (as their contents may still be present in other parts of the shared staging buffer). Some special register reads return sensitive information (such as RDRAND, RDSEED and SGX EGETKEY) and so an attacker executing code on the same CPU may be able to infer these values for another thread / process executing on the same CPU.

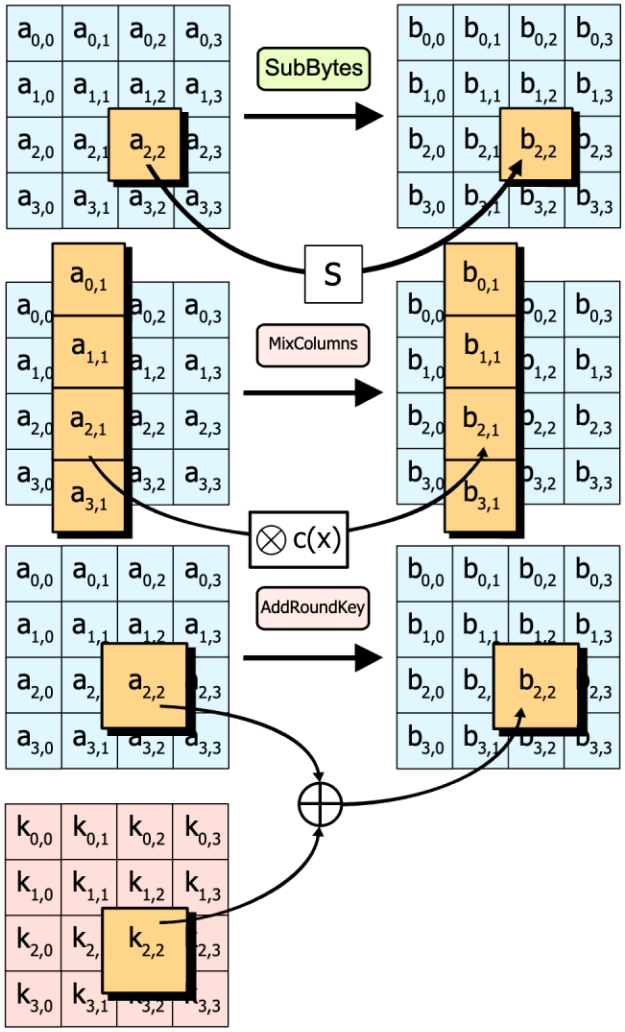
This attack relies on the same techniques used to exploit previous microarchitectural speculative execution vulnerabilities such as MDS and TSX Asynchronous Abort, and affects some client and Intel® Xeon® E3 processors; it does not affect other Intel Xeon or Intel Atom® processors.

Mitigations for this issue are provided by CPU microcode updates via the `intel-microcode` package. This mitigation consists of ensuring data in the shared staging buffer is overwritten by the RDRAND, RDSEED and EGETKEY instructions and serialising execution of RDRAND etc instructions across multiple logical processors. As a result, this will have an effect on the performance of these instructions. In conjunction, the microcode updates also supports an opt-out mechanism so that performance can be restored if desired - to support this, the Linux kernel supports a kernel command-line option `srbds=off` to allow system administrators to make use of the opt-out mechanism. For details on the boot command line option and how to check system status, please see the Linux Kernel Special Register Buffer Data Sampling Admin Guide.

Oversimplified Descriptions

- **Security CPU Instructions**
  - Trusted actions in standard hardware

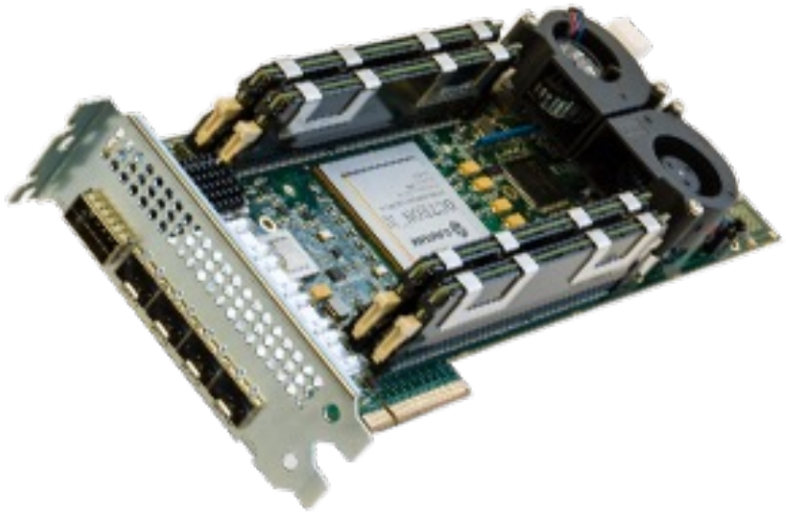
Cryptographic Accelerator



- Operations are often well-defined and repetitive
  - 14-rounds for AES256
  - Trial-and-error for bitcoin mining
  - Standardized protocols
- ASIC allows optimized pipelines for specific behavior

Cryptographic Accelerator

A **Cryptographic Accelerator** is an add-on component that allows software to leverage custom ASICs for improved performance.



Usage: Crypto Accelerator

Primitive-Level Variant

- Offload actions
- Software provides:
  - Action-specific input

    CT/PT/data/sig+data
  - Instance-specific secret
- Accelerator provides
  - Primitive algorithms
  - Action-specific output

Protocol-Level Variant

- Offload layers
- Software provides:
  - Configuration
  - Long-term secrets
- Accelerator provides:
  - Protocols negotiation
  - Primitive algorithms
  - Short-term secrets
  - Plaintext messages

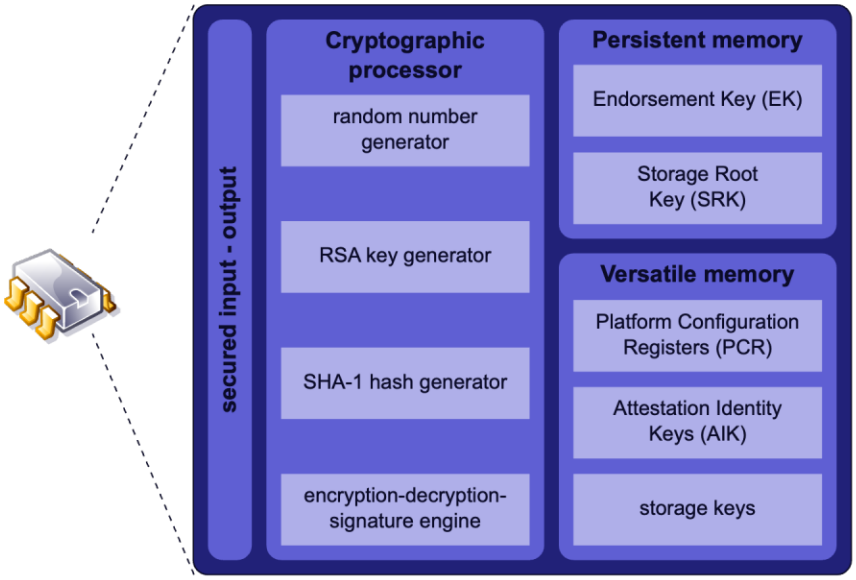
Oversimplified Descriptions

- **Security CPU Instructions**
  - Trusted actions in standard hardware
- **Crypto Accelerator**
  - Fast, trusted actions in add-on hardware

Trusted Platform Module

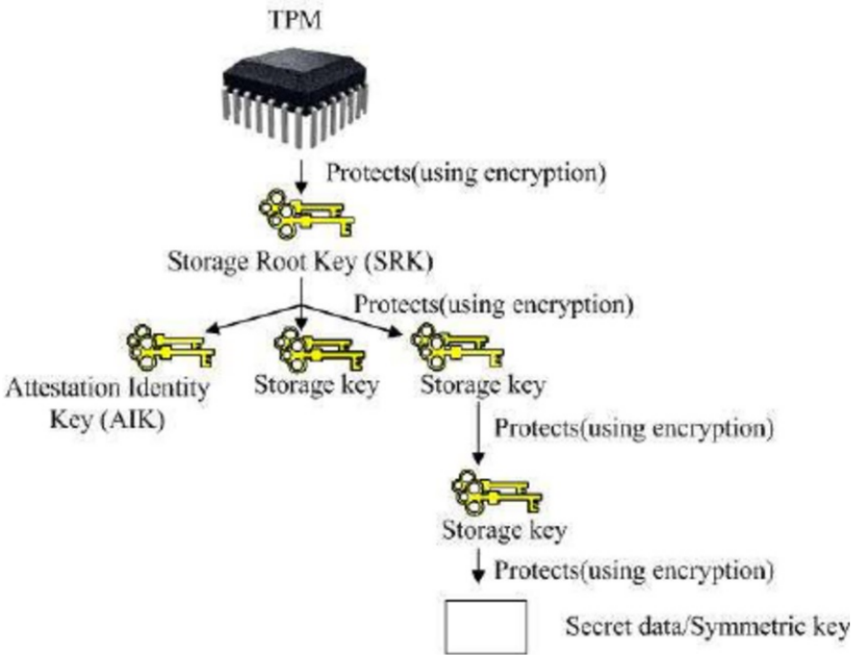
A **Trusted Platform Module (TPM)** is additional built-in, self-contained ASIC that provides a central “root of trust” for a device.

TPM Internals



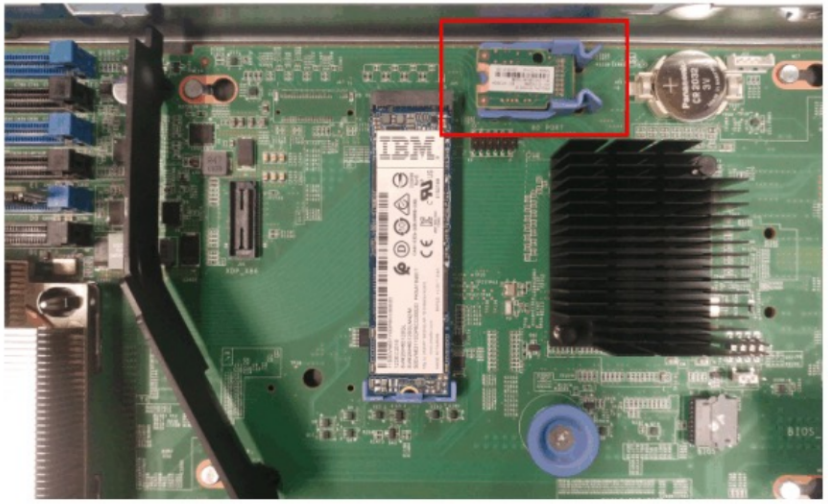
- Includes suite of crypto primitives
  - RNG
  - Algorithm implementations
  - Secure storage
- Arbitrary control logic
  - Timers, persistent counters, etc

TPM Keys



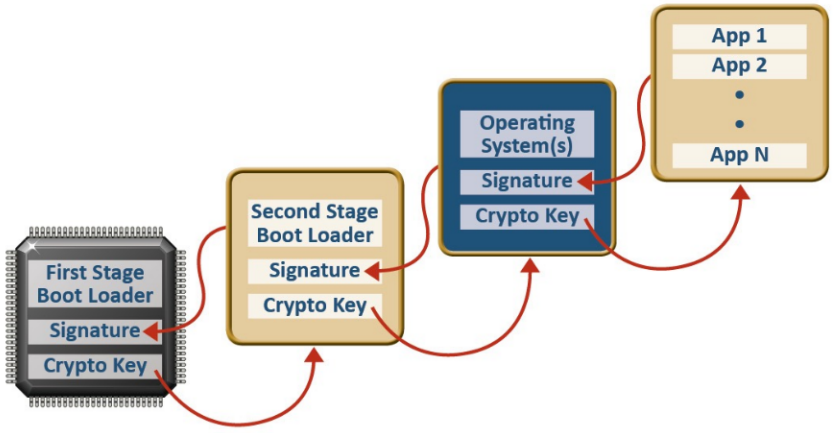
- Secrets are either generated on-board or injected during manufacturing
- Derive many secrets from single root secret via KDF

Usage: Building Block



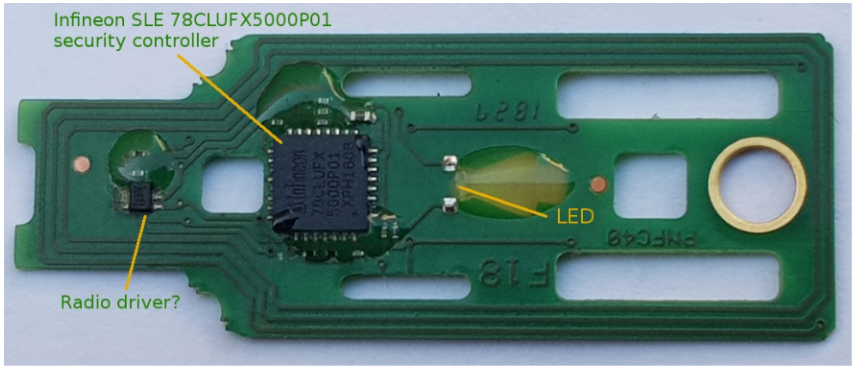
- Most commonly a component on motherboard
- Software treats as black-box operations
  - Hardened, well-defined interface for use

Example: Secure Boot



- TPM validates firmware signature before booting
- If invalid, refuse to launch bootloader
- Used as foundational trust for validating higher-level software

Usage: Out-of-Band Secret



- TPM available over removable USB
- Explicit trust boundary
- Greatly improved usability with strong security properties

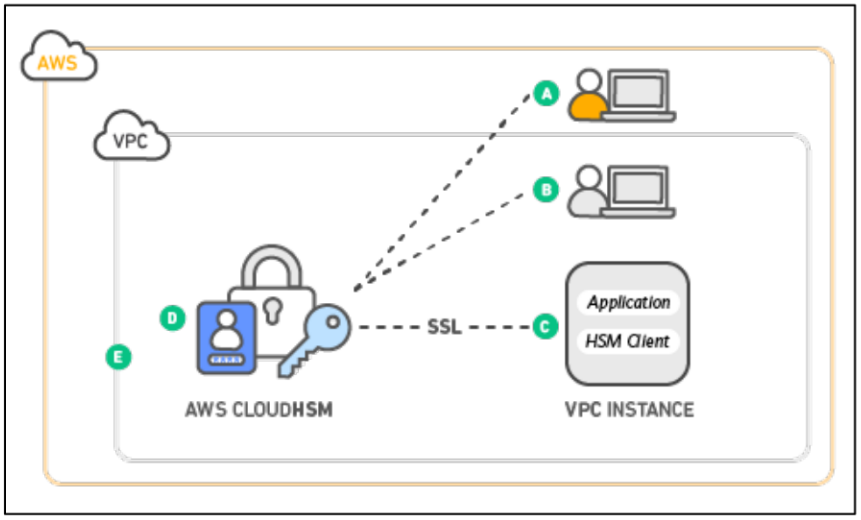
Oversimplified Descriptions

- **Security CPU Instructions**
  - Trusted actions in standard hardware
- **Crypto Accelerator**
  - Fast, trusted actions in add-on hardware
- **Trusted Platform Module**
  - Trusted actions in built-in hardware w/ keys

Hardware Security Module

A **Hardware Security Module (HSM)** is a special-purpose add-on component that securely stores cryptographic keys and performs cryptographic operations.

Hardware Security Module



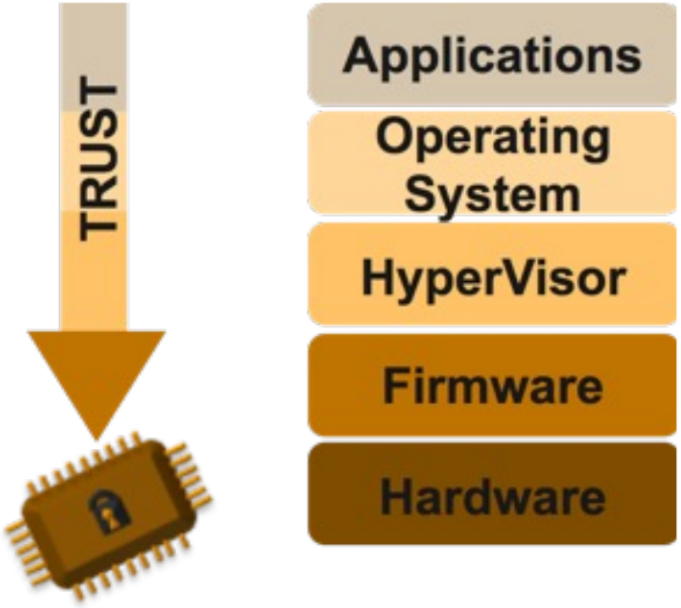
- High-performance operations
- Restricted logic
  - Most commonly used for signing operations
- Commonly available “in the cloud” for use with AWS/GCP/…

Oversimplified Descriptions

- **Security CPU Instructions**
  - Trusted actions in standard hardware
- **Crypto Accelerator**
  - Fast, trusted actions in add-on hardware
- **Trusted Platform Module**
  - Trusted actions in built-in hardware w/ keys
- **Hardware Security Module**
  - Fast, trusted actions in add-on hardware w/ keys

Trusted Computing Base

The **Trusted Computing Base (TCB)** is the collection of all components within a system critical to providing security properties.



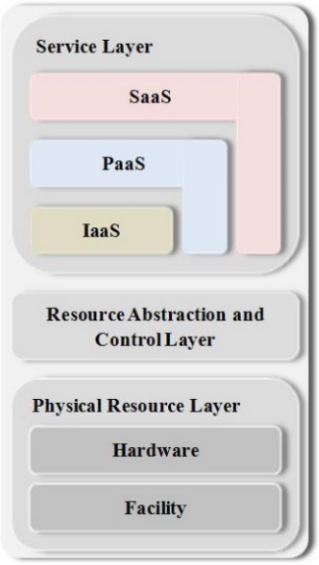
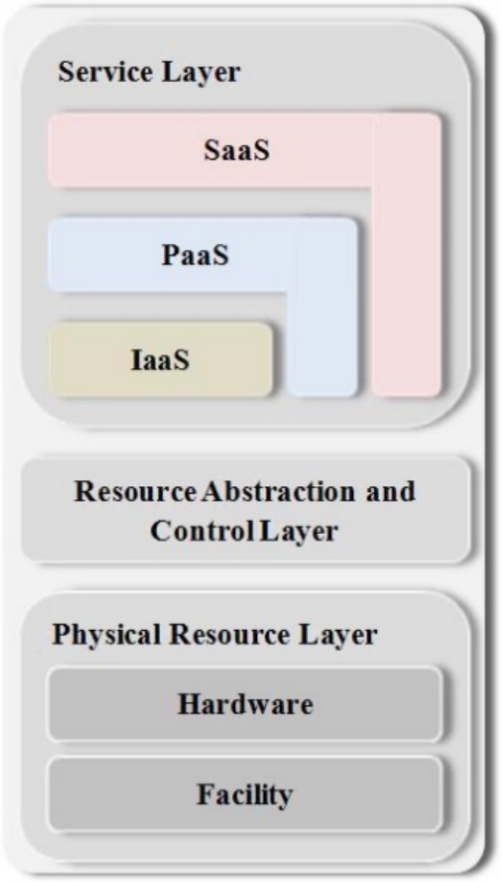


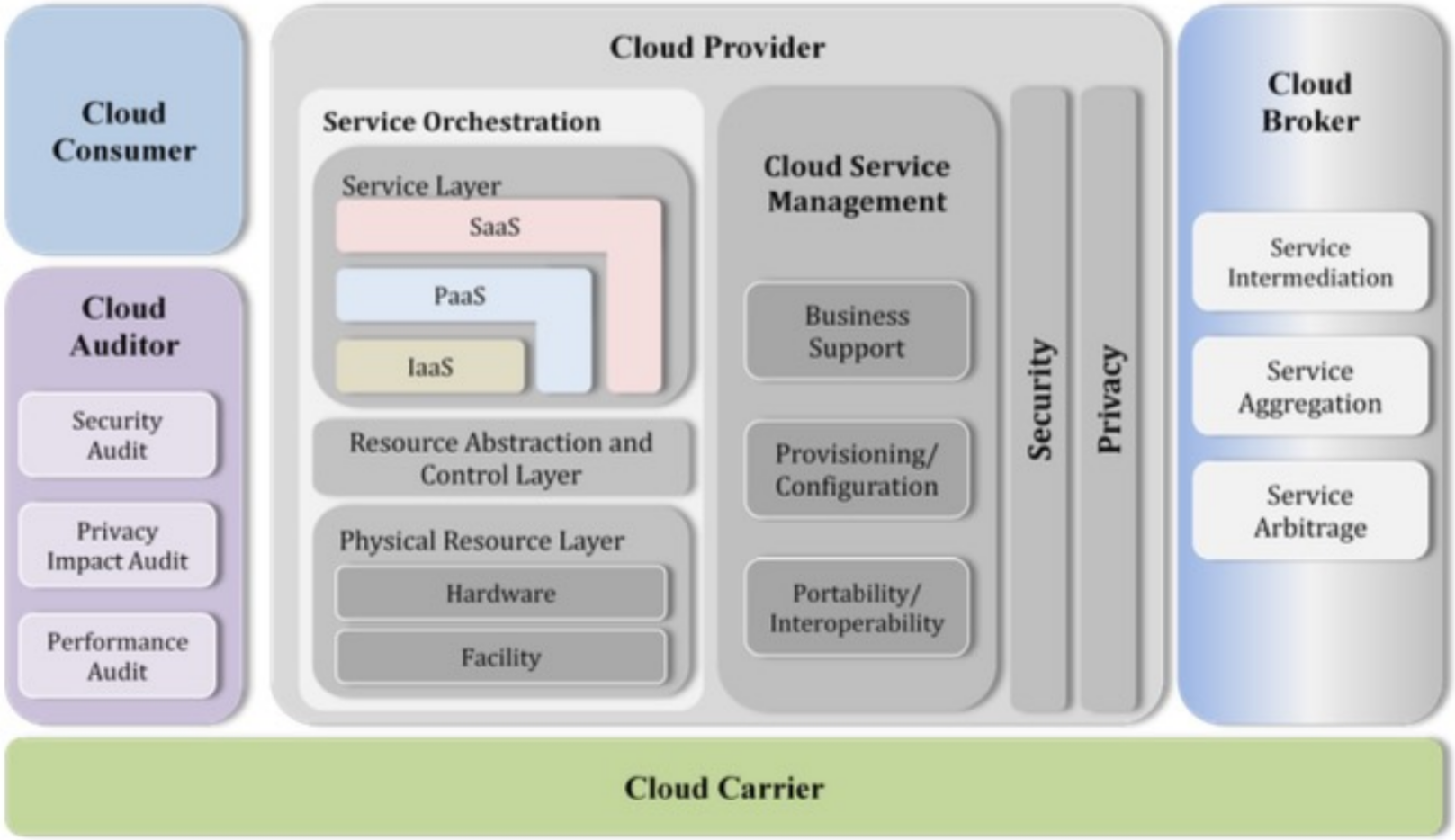
Figure 15: Cloud Provider - Service Orchestration

Cloud Provider TCB



- SaaS: Software
  - Office 365
- PaaS: Platform
  - Elastic Container Service
- IaaS: Infrastructure
  - EC2 Instances
- <many more layers of internal services>
- **All on top of Local TCB**

Cloud Computing Architecture TCB



Computer and Network Security

Lecture 14:
OS & Hardware Security

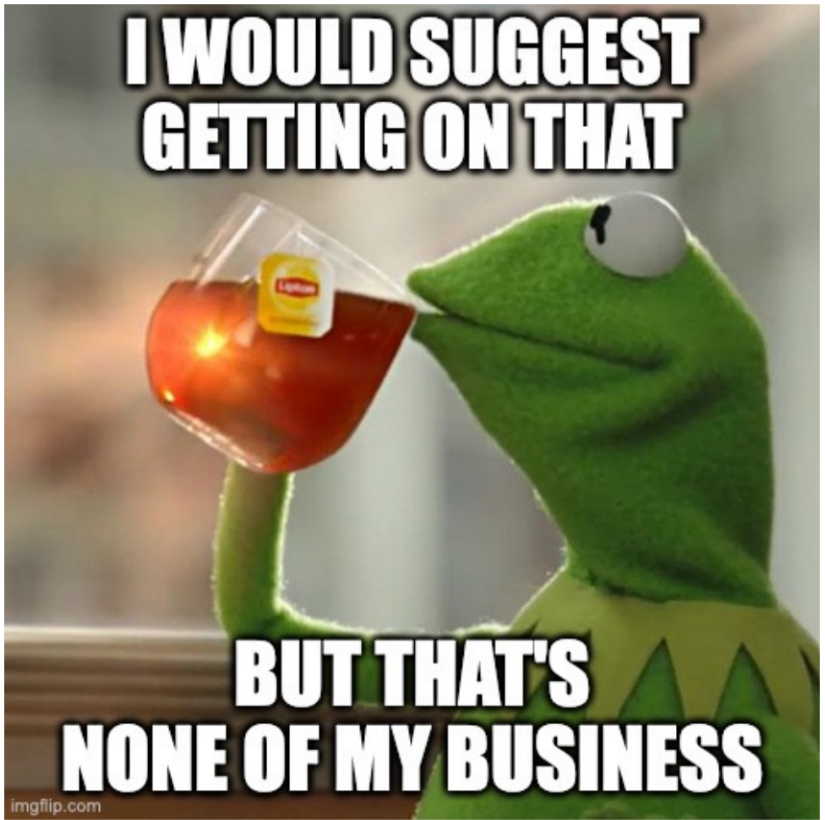
COMP-5370/6370
Fall 2025

Course Notes

- Project 1A grades posted
- Exam 1 grades posted
- **Midterm on Tuesday**
  - **Distance: Identify proctor and schedule**
- Same style/format/approach as Exam 1
  - Multiple choice, True/False, Matching, etc.
  - Short-answers for grad-section
  - Bonus available but low point value

Course Notes

- Project 2 due **next Friday** (10Oct2025)
  - If you haven't started…



Computer and Network Security

Lecture 14:
OS & Hardware Security

COMP-5370/6370
Fall 2025