

Snatching defeat from the  
jaws of victory

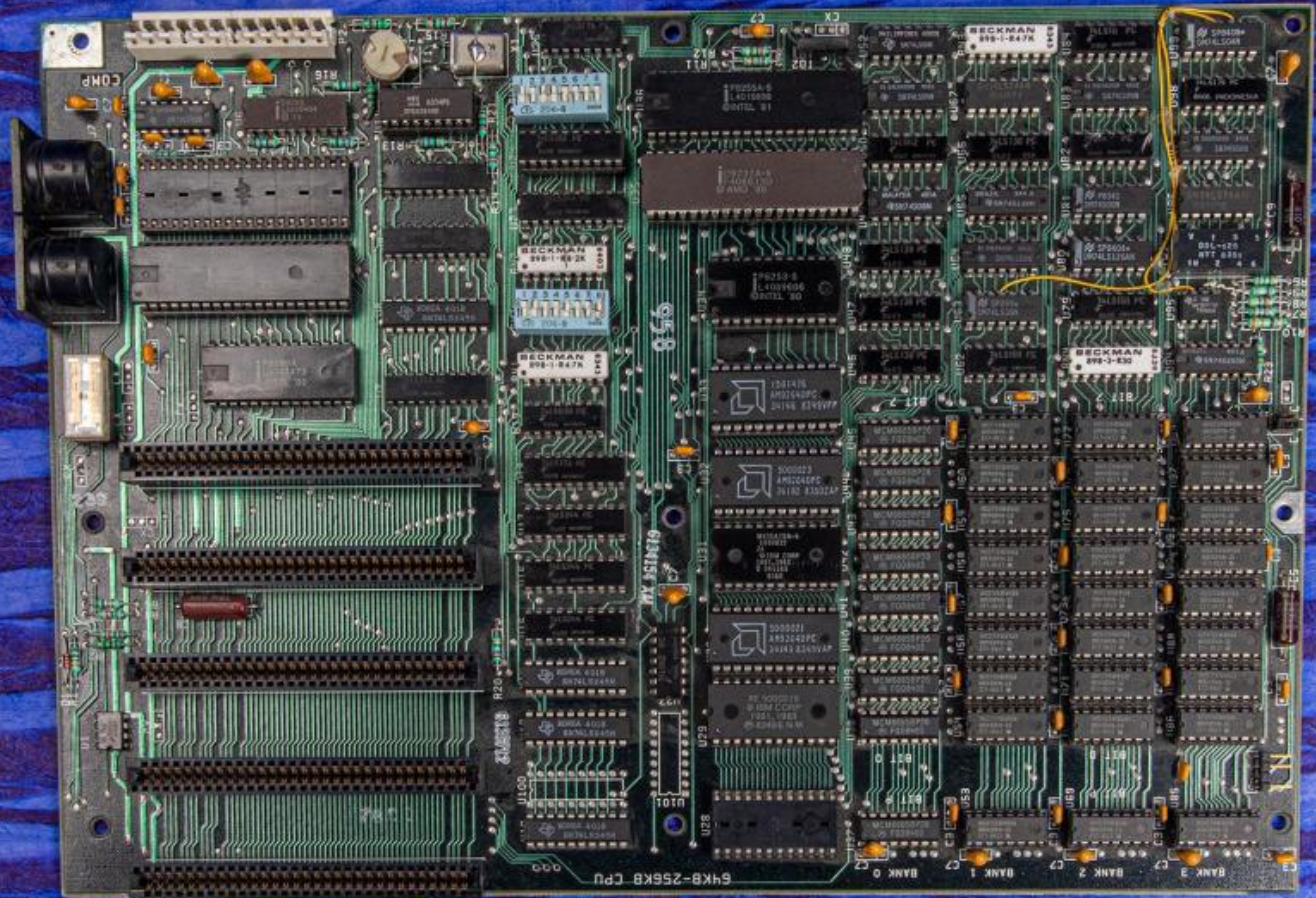


# overview

- Goal: give an overview of how well-intended security ideas go wrong
- With the hope of aiding your work
- Who am I
- Firmware and kernel hacker since 1976
- Invented LinuxBIOS, coreboot, oreboot (firmware in Rust), LinuxBoot, ...
- Spent 8 years building very large HPC systems software at Los Alamos
  - LinuxBIOS
  - Clustermatic
- Spent 2 years building ChromeBook firmware -- including new ARM port
- Led deployment of LinuxBoot to Google data centers at scale
- On the Board of lowrisc.org, as part of the Open Titan effort
- Tilting at windmills for 23 years now
  - Watching all the scary predictions from 1999 slowly come true

# Recent stories to start us off

- AMD CPU can not be moved between mainboards. Why?
- Intel IO chip can not be used on a different mainboard. Why?
- To defeat SPI corruption detection, remove SPI corruption notifier. Why?
- What's wrong? "Zero knowledge" of lower levels :-)






# The problem

- Linux no longer controls the x86 platform
- Between Linux and the hardware are *at least* 2.5 kernels
- They are completely proprietary and (perhaps not surprisingly) exploit-friendly
- And the exploits can *persist*, i.e. be written to FLASH, and you can't fix that
- Note: from 1999-2017 or so, we knew this was possible
- Starting around 2017, it was being discovered in practice
- We are already years behind the “bad guys”, and most vendors are still in denial



# Firmware Repeatable Failures

Vendor	Vulnerabilities	Number of Issues	BINARLY ID	CVE ID	CVSS score
	PEI Memory Corruption (Arbitrary Code Execution)	3	<a href="#">BRLY-2022-027</a> <a href="#">BRLY-2022-009</a> <a href="#">BRLY-2022-014</a>	CVE-2022-28858 CVE-2022-36372 CVE-2022-32579	8.2 High 8.2 High 7.2 High
	 DXE Arbitrary Code Execution	1	<a href="#">BRLY-2022-015</a>	CVE-2022-34345	7.2 High
	SMM Memory Corruption (Arbitrary Code Execution)	2	<a href="#">BRLY-2022-003</a> <a href="#">BRLY-2022-016</a>	CVE-2022-27493 CVE-2022-33209	7.5 High 8.2 High
	SMM Memory Corruption (Arbitrary Code Execution)	6	<a href="#">BRLY-2022-010</a> <a href="#">BRLY-2022-011</a> <a href="#">BRLY-2022-012</a> <a href="#">BRLY-2022-013</a> <a href="#">BRLY-2021-046</a> <a href="#">BRLY-2021-047</a>	CVE-2022-23930 CVE-2022-31644 CVE-2022-31645 CVE-2022-31646 CVE-2022-31640 CVE-2022-31641	8.2 High 7.5 High 8.2 High 8.2 High 7.5 High 7.5 High

# You can't make this up: actual quote

- “... if you look at edk2 specifically, we had a tons of CVEs in ~2015 timeframe, but these days things are slower. “
- That same week, from another engineer:
- “hey neat, the I2C handler will just happily increment an offset every time it receives an UfmWriteFIFO command ordinal, and so it will eventually write attacker-controlled data beyond the end of a fixed-size array”
- [https://github.com/opencomputeproject/Tektagon-OpenEdition/blob/f7350a3a175373532f\[...\]/hyr/ApplicationLayer/tektagon/src/Smbus\\_mailbox/Smbus\\_mailbox.c](https://github.com/opencomputeproject/Tektagon-OpenEdition/blob/f7350a3a175373532f[...]/hyr/ApplicationLayer/tektagon/src/Smbus_mailbox/Smbus_mailbox.c)
-



# What do most modern systems look like?

Application  
Processor (dozens  
of cores), e.g. x86,  
ARM server, etc.

Controlled

Service processors  
Manage power-on/reset

- Control AP power cycle
- Measure boot block
- Provide boot block
- Train/zero memory
- Install exploits
- Exfiltrate data

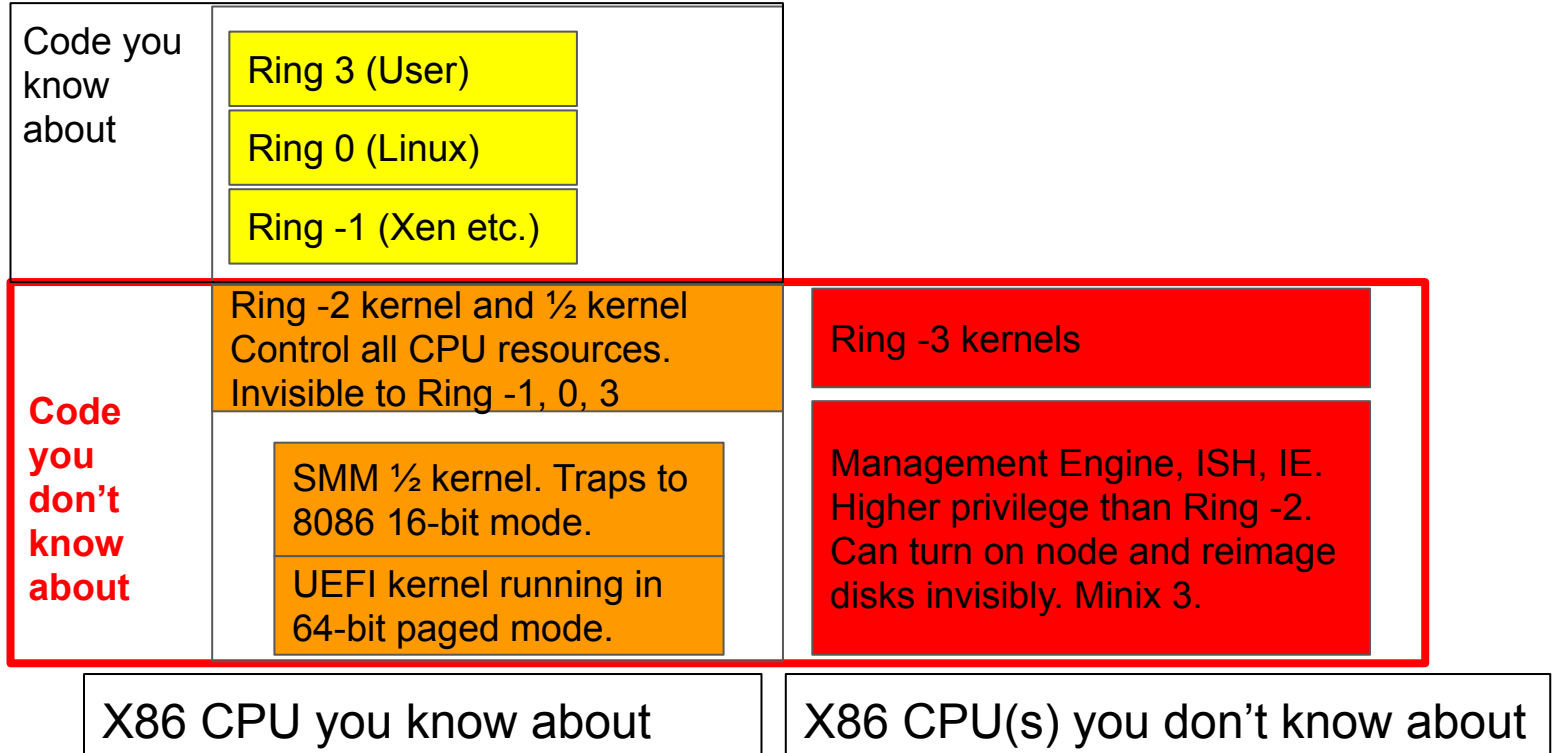
Controller

- The level of sophistication of the average server rivals mainframes
- Lots of great opportunity exploits

# Service processes on various systems

- IBM 370 (1970s): load microcode from floppy disc
- PPC 905 / DEC Alpha: blast bootstrap into L1 cache
- Power 8 servers: two RS6000 workstations
  - With a service processor on each of them
- AMD server since 2018 or so: train memory, load firmware into it
- Intel for around last 10 years: manage clocks, networks, USB, and boot block verification!
- ARM: manage power-on/reset activities
- Controlling the AP provides small comfort
- All the real action is moving to the service processors

# The operating systems



# What's in ring -2 and ring -3?

- IP stacks (4 and 6)
- File systems
- Drivers (disk, net, USB, mouse)
- Web servers
- Passwords (yours)
- Can reimage your workstation even if it's powered off

outcomes



<https://eclipsium.com/2018/08/27/uefi-remote-attacks/>




<https://www.bleepingcomputer.com/news/security/uefi-firmware-vulnerabilities-affect-at-least-25-computer-vendors/> (FEB 2022)

**HP Patches UEFI Vulnerabilities Affecting Over 200 Computers (May 12 2022)**

**CVE-2021-0144**

**Insecure default variable initialization for the Intel BSSA DFT feature may allow a privileged user to potentially enable an escalation of privilege via local access.**

# Firmware Repeatable Failures

Vendor	Vulnerabilities	Number of Issues	BINARLY ID	CVE ID	CVSS score
	PEI Memory Corruption (Arbitrary Code Execution)	3	<a href="#">BRLY-2022-027</a> <a href="#">BRLY-2022-009</a> <a href="#">BRLY-2022-014</a>	CVE-2022-28858 CVE-2022-36372 CVE-2022-32579	8.2 High 8.2 High 7.2 High
	DXE Arbitrary Code Execution	1	<a href="#">BRLY-2022-015</a>	CVE-2022-34345	7.2 High
	SMM Memory Corruption (Arbitrary Code Execution)	2	<a href="#">BRLY-2022-003</a> <a href="#">BRLY-2022-016</a>	CVE-2022-27493 CVE-2022-33209	7.5 High 8.2 High
	SMM Memory Corruption (Arbitrary Code Execution)	6	<a href="#">BRLY-2022-010</a> <a href="#">BRLY-2022-011</a> <a href="#">BRLY-2022-012</a> <a href="#">BRLY-2022-013</a> <a href="#">BRLY-2021-046</a> <a href="#">BRLY-2021-047</a>	CVE-2022-23930 CVE-2022-31644 CVE-2022-31645 CVE-2022-31646 CVE-2022-31640 CVE-2022-31641	8.2 High 7.5 High 8.2 High 8.2 High 7.5 High 7.5 High

# CosmicStrand: the discovery of a sophisticated UEFI firmware rootkit

<https://securelist.com/cosmicstrand-uefi-firmware-rootkit/106973/>

<https://arstechnica.com/information-technology/2022/07/researchers-unpack-unkillable-uefi-rootkit-that-survives-os-reinstalls/>

“Discovery of new UEFI rootkit exposes an ugly truth: The attacks are invisible to us

Turns out they're not all that rare. We just don't know how to find them.”

Why didn't anyone tell us? Oh wait. We did. Starting in ...  
1999



# Need open *development*, *not just open source*

CVE-2021-0144

Insecure default variable initialization for the Intel BSSA DFT feature may allow a privileged user to potentially enable an escalation of privilege via local access.

How is it possible for one bug to be around so long?

Closed development

2nd Generation Intel® Xeon® Scalable Processor Family

10th Generation Intel® Core™ Processor Family  
Intel® Xeon® Scalable Processor Family

9th Generation Intel® Core™ Processor Family  
Intel® Xeon® Processor W Family

8th Generation Intel® Core™ Processor Family  
Intel® Xeon® Processor E v3 and v5 Family

7th Generation Intel® Core™ Processor Family  
Intel® Xeon® Processor D Family

6th Generation Intel® Core™ processor Family  
11th Generation Intel® Core™ Processor Family

Intel® Core™ X-series Processor Family

Intel® Atom® Processor C3XXX Family.

# System management mode

- Introduced with 486 to support power management
  - Close lid, what happens? DOS is not going to handle sleep!
  - Backward compatibility has to work
- Hence, need:
  - Higher priv level than Ring 0 (i.e. beyond DOS)
  - Operations, e.g. sleep, that run without Ring 0 knowing it
  - Additional interrupts vectored to these operations
  - Operations must be deeply hidden so Ring 0 does not break them
    - I.e. in a memory space Ring 0 can never see
    - Protected by one-way-locking registers
  - No state leakage across the boundary, esp. to Ring 0
- Design based on these requirements ends up at SMM
- But note: in the beginning, it's all about DOS

# Evil-ution

- If there is a place to put secret code that can never be seen and is highest privilege, will vendors use it?
- Well, duh ....
- If secret code is written to lowest-common-denominator standards, and handed to random vendors who put in random features designed for customer lock-in, will it be full of 0-days and nasty bugs?

# Eliminating SMM: you can not

- No feature ever leaves the x86
  - See: DAA, the unused opcode that eats 1/256 of the space
- In some ways, it's easy: don't enable it
- In other ways, it's hard: there might be something about your hardware
- As soon as we remove it one place, x86 vendors find another use for it
  - Which is why the "SMM segment" grew from 64KiB to 8MiB
- Prediction: it will never go away on x86
- And the impact will always be "bad"
- Because of "all fall down" model -- if one core enters SMM, all cores do

# SMM: ever more uses, every more growth

## *Executive Summary*

In the current UEFI PI infrastructure, SMM drivers are loaded in the PI DXE phase. Usages such as the Intel® Firmware Support Package (FSP) may include requirements that the SMM initialization be done in the early PI PEI phase, namely since current FSP environments are in PEI. Intel FSP is a binary to encapsulate Intel silicon module initialization. In addition, server Reliability, Availability, Serviceability (RAS) features may also require some RAS SMM modules to be launched in PEI because portions of RAS are part of the silicon module set. This paper presents how to support launching silicon specific SMM drivers in the PI PEI phase, while at the same time maintaining compatibility to launch existing SMM drivers in the PI DXE phase.

# What about ARM?

- ARM has proven to be very supportive of open source firmware
  - For example, they were very quick to support the LinuxBoot (LBBS) firmware effort
  - For server, client, and IoT, companies have also found ARM supporting their efforts
- From there, it depends on the SoC vendor
- <https://github.com/AmpereComputing/edk2-platforms>
  - In that repo you will find chipset enabling code that is NDA in the x86 world
  - It's not all there, but it's far more open
  - This is not the common case on ARM server!
- For client in general ARM is open (consider ChromeBooks)
  - In coreboot, external entities are code, not blobs as in x86
- With ARM, one must consider both ARM *and* the SoC vendor
- Remember the rule: Repos talk, \*\* walks

But RISC-V will fix all this, right?

# RISC-V is following the same path

- SBI (BIOS) spec is 64K and growing
- Almost 100 different BIOS calls
- Some take virtual address (no virtual addressing in RISC-V BIOS mode)
- *Some can suspend*
- Eight different implementors
- Two different languages
- No apparent plan for what HPC people call Verification and Validation
- Especially performance verification
- So we're building an FPGA RISC-V that turns it off :-)



# Ground rules for future hardware

- No fuses -- and “not programming the fuse” doesn’t count
  - Unprogrammed fuses in mainboards enabled another exploit
  - Bad guys installed firmware, locked fuses, game over, you must shred the mainboard
- No blobs that require secret keys
  - Intel Alder Lake BIOS code leak -- contains signing keys
  - Microcode is signed on x86 (but not IBM Power!)
- *All* code for *all* cpus must be available, buildable, installable
  - AP code is not much use without service processor code
- Open development, not open source
- Which leads to a controversial point ...

# Controversial point

- Code at this level should be GPL
  - So you can audit
  - So you can repair and provide repairs to others
  - So you can share bug findings and fixes
  - So the system can outlive the manufacturer
- Why aren't MIT/BSD/MPL/whatever good enough?
- Because vendors can (and have)
  - Taken non-GPL licensed code
  - Added bugs ("features")
  - Released only binary form (occasionally non-buildable source under NDA)
  - Made any kind of third-party review impossible
  - Made bug reporting and fixing impossible
- non-GPL for firmware is a very bad model for security