
HITB
Hardware Security is Hard:
How Hardware Boundaries Define Platform Security
Alex Matrosov
Chief Offensive Security Researcher, NVIDIA
002
HITB LOCKDOWN
livestream

Offensive Security REsearch at NVIDIA

Previously Principal Security Researcher @Cylance @Intel @ESET

Doing Security REsearch since 1997

https://www.platformsecuritysummit.com/2019/speaker/matrosov/

https://www.platformsecuritysummit.com/2019/speaker/matrosov/

HW THREAT MODEL
OR
FW THREAT MODEL

NOWADAYS IT'S HARD TO SPOT REAL BOUNDARIES BETWEEN HARDWARE AND FIRMWARE

HW THREAT MODEL
AND
FW THREAT MODEL

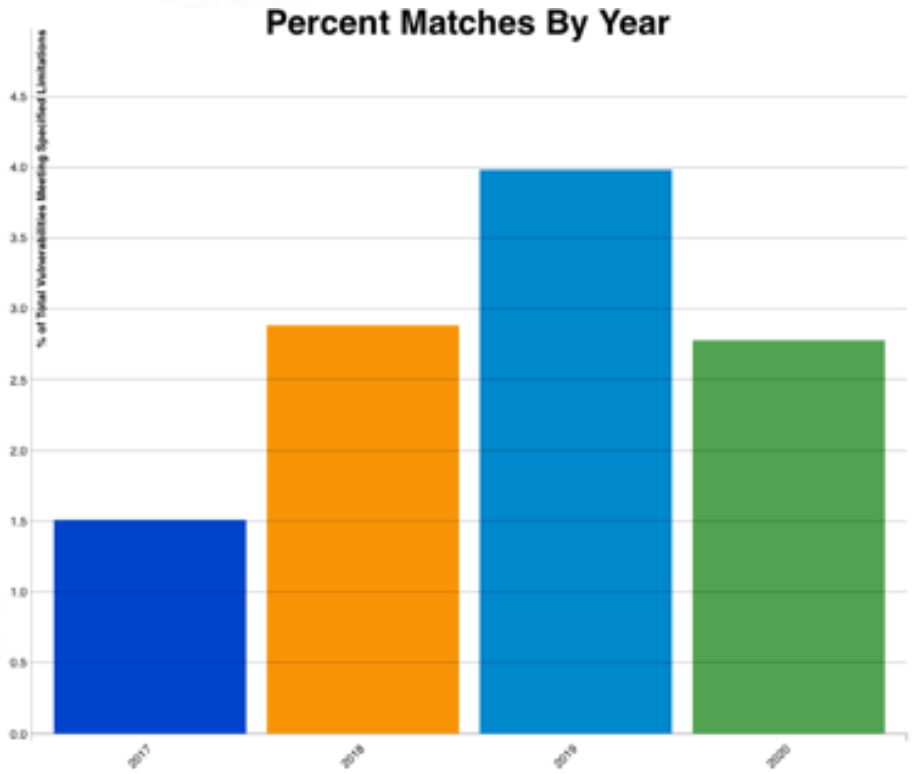
Identify Threats

- Threats can be identified by analyzing the security requirements and platform diagram
- Threats should be categorized for further analysis
  - Techniques for analyzing threats: STRIDE, DREAD, PASTA, LINDDUN, etc.

https://uefi.org/sites/default/files/resources/UEFI%20SDL%20Webinar_Final%20Slides%20-%20PDF.pdf

IT'S HARD TO FIND REAL SECURITY PROBLEMS IN PLATFORM DIAGRAM BASED ONLY ON REQUIREMENTS

Numbers of reported issues related to FW significantly increasing every year!



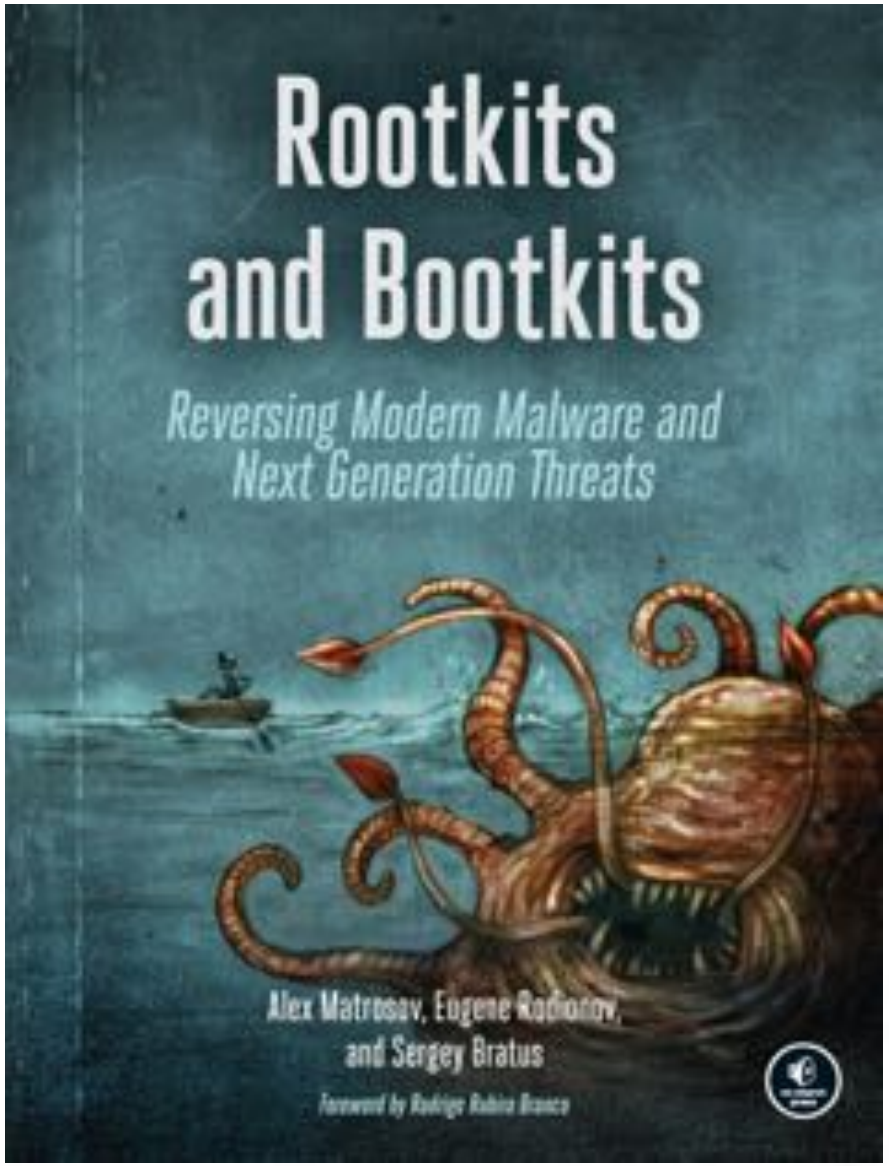
https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&query=firmware&search_type=last3years

Security Through Obscurity

- Firmware binaries are freely available online
- Tools to analyze binaries are available
- Security researchers are decompiling binaries
  - Most 3rd party reports received include disassembled code

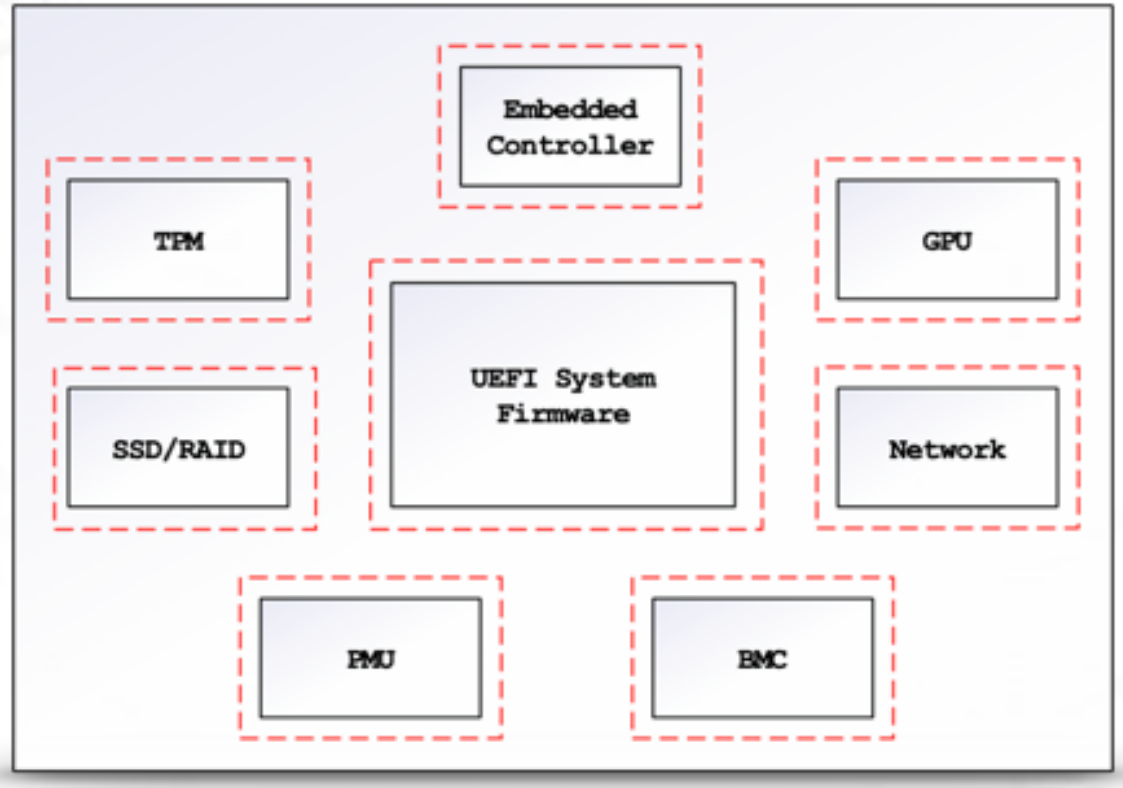
not a binaries, technology specs please

https://uefi.org/sites/default/files/resources/UEFI%20SDL%20Webinar_Final%20Slides%20-%20PDF.pdf


Rootkits
and Bootkits
Reversing Modern Malware and
Next Generation Threats
Alex Matrosov, Eugene Rodionov,
and Sergey Bratus
Foreword by Rodrigo Rubira Branco


Security Architect

HW/FW Security != sum of all Boundaries



Hardware Security Boundaries

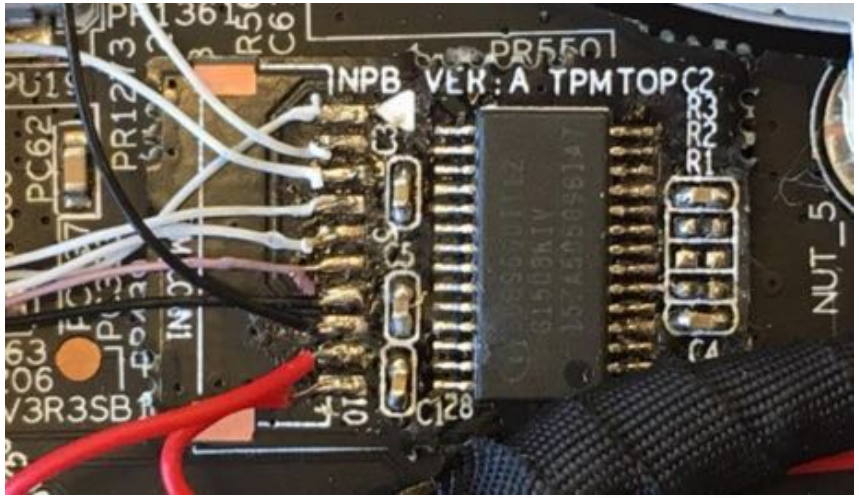
Most of those chips are:

- Not under direct control from hardware vendors
- Involved in security features implementation
- Connected to UEFI firmware (BIOS)
- Considered to generate trusted I/O
- Mostly out of the supervision scope of the main CPU

How can we trust anything that is not under our system control!

IN CURRENT REALITIES HW AND FW THREAT MODELING SHOULD BE UNDIVIDED

In current threat model HW considered trusted!



BMC is inside trusted boundaries 😈

- UEFI firmware blindly trust most of the hardware
- But hardware can attack UEFI firmware

https://airbus-seclab.github.io/ilo/ZERONIGHTS2018-Slides-EN-Turning_your_BMC_into_a_revolving_door-perigaud-gazet-czarny.pdf

How many 3rd-party chips in your platform?

TPM module

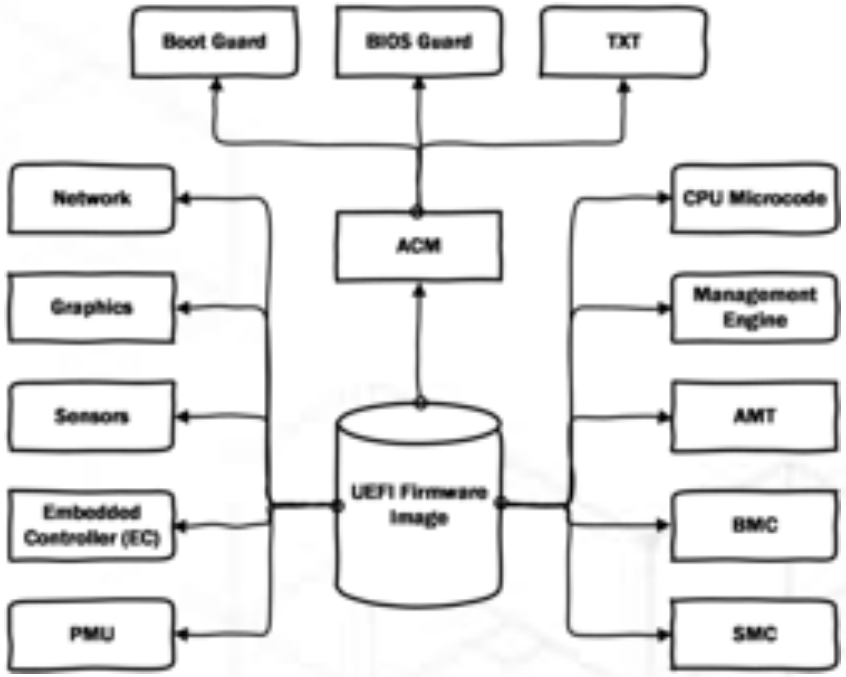
USB controller

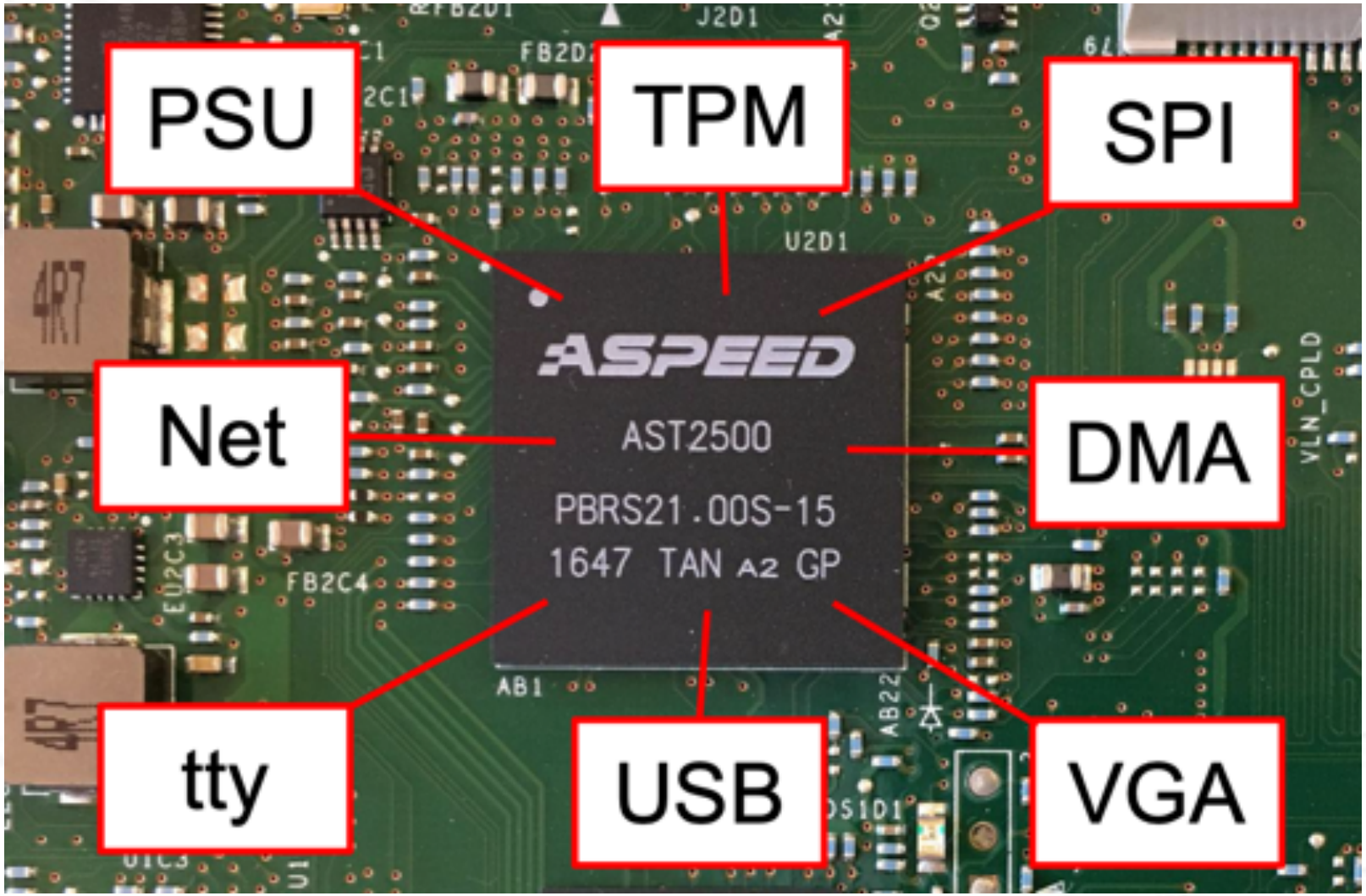
Embedded Controller (EC)

Fingerprint Reader

Touchpad

and many others





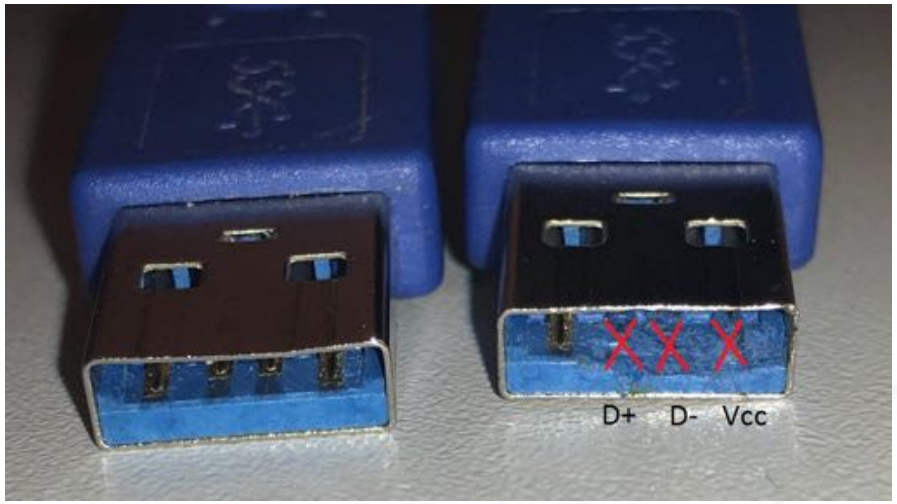
https://media.hardwear.io/roots-of-trust-and-attestation/

ASD - AT-SCALE DEBUG

AMI also offers solution for At-scale remote debugging solution via MegaRAC BMC and AMI Hardware Debugger. AMI Hardware Debugger communicates over network with BMC to perform Host debugging operation.

https://ami.com/ami_downloads/AMI_Hardware_Debugger_Data_Sheet.pdf

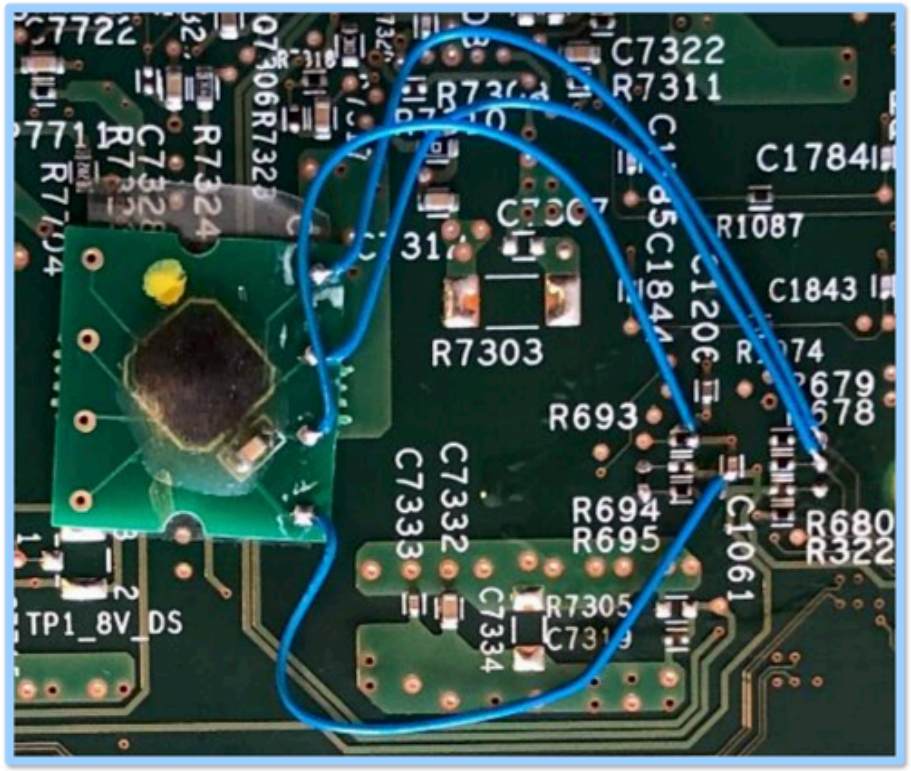
Intel DCI debug commoditized by recent research



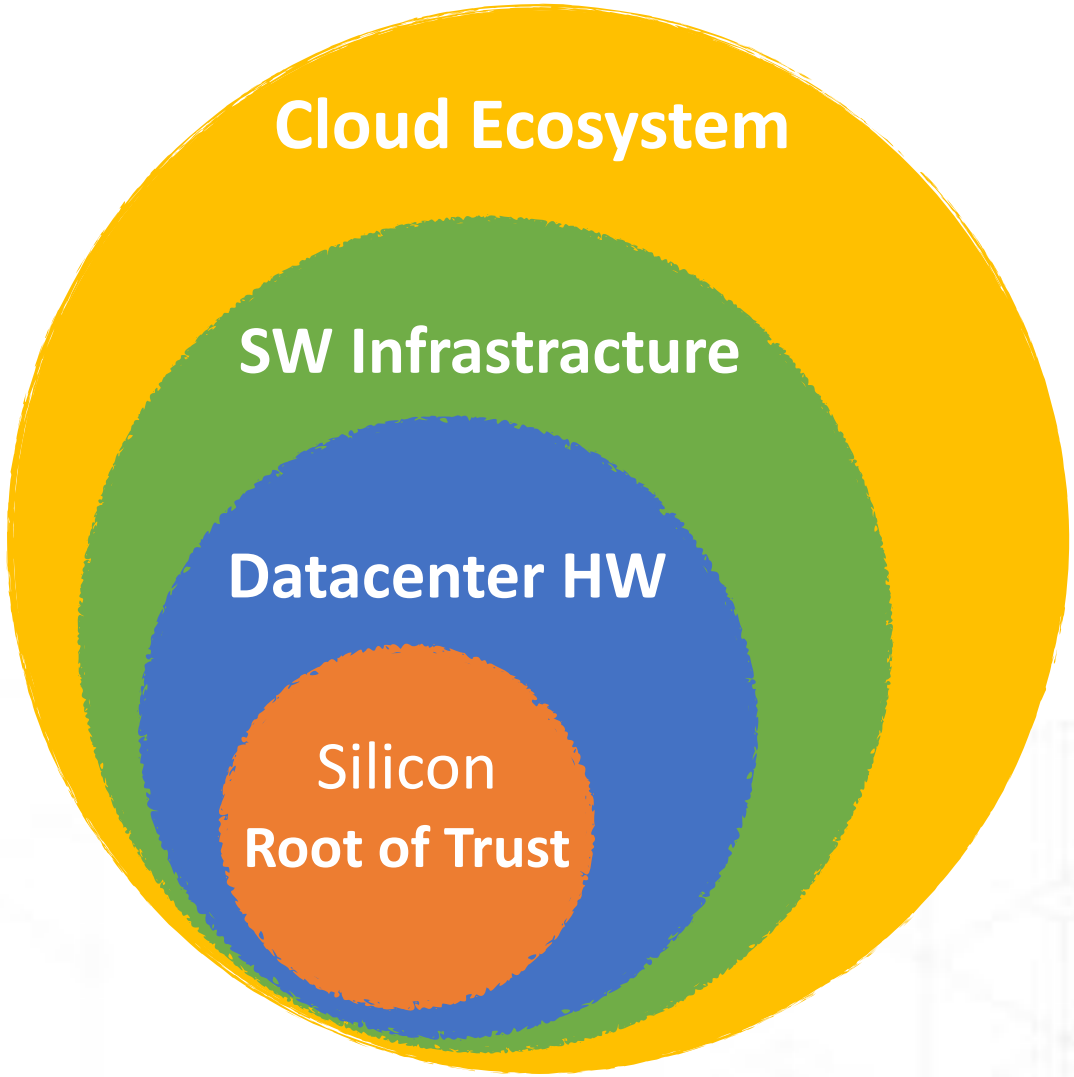
https://github.com/ptresearch/IntelTXE-PoC

Counterfeit devices with HW implants not rare

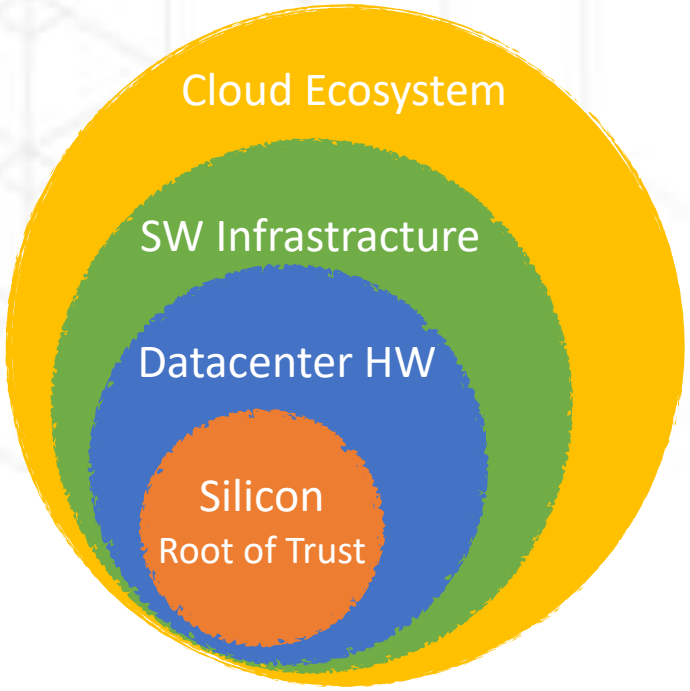
https://www.youtube.com/watch?v=YFE4RqHRYOA

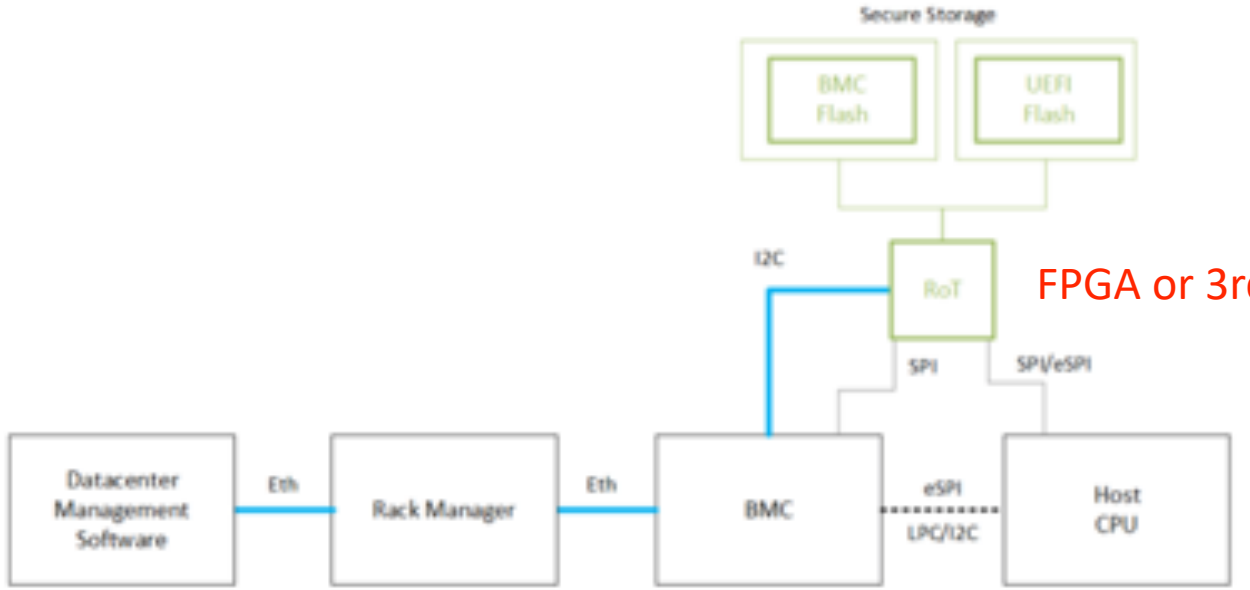


https://labs.f-secure.com/assets/BlogFiles/2020-07-the-fake-cisco.pdf


Cloud Ecosystem
SW Infrastracture
Datacenter HW
Silicon
Root of Trust

How external RoT on “Secure” chip can help?





FPGA or 3rd party chip

https://github.com/opencomputeproject/Project_Olympus/tree/master/Project_Cerberus

Lenovo Thinkpad EC update process

OS

Lenovo TDK update tool

map EC update image to memory	set NVRAM var 'LenovoEcfwUpdate'

```
while ( v7 - &LenovoEcfwUpdate <= v5 );
memset_(buffer, 0, 1u);
buffer[0] = 1;
TdkBinCreateFromBuff(buffer, 1ui64, &tdk_bin);
result = TdkVariableSet(&a1, &a2, 7u, tdk_bin);
```

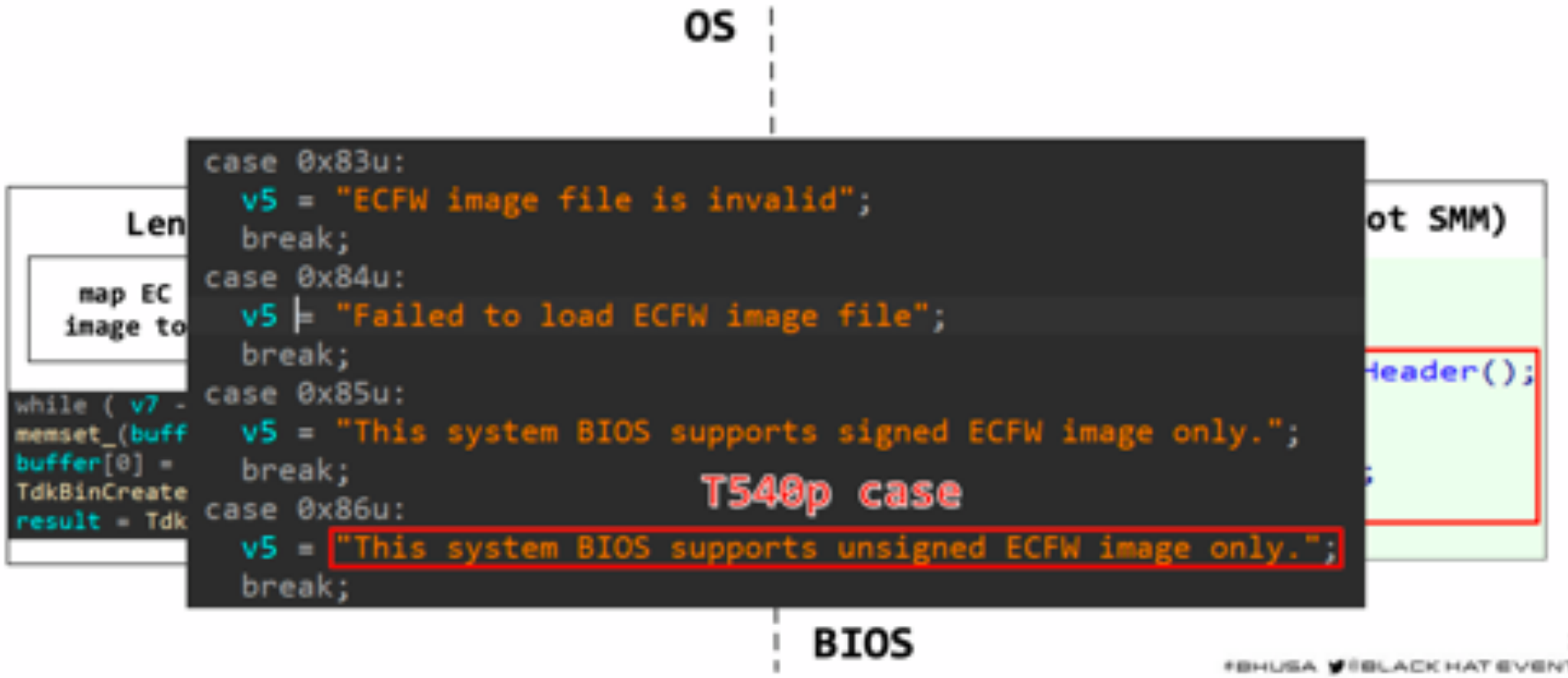
Lenovo EcFwUpdateDxe (not SMM)

```
res = LoadFirmware();
if ( res >= 0 )
{
  res = ValidateFirmwareHeader();
  if ( res >= 0 )
  {
    UpdateEcFw(ecfw_bin);
    res = 0i64;
  }
```

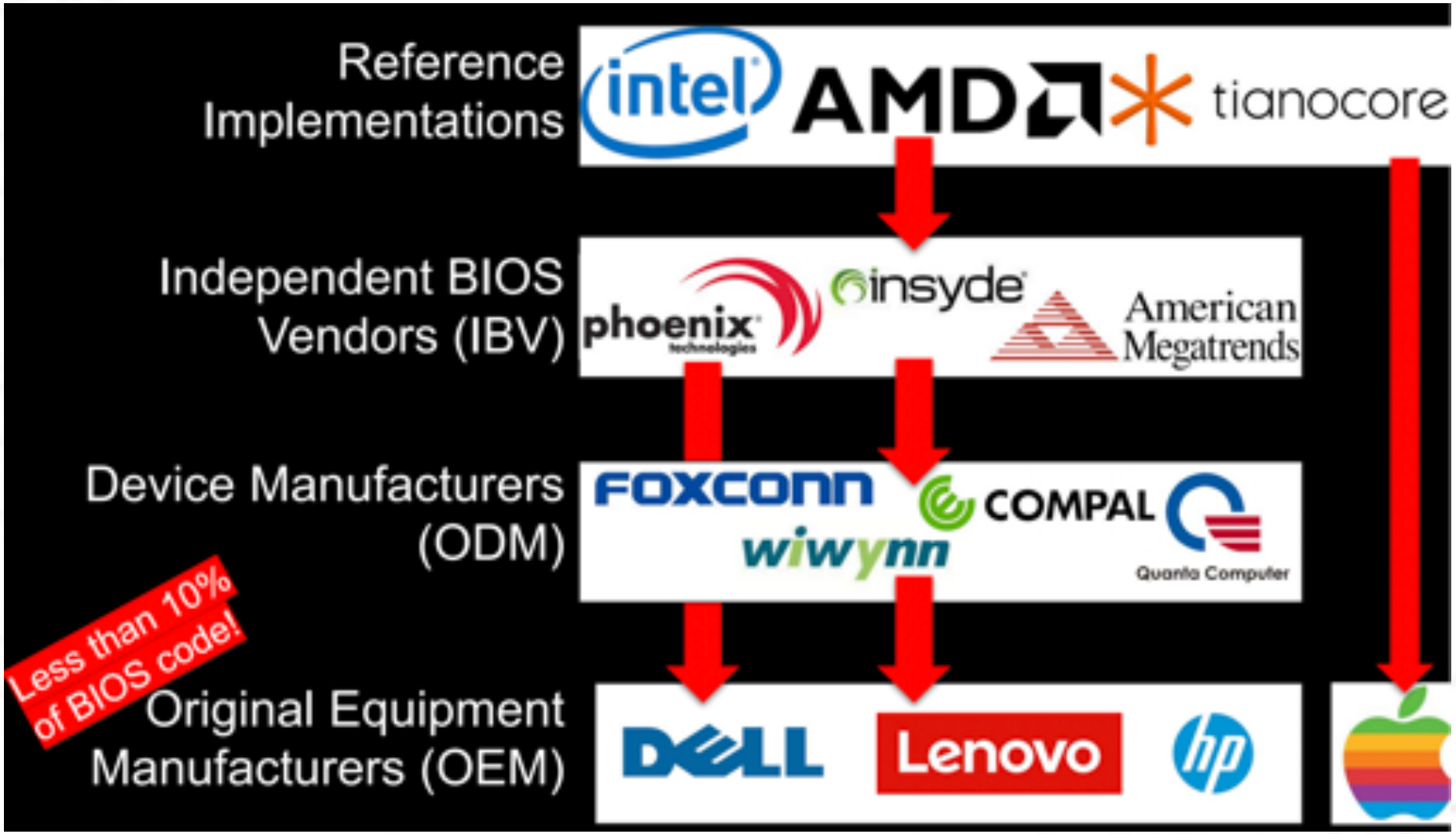
BIOS

https://medium.com/@matrosov/breaking-through-another-side-bypassing-firmware-security-boundaries-85807d3fe604

Lenovo Thinkpad EC update process



https://medium.com/@matrosov/breaking-through-another-side-bypassing-firmware-security-boundaries-85807d3fe604



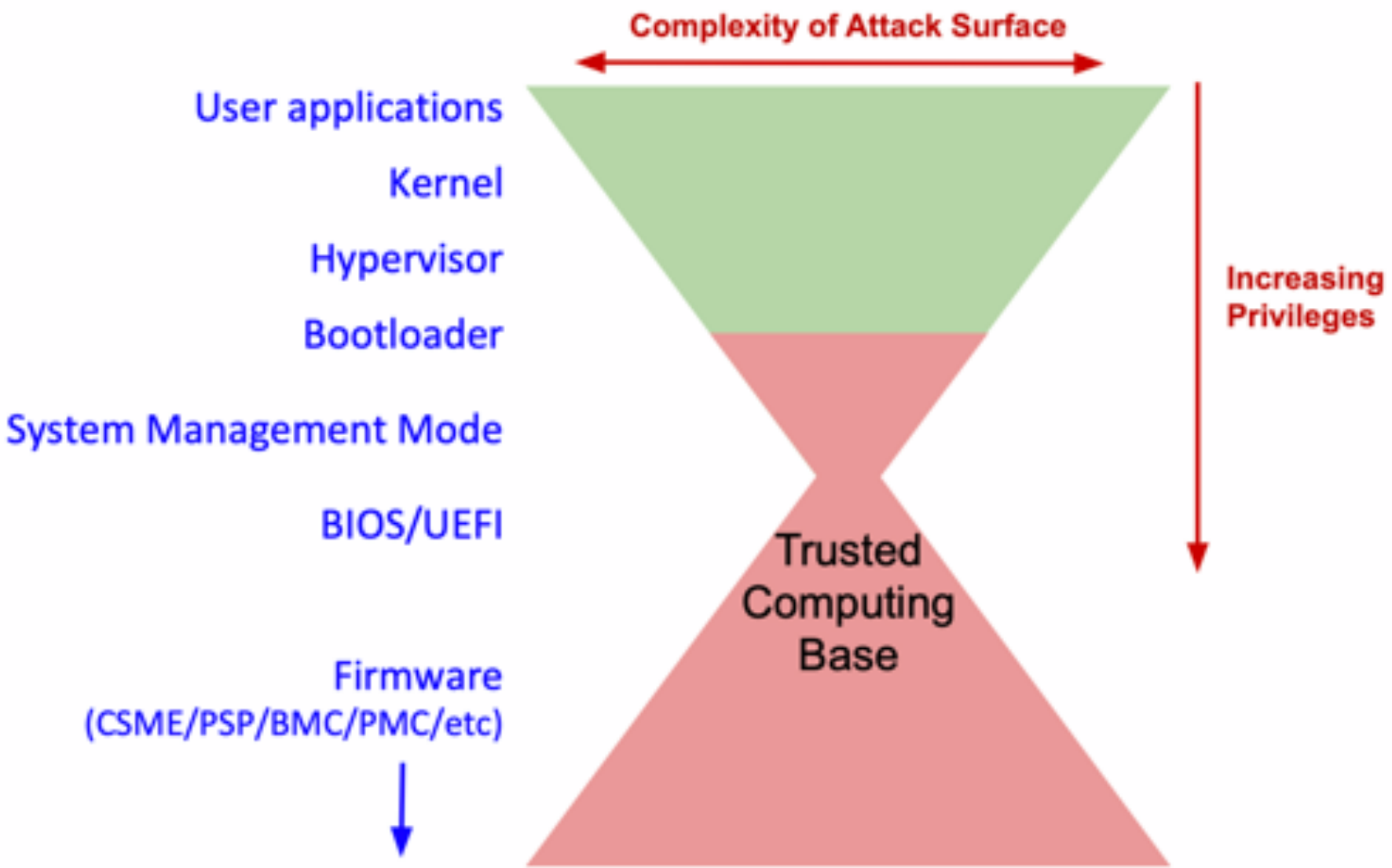
https://media.hardwear.io/roots-of-trust-and-attestation/

Recently we released efiXplorer plugin to make UEFI Firmware RE more enjoyable

```
loc_1149E:
mov     rax, cs:qword_F0428
lea     r8, unk_F0480
xor     edx, edx
lea     rcx, EFI_HII_FONT_PROTOCOL_GUID_8f998
call    qword ptr [rax+140h] ; gBS->LocateProtocol()
                        ; EFI_STATUS(EFIAPI * EFI_LOCATE_PROTOCOL) (IN EFI_GUID *Protocol, IN VOID *Registration, OPTIONAL OUT VOID **Interface)
                        ; Protocol       Provides the protocol to search for.
                        ; Registration   Optional registration key returned from RegisterProtocolNotify().
                        ; Interface      On return, a pointer to the first interface that matches Protocol and Registration.
```

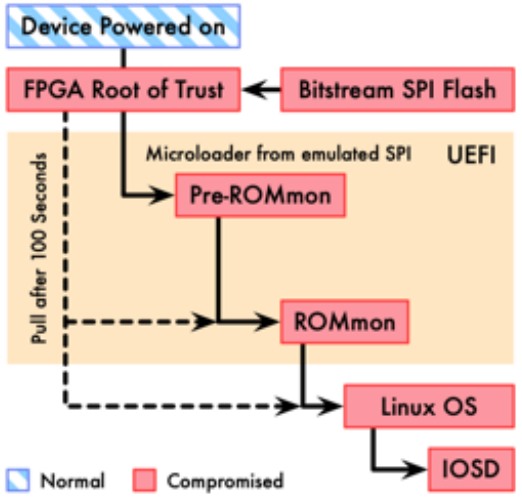
```
loc_AAE:
mov     rax, cs:gRT
lea     rsi, word_3D40
lea     r9, qword_5E00
lea     r8, [rbp+arg_8]
lea     rdx, unk_3940
lea     rcx, aMebiosextensio ; "MeBiosExtensionSetup"
mov     [rbp+arg_8], r14d
mov     cs:qword_5E00, 1Ah
mov     [rsp+68h+var_48], rsi
call    qword ptr [rax+48h] ; gRT->GetVariable()
                        ; EFI_STATUS(EFIAPI * EFI_GET_VARIABLE) (IN CHAR16 *VariableName, IN EFI_GUID *VendorGuid, OUT UINT32 *Attributes, OPTIONAL IN OUT UINTN *DataSize, OUT VOID *Data)
                        ; VariableName   A Null-terminated string that is the name of the vendor's variable.
                        ; VendorGuid     A unique identifier for the vendor.
                        ; Attributes     If not NULL, a pointer to the memory location to return the attributes bitmask for the variable.
                        ; DataSize       On input, the size in bytes of the return Data buffer. On output the size of data returned in Data.
                        ; Data           The buffer to return the contents of the variable.
```

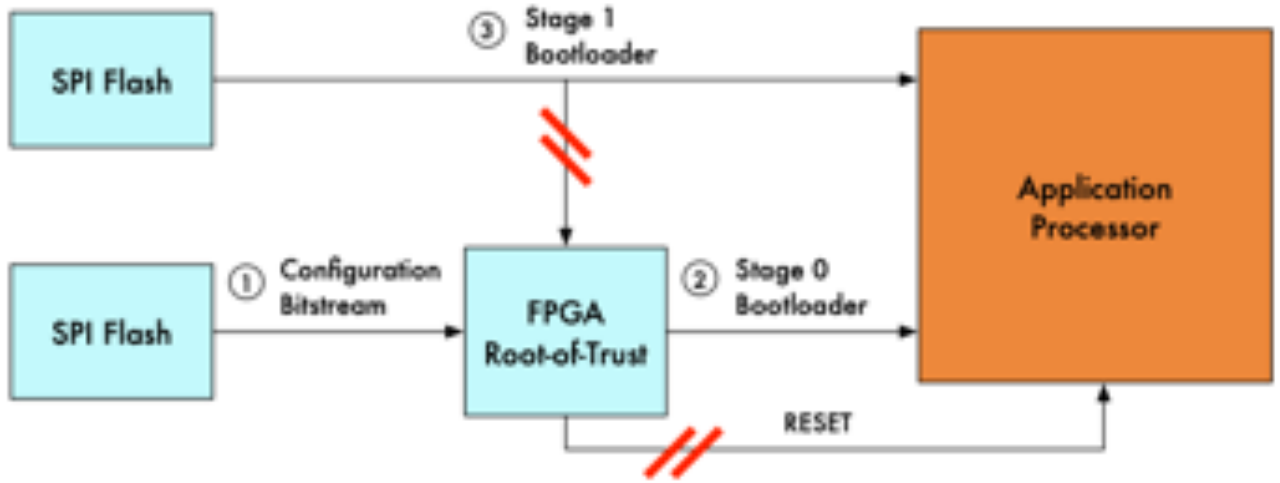
https://github.com/binarly-io/efiXplorer



https://www.platformsecuritysummit.com/2019/speaker/wood/

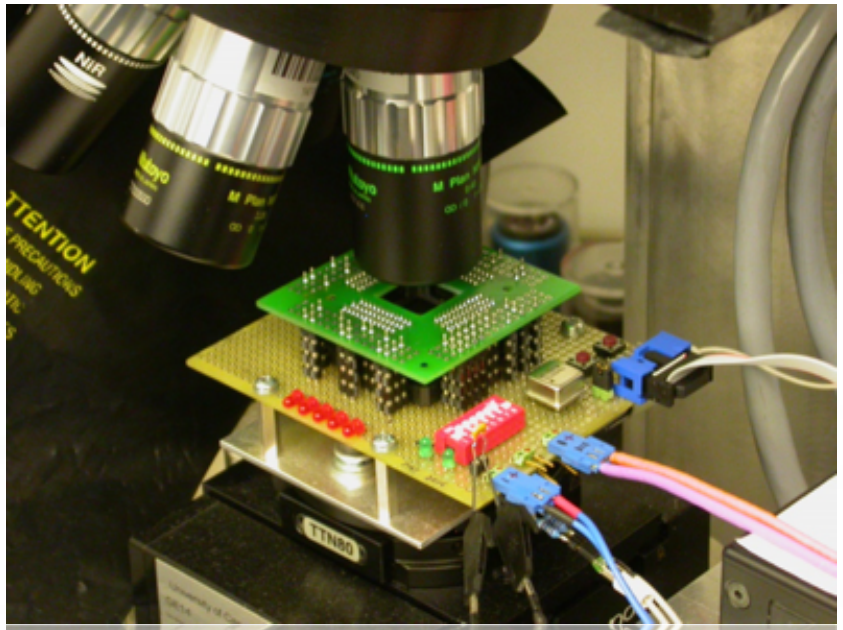
Defeating Cisco Trust Anchor (inside FPGA)





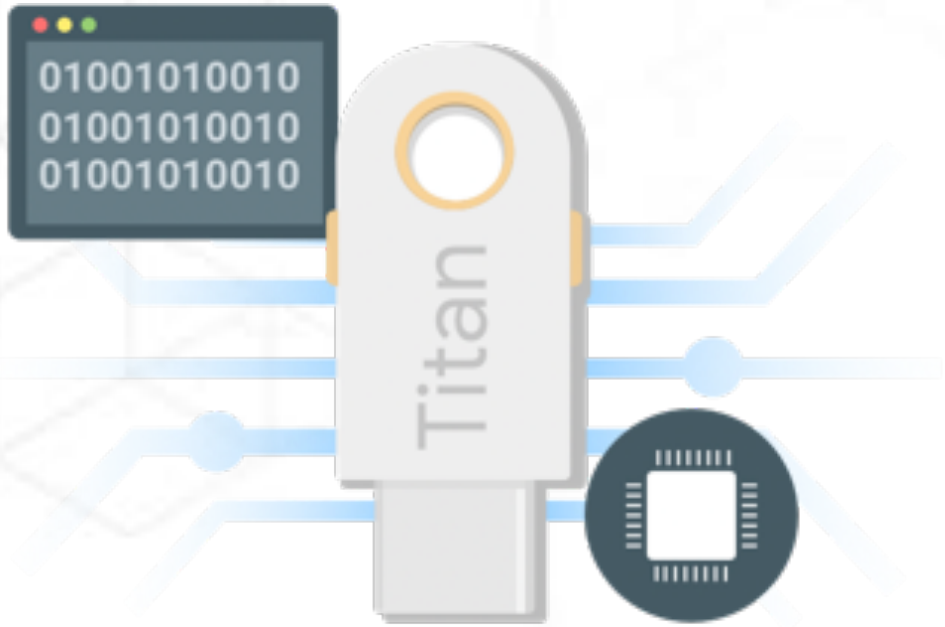
https://www.usenix.org/system/files/woot19-paper_kataria_0.pdf

A little bootloader inside FPGA is immutable

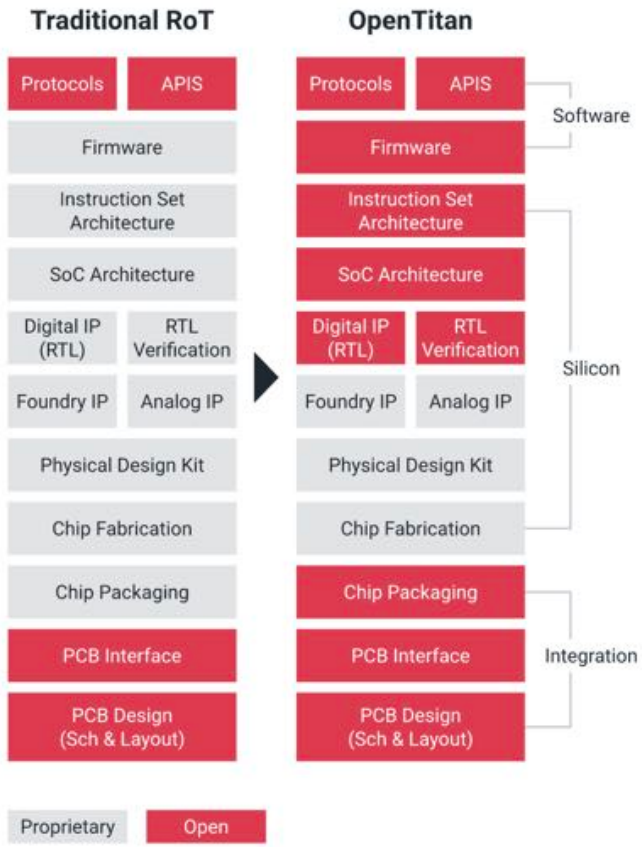


https://arxiv.org/pdf/1910.05086.pdf
https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00349.html
https://media.hardwear.io/hardware-security-evaluation-of-intel-max-10-fpgas/

How open silicon helps to reduce the risk?



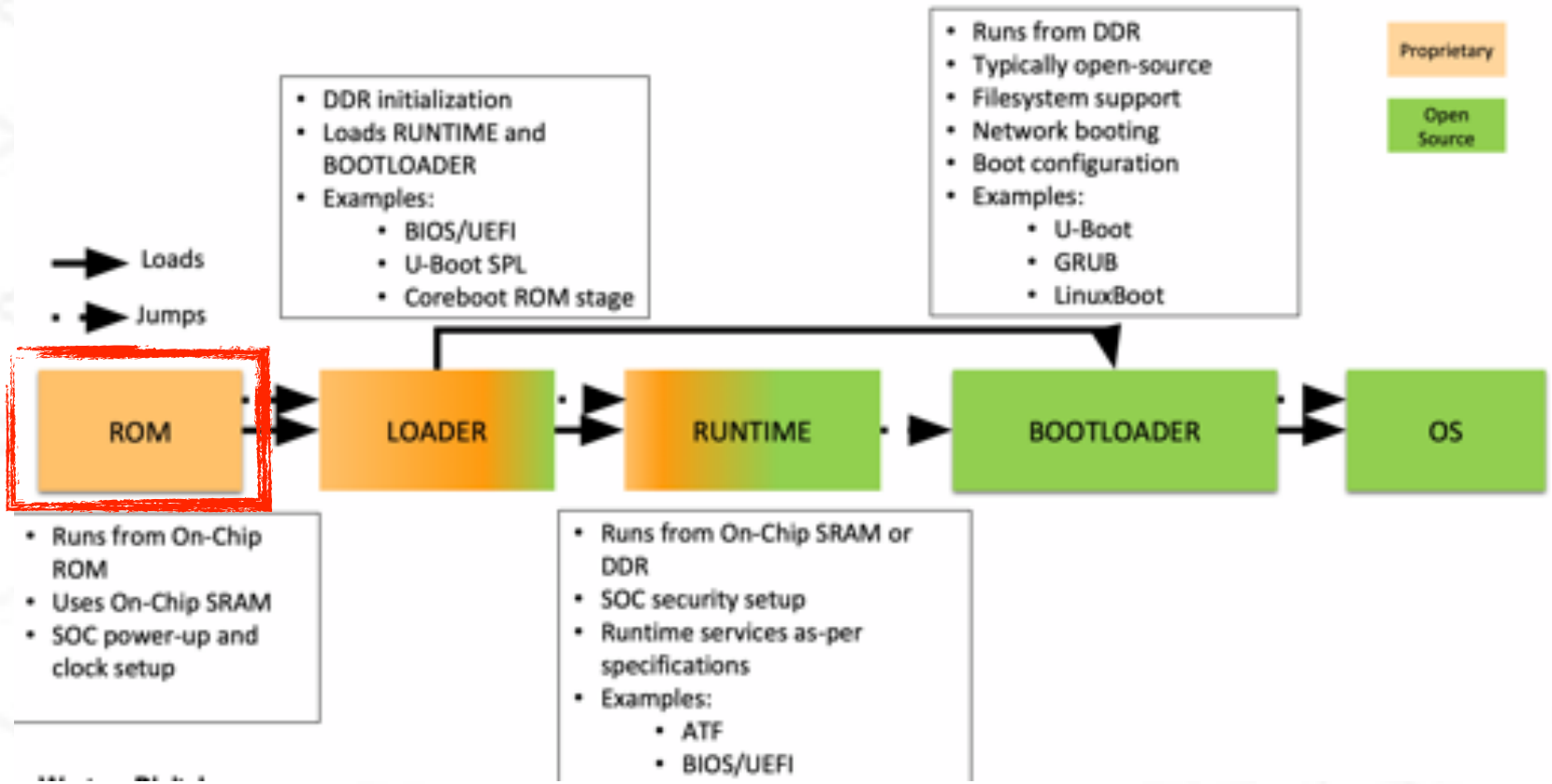
opentitan



https://opentitan.org

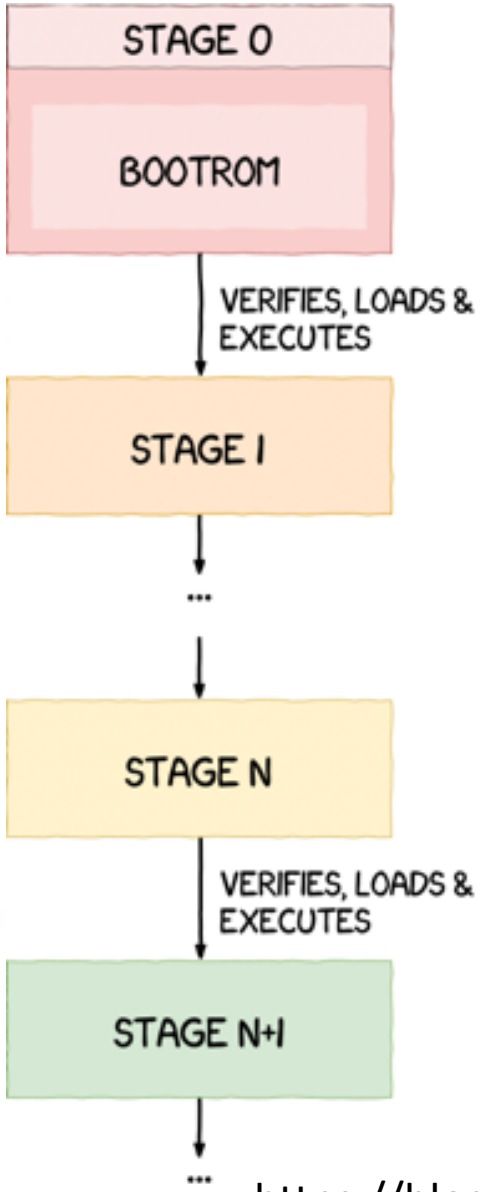
Google recalls some Titan security keys after finding Bluetooth vulnerability

Industry standard boot loaders



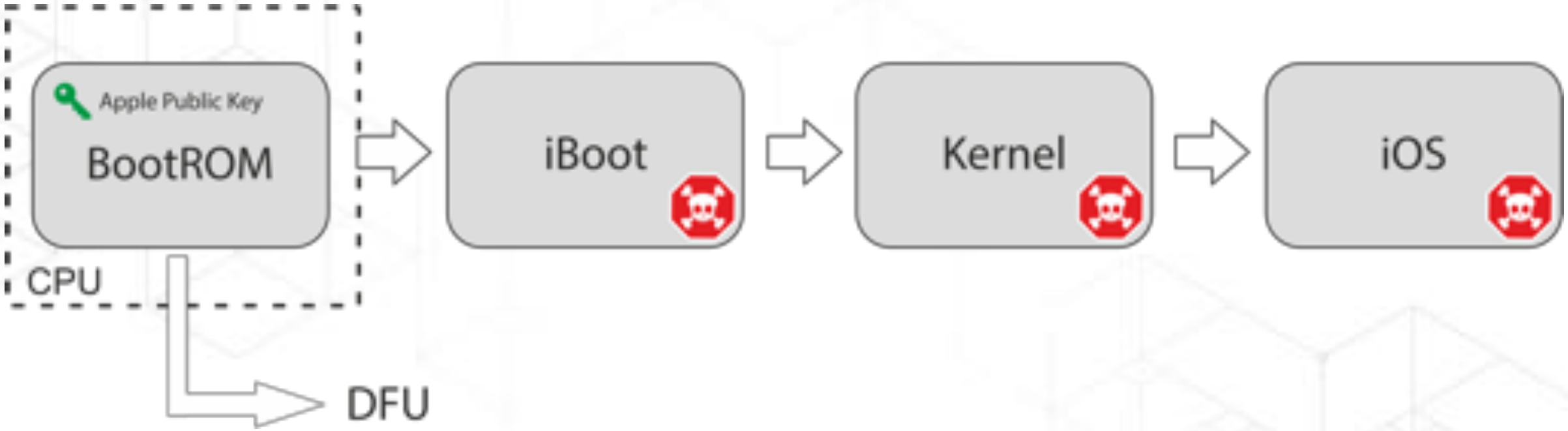
https://osfc.io/uploads/talk/paper/14/The_role_of_open_source_firmware_in_RISC-V.pdf

SECURE BOOT FIX EVERYTHING?



https://blog.quarkslab.com/analysis-of-qualcomm-secure-boot-chains.html

Silicon bugs stay forever with the customers



https://habr.com/en/company/dsec/blog/472762/

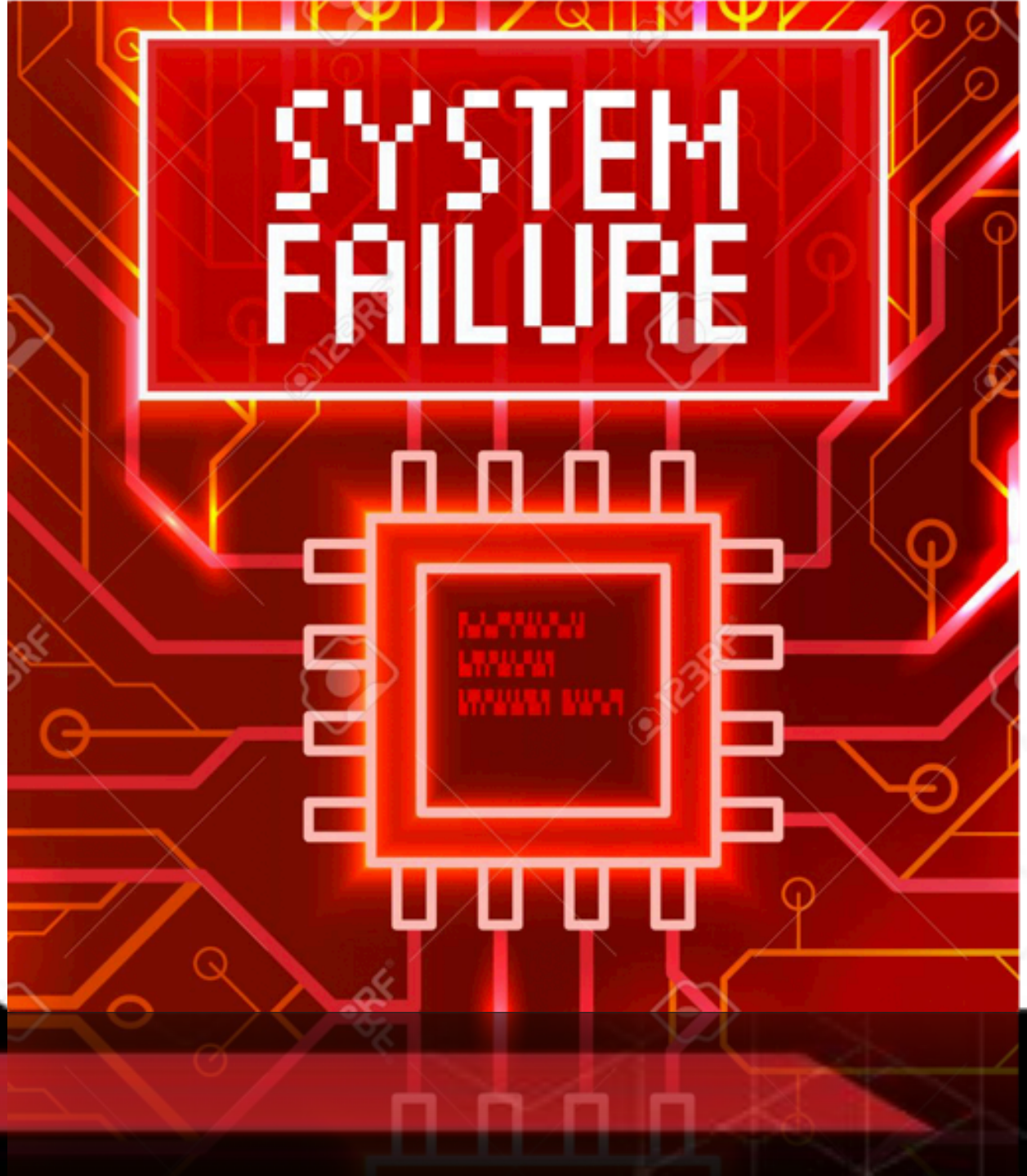
Another good example Intel CSME (CVE-2019-0090)

```
heci_mmio_fwsts0 &= 0xf00ffffu;
heci_mmio_fwsts0 |= 0xf60000u;          // AFTER_SRAM_INIT
if ( _BitScanForward(&first_sram_bank_idx, v0) )
{
  page_dir_phy_addr = first_sram_bank_idx << 0x11;
  *PM_FP(0x10, page_dir_phy_addr + 0x80) = (page_dir_phy_addr + 0x1000) & 0xFFFFF000 | 3;// pde for 0x80000000-0x80400000: phys 0x00001000: Page Table #0
  *PM_FP(0x10, page_dir_phy_addr) = (page_dir_phy_addr + 0x4000) & 0xFFFFF000 | 7;// pde for 0x00000000-0x00400000: phys 0x00004000: Page Table #1
  *PM_FP(0x10, page_dir_phy_addr + 0xFFC) = 0xFFC00005;// pde for 0xffc00000-0xfff00000: phys 0xffc00000
  *PM_FP(0x10, page_dir_phy_addr + 0x1000) = (page_dir_phy_addr + 0x1000) & 0xFFFFF000 | 3;// pte for 0x80000000: phys 0x00001000: Page Table #0
  *PM_FP(0x10, page_dir_phy_addr + 0x1004) = page_dir_phy_addr & 0xFFFFF000 | 3;// pte for 0x80001000: phys 0x00000000: Page Directory
  *PM_FP(0x10, page_dir_phy_addr + 0x1008) = (page_dir_phy_addr + 0x4000) & 0xFFFFF000 | 3;// pte for 0x80002000: phys 0x00004000: Page Table #1
  v12 = (page_dir_phy_addr + 0x2000) & 0xFFFFF000 | 3;
  *PM_FP(0x10, page_dir_phy_addr + 0x1FF8) = v12;// pte for 0x803fe000: phys 0x00002000: stack page 0
  *PM_FP(0x10, page_dir_phy_addr + 0x1FFC) = (v12 + 0x1000) | 3;// pte for 0x803ff000: phys 0x00003000: stack page 1
  __writecr3(page_dir_phy_addr);
  __writecr4(0x10u164);                 // enable large pages(4 Mb) PSE
  cr0 = __readcr0();
  __writecr0((cr0 | 0x80010015) & 0x9FFFFFFF);// enable paging PG and write protect WP
  v14 = 0xFFF00005;                     // map ROM: 0x1000-0x20000
  for ( i = 4; i <= 0xA0; i += 4 )
  {
    *PM_FP(0x10, i + 0x0002000) = v14;
    v14 += 0x1000;
  }
  (0x35510);                            // rom_init_stack_call_main
```

```
1  void __cdecl rom_misa_init_aunit_iommu()
2  {
3    int cur_master; // ebx
4    int i; // eax
5    unsigned int sai_master_idx; // [esp-18h] [ebp-18h]
6    _DWORD ini_masters_sai[8]; // [esp+0h] [ebp-20h]
7
8    cur_master = 0;
9    rom_memcpy(ini_masters_sai, g_rom_initial_dma_masters_sai, 0x18u);
10   dword_F0001160 = 0;
11   dword_F0001164 = 0;
12   dword_F0001168 = 0;
13   dword_F000116C = 0;
14   dword_F0001170 = 0;
15   dword_F0001174 = 0;
16   dword_F0001178 = 0;
17   dword_F000117C = 0;
18   dword_F0001120 = 7;                    // Turn on MSI protection (already on), Transalation and Access Control
19   dword_F0001100 = 0xF0000000;
20   do
21   {
22     sai_master_idx = ini_masters_sai[cur_master++];
23     rom_misa_acc_ctrl_add_dma_buffer(sai_master_idx, 0, 0x40000, 0x7FFBFFFF, 1, 1);
24   }
25   while ( cur_master != 6 );
26   for ( i = 0; i != 0x40; ++i )
27     g_rom_misa_att_used_slots[i] = 0xFF;
28 }
```

CSME ROM initialize page directory, turns-on paging and maps ROM.
IOMMU is turned-on too late an attacker can remap execution pages to arbitrary physical address (SPI 😈)

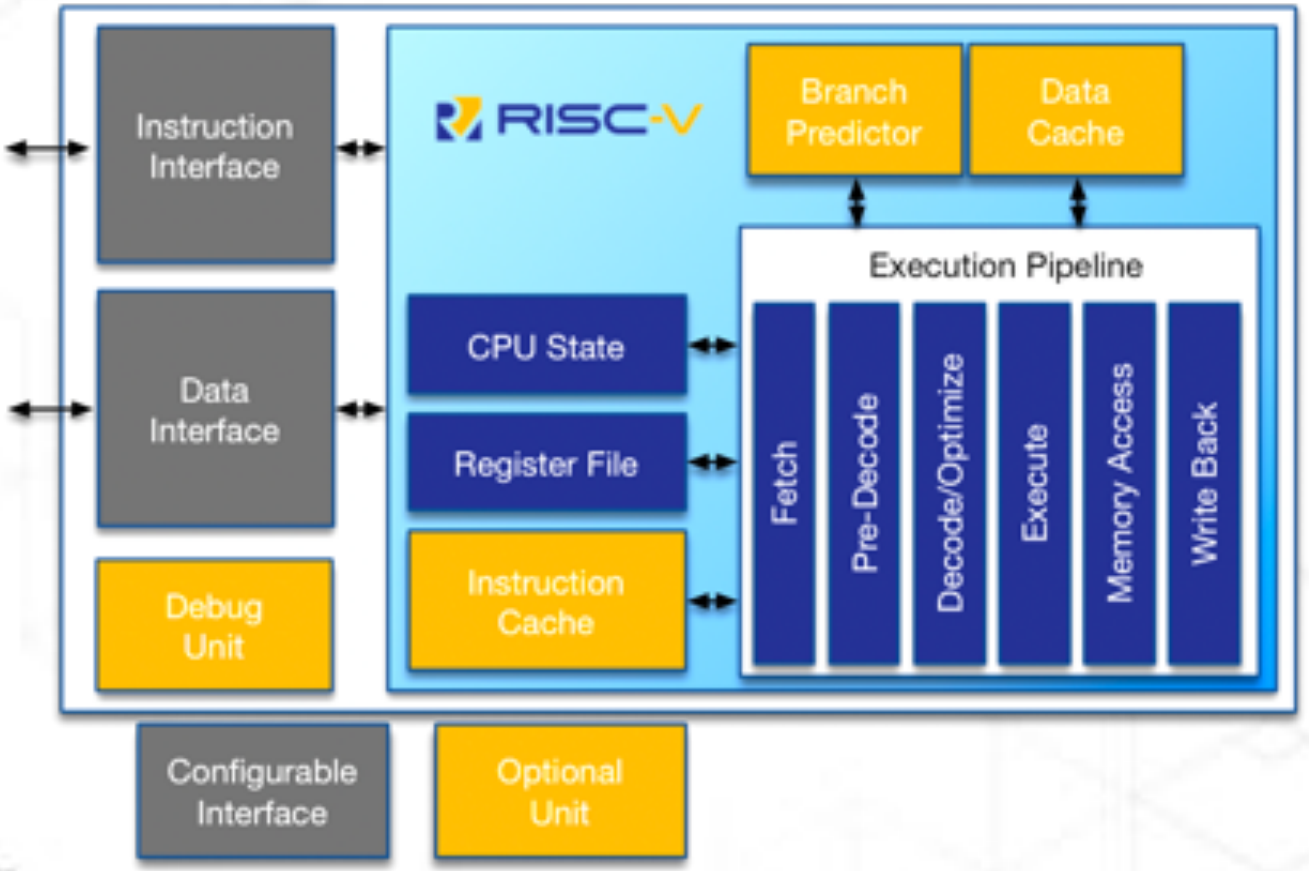
http://blog.ptsecurity.com/2020/03/intelx86-root-of-trust-loss-of-trust.html

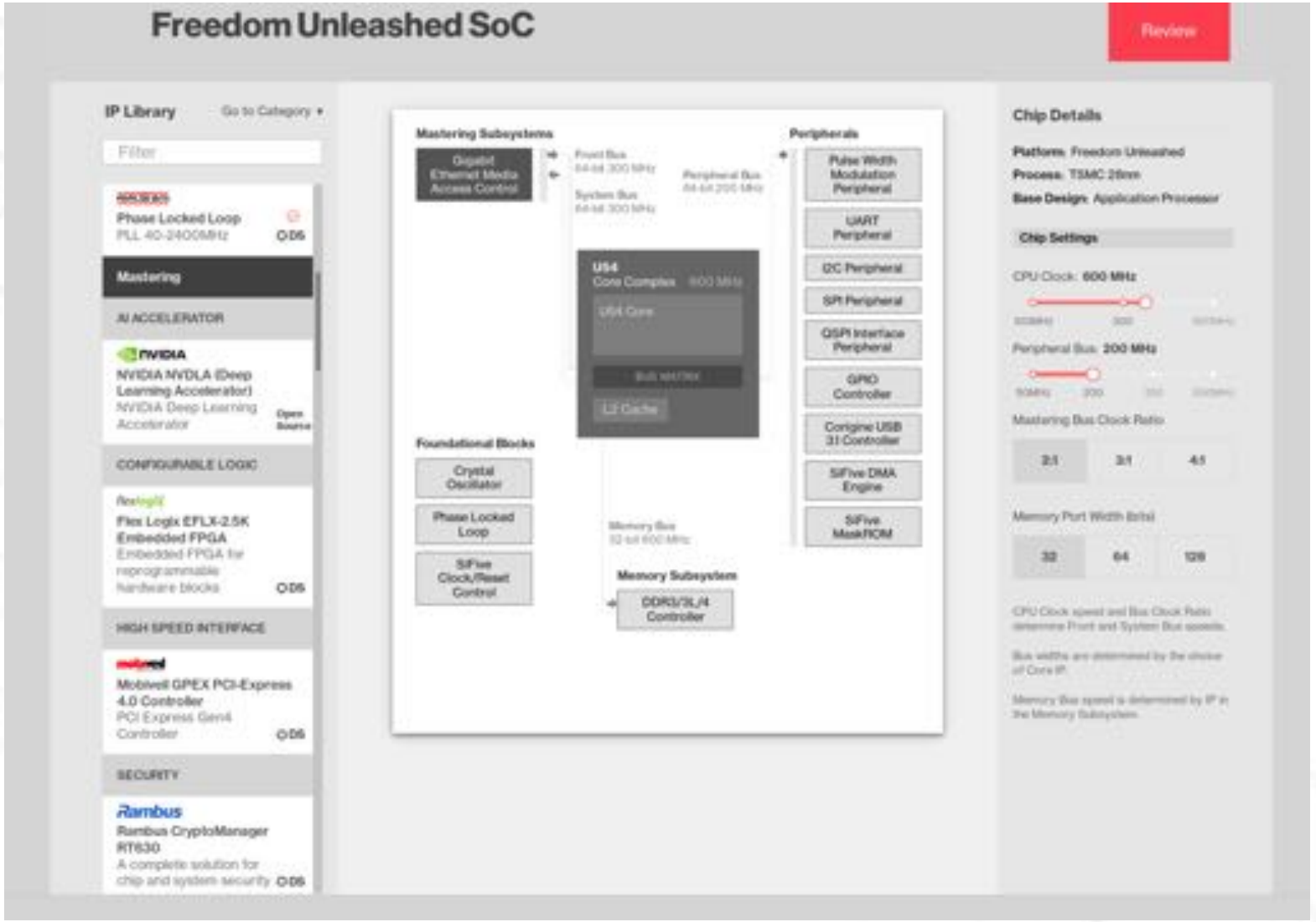

SYSTEM
FAILURE

WE TRUST BLINDLY ANYTHING WHICH IS SIGNED AND COME FROM TRUSTED SOURCE

RISC-V®

RISC-V®

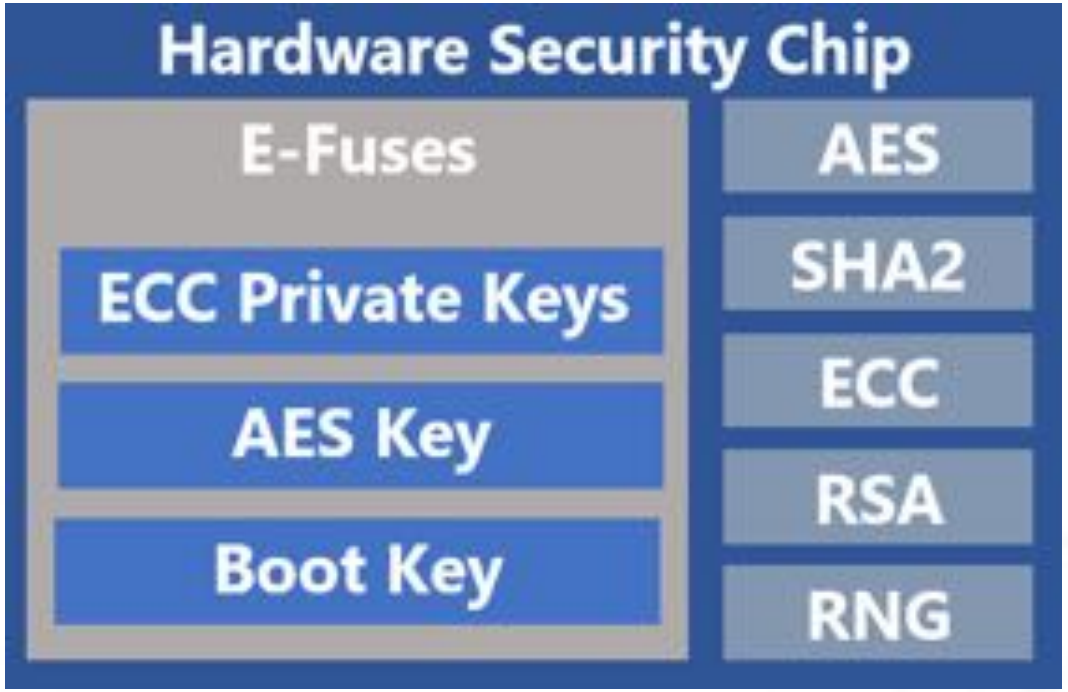

Instruction Interface
Data Interface
Debug Unit
RISC-V
Branch Predictor
Data Cache
Execution Pipeline
CPU State
Register File
Instruction Cache
Fetch
Pre-Decode
Decode/Optimize
Execute
Memory Access
Write Back
Configurable Interface
Optional Unit

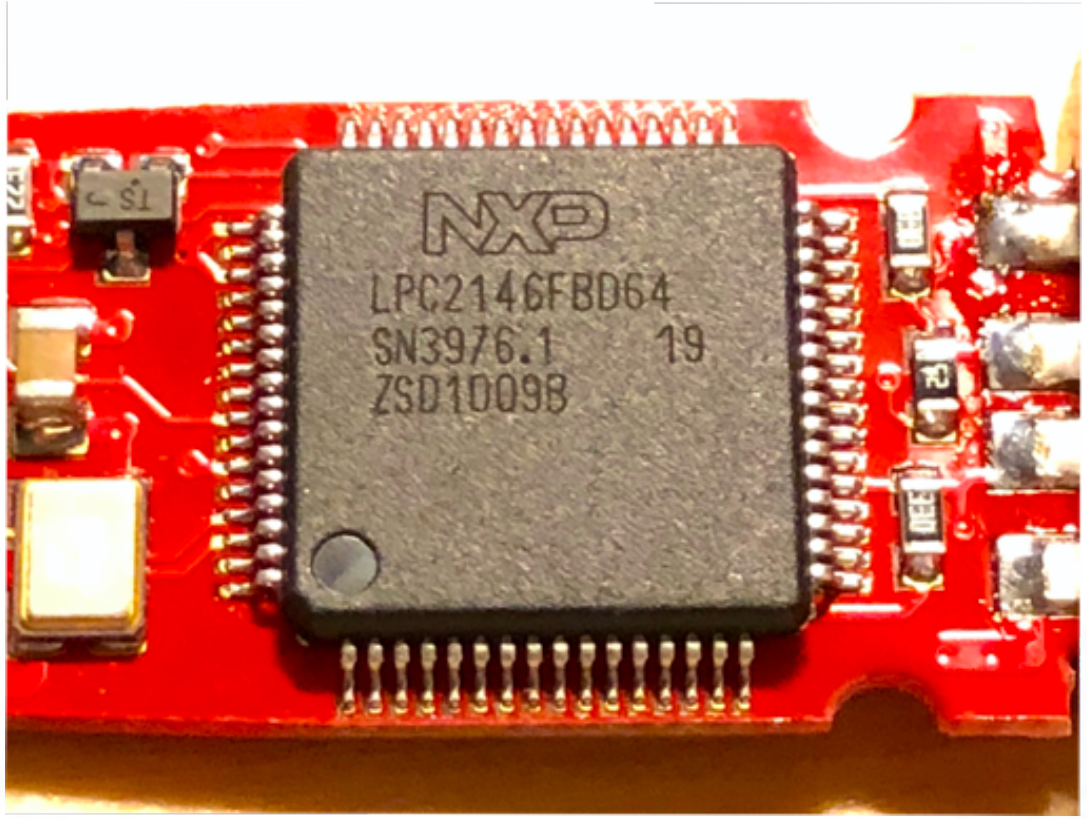


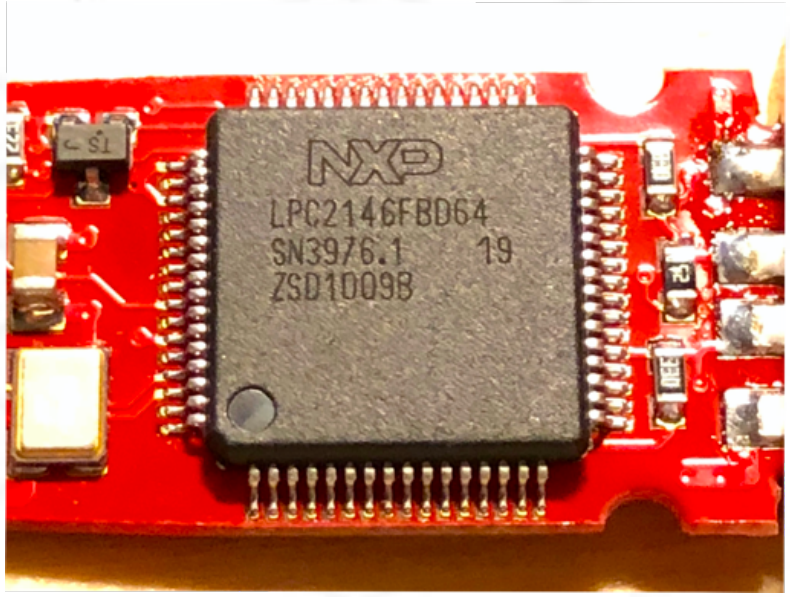
https://www.sifive.com/chip-designer

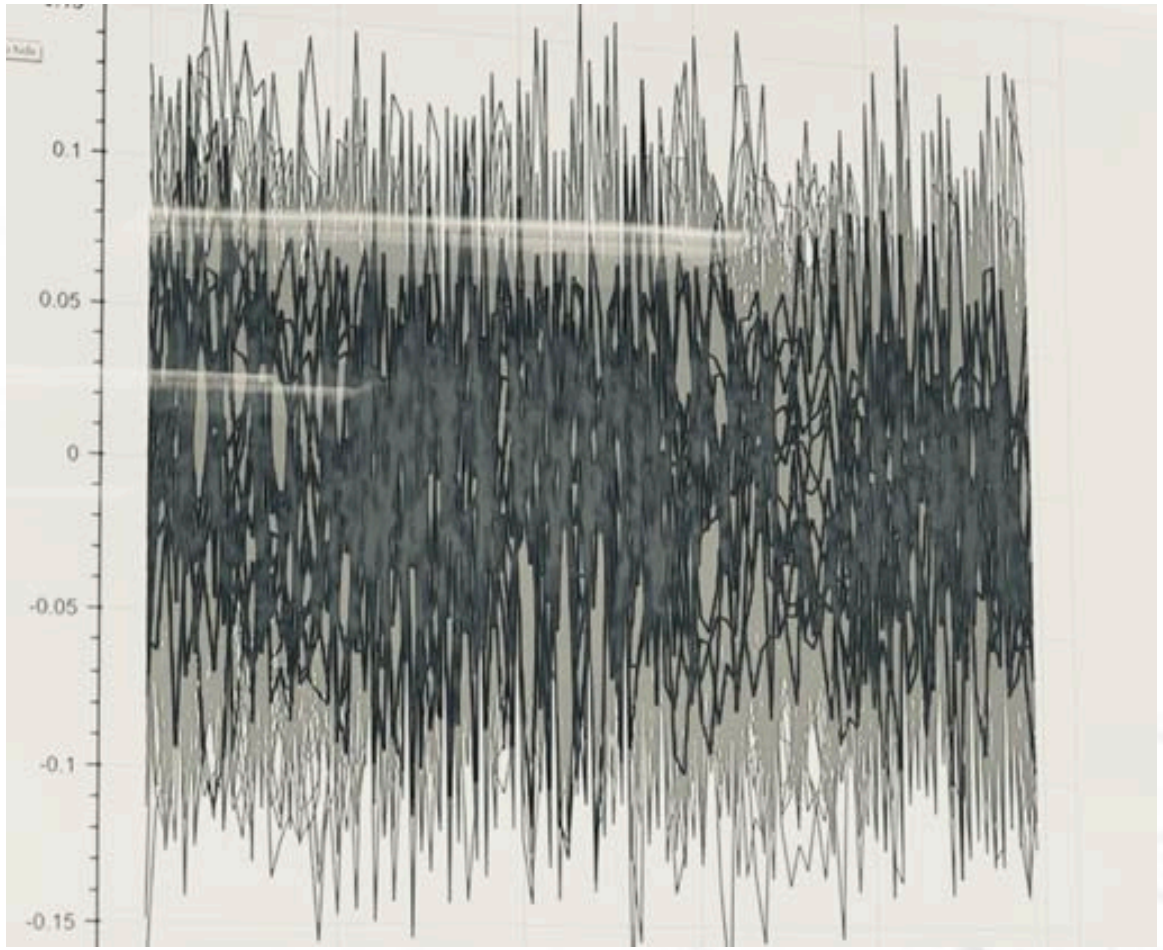
Fault injection attacks do not scale well, but their results do! Whenever something is extracted (E.g secrets, keys, firmware and vulnerabilities identified in the firmware), it will be distributable! **Fault injection attacks are a first step in a complete chain of attacks!**

Niek Timmers (@tieknimmers)


Hardware Security Chip
E-Fuses
ECC Private Keys
AES Key
Boot Key
AES
SHA2
ECC
RSA
RNG


NXP
LPC2146FBD64
SN3976.1 19
ZSD10098


NXP
LPC2146FBD64
SN3976.1 19
ZSD10098


0.1
0.05
0
-0.05
-0.1
-0.15

HARDWARE SECURITY IS HARD!

WE REALLY NEED TO RETHINK MEANING HARDWARE SECURITY IN REALITIES OF MODERD THREAT LANDSCAPE

Thank You!

@matrosov

HITB LOCKDOWN 002 livestream