Replacing x86 firmware with Linux and Go

[LWN subscriber-only content]

Welcome to LWN.net

The following subscription-only content has been made available to you by an LWN subscriber. Thousands of subscribers depend on LWN for the best news from the Linux and free software communities. If you enjoy this article, please consider accepting the trial offer on the right. Thank you for visiting LWN.net!

Free trial subscription

Try LWN for free for 1 month: no payment or credit card required. Activate your trial subscription now and see why thousands of readers subscribe to LWN.net.

By **Jake Edge**
November 20, 2017
ELC Europe

The Intel Management Engine (ME), which is a separate processor and operating system running outside of user control on most x86 systems, has long been of concern to users who are security and privacy conscious. Google and others have been working on ways to eliminate as much of that functionality as possible (while still being able to boot and run the system). Ronald Minnich from Google came to Prague to talk about those efforts at the 2017 Embedded Linux Conference Europe.

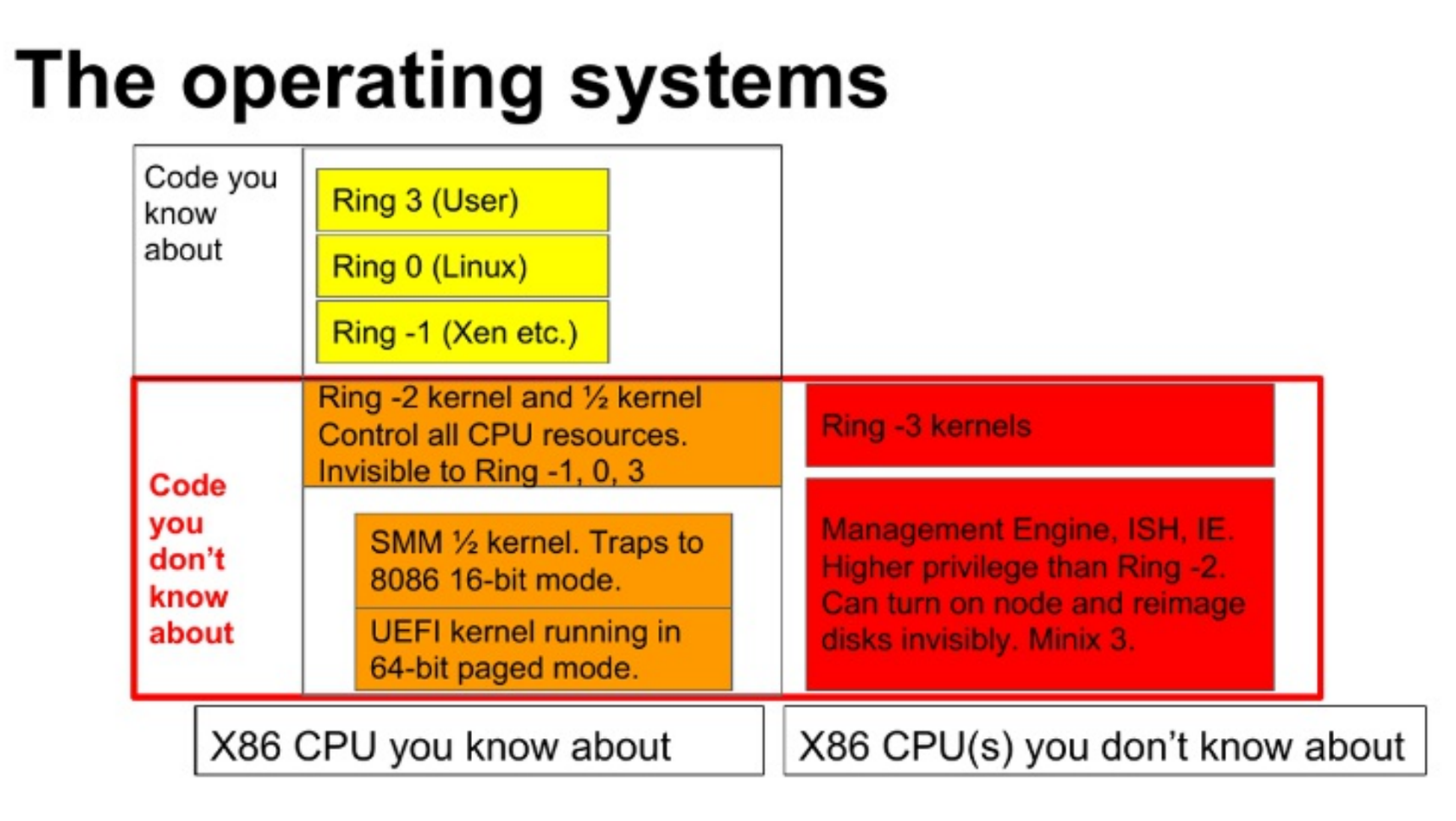
He began by noting that most times he is talking about firmware, it is with his coreboot hat on. But he removed said "very nice hat", since his talk was "not a coreboot talk". He listed a number of people who had worked on the project to "replace your exploit-ridden firmware with a Linux kernel", including several from partner companies (Two Sigma, Cisco, and Horizon Computing) as well as several other Google employees.

The results they achieved were to drop the boot time on an Open Compute Project (OCP) node from eight minutes to 20 seconds. To his way of thinking, that is "maybe the single least important part" of this work, he said. All of the user-space parts of the boot process are written in Go; that includes everything in initramfs, including `init`. This brings Linux performance, reliability, and security to the boot process and they were able to eliminate all of the ME and UEFI post-boot activity from the boot process.

Describing the mess

The problem, Minnich said, is that Linux has lost its control of the hardware. Back in the 1990s, when many of us started working with Linux, it controlled everything in the x86 platform. But today there are at least two and a half kernels between Linux and the hardware. Those kernels are proprietary and, not surprisingly, exploit friendly. They run at a higher privilege level than Linux and can manipulate both the hardware and the operating system in various ways. Worse yet, exploits can be written into the flash of the system so that they persist and are difficult or impossible to remove—shredding the motherboard is likely the only way out.

He used to give a talk with the title: "If you trust your computer, you're crazy", due to all of that proprietary code running on our systems. He hopes that this talk will give folks ways to deal with some of those problems, "so we can stop being crazy and maybe get a little sane".



He showed one of his slides [PDF] (above) that described the seen and unseen operating systems running on an x86 system. Ring 0 is Linux and, because "we ran out of ring numbers", hypervisors like Xen are ring -1, but below that are rings that are running code that you don't have access to, sometimes on processors you don't even know are part of the system. Ring -2 has a kernel and a half kernel; it consists of UEFI, which is the full kernel, and system management mode (SMM), which traps to 8086 16-bit mode, thus the "half" designation. Those control everything about the CPU and are invisible to the rings above. Every time you close the lid of your laptop, or do certain other things, SMM traps to classic 8086 mode; "that should make you happy", he said sarcastically.

Ring -3 is "the one that has people really worried". It runs MINIX 3 and is where the ME runs. It is the cause of the "year of MINIX 3 on the desktop", he joked, since there are more systems with the ME than any of Linux, macOS, or Windows.

There is no common code between the systems running in ring -2 and ring -3 as far as he knows, but they both have a wide range of capabilities. Both have IPv4 and IPv6 networking stacks, filesystems, drivers for various devices (disk, network, USB, ...), and web servers. The ME needs filesystems because it can be used to reimage the system; in fact, Minnich said, it can reimage the system even if the power is turned off as long as it is plugged into the wall and the network.

There is a whole raft of components that make up the ring -3 ME, many of which he does not understand. For example there are components named "full network manageability", "regular network manageability", and "manageability", as well as the "outbreak containment heuristic". He pointed to a Master's thesis [large PDF] from Vassilios Ververis about ten years ago that looked at many different flaws in the ME. It is rather depressing, Minnich said, since it showed that almost every part of the ME could be attacked; some of those bugs still have not been fixed.

He referenced the headline of a *Wired* article about an ME exploit ("Intel Fixes a Critical Bug That Lingered for 7 Dang Years") that he thought was funny. Less funny was the bug itself that allowed a zero-length password to be sent to the web server to give administrator access to systems with the ME. Since the bug was present for seven years, that adds up to around a billion systems, he said, and he strongly doubts that all of those have been patched with a firmware update.

He moved on to the half OS in ring -2. SMM was originally meant to handle power management on DOS systems; it can take over the system out from under ring 0 when certain events (system management interrupts or SMIs) occur. There are a lot of SMI exploits and, once SMM is enabled, it cannot be turned off. It takes 8MB of memory away from the rest of the system for its purposes. SMM is "a good way to maintain vendor control over you", he said.

The other thing running in ring -2 is UEFI; both it and SMM run on the main CPU. UEFI is "an extremely complex kernel"; vendors are writing code for the kernel, but they don't understand all of the rules, so they make mistakes. The result is that "there are big, giant holes that people can drive exploits through". The UEFI security model, as far as he can see, is obscurity.

There are tons of exploits for UEFI, he said. Because UEFI is updated by handing off bits of UEFI code to the UEFI kernel, he is worried that exploits will persistently infect that process, such that it will claim to update itself, but not do so. That only leaves the shredder.

He summarized by reiterating what he had just described: 2.5 hidden OSes with network stacks, web servers, and other capabilities. These OSes have bugs that can persist across power cycles and reinstalls and those bugs have been exploited in the past. His old talk used to end here with a question: "Are you scared yet?"

Fixing the mess

"So how do we fix this mess?", he asked. Some people say to switch to AMD processors, but that is not really a solution now. Ryzen is touted to be open, but that is not truly the case, there are still closed parts. So the project is focusing on Intel x86 processors and has the goal of reducing the scope of the 2.5 OSes. The project is called "non-extensible reduced firmware" (NERF), partly because the team believes the "extensible" in UEFI is harmful. ~~Apparently, there is no overall web page for NERF itself, though some of the components Minnich talks about do have web pages.~~ [Update: As noted in the comments, there is a NERF web page.]

The idea behind NERF is to reduce the harm that the firmware is capable of. In addition, there is an effort to make what the firmware is doing more visible. It does this by removing almost all of the runtime components from the firmware; the "almost" refers to the ME, which is hard to kill completely, he said. If you completely remove the ME, your node probably will not boot, but NERF has taken away the ME's web server and IP stacks. The UEFI IP stack and other drivers have also been removed. Beyond that, the self-reflash capability for ME and UEFI has been removed, so Linux manages all flash updates.

The NERF components are a de-blobbed ME ROM and a UEFI ROM that has been reduced to its most basic parts; in addition, SMM has been disabled or vectored to Linux where that is needed. On top of that runs a Linux kernel with a Go-based user space (u-root). He noted that the project is particularly interested in any Go programmers who want to contribute to just that piece.

They would prefer to remove the ME entirely, but that simply is not an option. If you remove it, the system may not boot, power on, or, if it does power on, it may shut down again in 30 minutes. But there is some good news: the ME has multiple components and most of them *can* be removed.

He pointed to the me_cleaner project that will process an ME ROM to remove most of it. For example, on the MinnowMax, 5MB of the 8MB flash was used for the ME, but that was reduced to 300KB by me_cleaner. So you only need 300KB of the ME to boot Linux and that gets rid of all of the stuff you really don't want the ME to be doing anyway. The ME reduction is working for MinnowMax and a number of other boards, he said.

If you "get into the game early enough", and they believe their Linux kernel does, SMM can be completely disabled. As far as they can tell, there is no requirement to run SMM; it is mostly there for "value add" by the vendors, which is just a way to try to lock people into their platform. If it ever becomes an issue for some hardware, though, there are ways to vector the SMIs to the kernel. The theme is to keep Linux in control, he said.

UEFI is "huge and extremely complex", but there are a lot of mistakes made in the implementation of it. Some interrupts, including memory-error-detection interrupts, still need to be routed to UEFI, though. They want to remove the opportunities for UEFI drivers to put in exploits by making it non-extensible. He showed "an eye chart" of all of the different services that UEFI provides; he noted that it looks like a kernel, because it is, and said that "it is a sizable fraction of the size of Linux".

Next up, he showed the standard UEFI boot process; it starts with two phases (security or SEC and pre-EFI initialization or PEI) that are completely proprietary and will never be released by the vendors, he said. Beyond that, though, the next phase, which is called the driver execution environment (DXE), has a well-defined interface that multiple components (DXE core, drivers, boot manager, ...) conform to.

The boot manager is responsible for starting up the operating system. When you see the screen that allows choosing what to boot on a UEFI system, that is the boot manager. What they have done is to replace the boot manager with a Linux kernel that conforms to the DXE interface. On the OCP node system that was being demonstrated elsewhere at the conference, booting the Linux kernel took 20 seconds from power-on; the Go-based user space does a DHCP query, a `wget` for the server kernel, and then a kexec into the new kernel, which takes an additional three seconds. There are plans to replace the DXE core component with something that is open source and knows more about how to boot Linux; that should reduce the boot time even further, Minnich said.

He does not believe that we will ever get access to that early boot code (SEC and PEI) for UEFI. Even for Chromebooks running coreboot, that piece is a binary blob. The best we can do, he said, is to replace the pieces at that well-defined interface, which is what has been done. In addition, the goal was to get rid of all the UEFI runtime services, which has been accomplished.

As part of his Heads project, Trammel Hudson has put together some Makefiles and the like to create a NERF image. That can be used with a custom kernel and initramfs to replace as much of UEFI as possible. They have had good results on a Dell server, the MinowMax, and the OCP nodes.

Using Linux makes the firmware easier to work with, Minnich said. Normally, there are lots of fiddly, hardware-specific pieces that need to be changed in the firmware, but using the DXE interface makes a lot of those problems go away. He expected that different kernels would be required for the different systems, but he has been using the same kernel on the MinnowMax, which is a small system, and the OCP node, which is a rather large system.

The user-space piece is all written in Go, which is generally more trusted than C within Google, he said. The 5.9MB initramfs contains all of the source code for the user space, all of the Go compiler and package sources, and a Go toolchain. The commands are built on the fly, as they are needed, which usually takes around 200ms per command; once they are built, it is "nearly instantaneous" (1ms) for them to run. From a security angle, that's good because all of the source is available to be examined.

For cases where there is not sufficient space for an initramfs of that size or enough CPU power to do even a fast compile step on the way to booting, there is another mode for the u-root Go commands. It is like BusyBox, in that there is one binary that is linked to a bunch of different command names; this mode uses the Go abstract syntax tree package to rewrite the commands as packages. That reduces the footprint to 2MB, which is useful on systems with less flash space.

There are some implications of the u-root work that has Minnich thinking about booting for desktop systems. With u-root, there are no scripts or unit files to deal with, there is simply a single program that boots the system, which leads to "things coming up really fast". It is more understandable for him and makes the boot process faster. There is a project at Google, called NiChrome, that can bring up a Chromebook all the way from power-up to X11 and a browser in five seconds.

Go is a compiled language, but it is often used for scripting. Minnich uses it that way "all the time"; he stopped writing Bash scripts years ago in favor of Go. It is "easier and more reliable" to write scripts in Go.

He concluded by saying that he is hoping to see companies ship hardware with NERF and u-root in 2018. Companies want to have firmware that they understand, he said; they also want it to boot quickly and be secure. In the Q&A, Minnich was asked about secure boot and TPMs. Neither is supported currently, though there is a non-working verified boot program in u-root at this point. For TPM support, he thinks the project will follow what Chrome OS has done, rather than take the secure boot path.

He was also asked about the relationship of this work to coreboot. Minnich said that coreboot should always be preferred, but it has not been available for server platforms for 12 years. So he would suggest that developers "always use coreboot if you can", but if not, look at NERF.

Those interested can view the YouTube video of Minnich's talk.

[I would like to thank LWN's travel sponsor, the Linux Foundation, for supporting my travel to Prague for ELC Europe.]

Send a free link

Did you like this article? Please accept our trial subscription offer to be able to see more content like it and to participate in the discussion.

(Log in to post comments)

Replacing x86 firmware with Linux and Go
Posted Nov 20, 2017 21:03 UTC (Mon) by **kjp** (subscriber, #39639) [Link]

> he stopped writing Bash scripts years ago in favor of Go. It is "easier and more reliable" to write scripts in Go.

My shell scripts use "set -e" and my python scripts raise exceptions, so errors don't pass silently. Also swift has really, really good syntax for exceptions. Wake me up when this passes GO.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 1:20 UTC (Tue) by **avasu** (subscriber, #99452) [Link]

The issue with most scripts is that they usually rely on other system binaries to do the actual "work", which makes it less useful in the case where we don't want to be dependent on other binaries. A simple example would be to try and add filesystem check in pure bash scripts. This is trivial in Go.

Python on the other hand can also be used, but the need for an interpreter and also the need for the packages, makes it a little harder imho, for the type of work that we are talking about here.

On a side note, it was really funny that as I was reading this article, there is another article which talked about a bug fix in Intel's ME https://www.theregister.co.uk/2017/11/20/intel_flags_firm...

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 12:46 UTC (Tue) by **abo** (subscriber, #77288) [Link]

That's true, but the point was, I think, that it's not hard or cumbersome to write reliable bash scripts. bash is a domain specific language for calling open(), dup(), fork(), exec() etc., so naturally it's not very useful without something to exec() into.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 15:21 UTC (Tue) by **jond** (subscriber, #37669) [Link]

> That's true, but the point was, I think, that it's not hard or cumbersome to write reliable bash scripts.

I've been writing (what I hope are, at this point, reliable) bash scripts for an embarrassingly long amount of my professional career, as well as my hobby stuff, and I couldn't disagree with this more. It's fiendishly hard to do it well.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 16:36 UTC (Tue) by **flussence** (subscriber, #85566) [Link]

That's true, though there is a really good static linter for bash/sh, shellcheck. I try to write good shell scripts and yet it still catches an awful lot of mistakes.

If someone gets better results by using Go (or dmd-run, or perl, or whatever), more power to them.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 6:33 UTC (Tue) by **hugelgupf** (subscriber, #106267) [Link]

I think it's worth mentioning here that Go is absolutely not mandatory to this project. The Linux-in-firmware component (which we have started calling LinuxBoot) can go with whatever userspace you like -- we just happen to use a small Go-based busybox-like initramfs.

Replacing x86 firmware with Linux and Go
Posted Nov 20, 2017 23:09 UTC (Mon) by **joib** (subscriber, #8541) [Link]

>Apparently, there is no overall web page for NERF itself, though some of the components Minnich talks about do have web pages.

I believe it's https://trmm.net/NERF

Replacing x86 firmware with Linux and Go
Posted Nov 20, 2017 23:12 UTC (Mon) by **jake** (editor, #205) [Link]

> I believe it's https://trmm.net/NERF

ah, thanks for that ... I searched for it but couldn't find it ... Ron's slides (and his employer's most famous tool :) didn't seem to have it. Will update the article.

jake

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 6:29 UTC (Tue) by **hugelgupf** (subscriber, #106267) [Link]

We'll be working on something more comprehensive (and representative of the LinuxBoot/NERF effort) soon.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 23:22 UTC (Tue) by **rahvin** (subscriber, #16953) [Link]

I commend your work but the question I've been asking all along is why is this necessary? Google should have the market power to force Intel to sell processors with the ME physically disabled. And even if Google alone doesn't have the market power they should be able to partner with Amazon, Facebook, Microsoft and the other cloud providers to demand CPU's with the ME disabled.

It bothers me greatly that Google and other cloud providers don't use this market purchasing power if they value the ME being fully disabled. Google purchase millions of servers a year and combined with the other cloud providers you constitute a major portion of the server market, I'd guess near 50% or higher. The cloud providers have the power to effect Intel directly and have not acted when the ME is a security threat of unknown proportion.

There are already blackhat exploits to provision the ME with user access to the system. Right now these attacks require user access but as well all know user access can quickly become remote access as the vulnerability is probed and more information comes to light. The ME is a massive security threat and if the cloud providers care about security they should be doing everything in their power to see processors without the ME made available.

Replacing x86 firmware with Linux and Go
Posted Nov 22, 2017 16:41 UTC (Wed) by **drag** (subscriber, #31333) [Link]

> Google should have the market power to force Intel to sell processors with the ME physically disabled.

Google's 'market power' is being expressed in these sorts of projects that help get Chromebooks running with Coreboot and such things.

> partner with Amazon, Facebook, Microsoft and the other cloud providers to demand CPU's with the ME disabled.

Believe it or not features like ME and vPro are actually _selling points_. Intel didn't add these features out of eviliness, having lights off management and back doors into computers is valuable features for people running large numbers of desktops and other systems. They can use them to help enforce corporate policy and such things.

This is what you get when you deal with closed source software.

These features are valuable, but unless they are open source then they are evil.

> It bothers me greatly that Google and other cloud providers don't use this market purchasing power if they value the ME being fully disabled.

Being a big customer does not give you godlike powers over another person's business by itself. If you want to have clout in purchasing decisions you have to have the ability to go and use competator's products.

Imagine if Google demanded Intel to produce binary-blob-free versions of their processors and Intel refused. What is Google going to do about it? Go to AMD and demand binary-blob-free versions of their processors and hope they comply? Migrate their server farms over to ARM processors?

Markets are self-regulating, but only if there is substantial competition. IP laws and such things limit competition and thus limit the ability for people to demand large corporations to obey. Production of unencumbered processors like RISC-V is the ultimate 'out' for this thing and if they are successful will solve these sorts of regulatory issues, but in the meantime Intel is only going to care if it costs them customers.

Hopefully AMD takes the hint.

Replacing x86 firmware with Linux and Go
Posted Nov 20, 2017 23:26 UTC (Mon) by **joib** (subscriber, #8541) [Link]

Another interesting thing which was mentioned in a comment after the talk was that as of the Skylake platform it's possible to disable the ME by just flipping a bit on the flash, instead of the slightly ad-hoc method me_cleaner is using. Seems this is related to a "High Assurance Platform" program for three-letter agencies. So apparently the idea is that spooks don't want the ME on their own systems. Whereas the public is not supposed to get a choice.

Anyway, some clever people figured out how to flip this bit even if you're not the NSA:
http://blog.ptsecurity.com/2017/08/disabling-intel-me.html

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 1:07 UTC (Tue) by **ThinkRob** (subscriber, #64513) [Link]

This may or may not be feasible for systems with Boot Guard though. It seems that Boot Guard does $something when this bit is set, but as of yet I don't know if any testing has been done to determine what it actually does (or, for that matter, if the various vendors which support Boot Guard have implementations that will allow the machine to boot once the HAP bit is set.)

It's also worth noting that the HAP bit trick sorta requires you to implicitly trust that the ME is, in fact, disabling itself. AFAIK there's nothing to prevent it from silently switching into "do evil things, but more subtlely" mode. Given the assumed target audience of HAP, I think that's probably unlikely. Still though, given that the point of all of this is well-intentioned paranoia, it's worth considering.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 18:25 UTC (Tue) by **cyphar** (subscriber, #110703) [Link]

You can use me_cleaner in both the "remove as many modules as possible" and "enable HAP / MeDisable bit" modes -- which should reduce your concerns. This is what I've done on my X220, and I believe it is what Purism is doing for their new laptops.

Replacing x86 firmware with Linux and Go
Posted Nov 22, 2017 0:32 UTC (Wed) by **ThinkRob** (subscriber, #64513) [Link]

AFAIK this doesn't work for laptops where Boot Guard is enabled and enforcing using the keys burned in during board manufacture.

I have an X230 running coreboot w/ a stripped + HAPd ME, but with subsequent ThinkPads actually modifying the ME seems to cause Boot Guard to brick the whole system. And since the key is supposedly OEM-specific and burned into the chipset (CPU?) itself, it's not like you can replace it yourself.

What I *don't* know is if setting the HAP bit but leaving the rest of the ME image the same will trigger Boot Guard's hissy fit.

(The above is based on my understanding of Boot Guard, which may be comically wrong. Please correct if it is!)

For further reading
Posted Nov 21, 2017 0:17 UTC (Tue) by **corbet** (editor, #1) [Link]

...see this advisory from Intel on the results of a security review of its management engine. In short: there's a lot of systems out there in need of a firmware update.

For further reading
Posted Nov 21, 2017 12:33 UTC (Tue) by **danpb** (subscriber, #4831) [Link]

But don't expect this will be the end of it. With the level of complexity in the firmware, and its increased profile amongst security researchers, there's no reason to expect it will fair better than any other comparative software out there. IOW, it is reasonable to expect a steady stream of security vulnerabilities being reported for years to come... :-(

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 0:56 UTC (Tue) by **pbonzini** (subscriber, #60935) [Link]

There is one important thing that SMM does that is not just vendor crap. By accessing flash memory and handling the UEFI persistent store, SMM provides the trusted base for Secure Boot. So, if they want to implement some kind of code signing, they need to do it in SMM.

In addition, APEI functionality (ACPI Platform Error Interface) is usually implemented via SMM.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 16:47 UTC (Tue) by **luto** (subscriber, #39314) [Link]

I don't really buy this. I see no reason that the secure part of flash ever needs to be written after boot. A high quality implementation of UEFI should be able to queue up variable writes such that the next boot will find them, validate them, and apply them before ever executing code off the disk.

I'm not at all sure that Intel's flash hardware can do this, but it's surely doable with a second plain I2C flash chip on the system SMBUS.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 21:06 UTC (Tue) by **pbonzini** (subscriber, #60935) [Link]

Well, if you can throw additional hardware at the task that's certainly a possibility. :-) Instead of a second I2C flash you could also delegate UEFI variable services and APEI to a microcontroller (also on SMBus).

Oh, and there's actually another thing that SMM does, which is orchestrating CPU hotplug. And that's an ugly one because, due to the asinine default choice of SMBASE made in the 386SL(*), it's the only case that really, really needs to bring *all* processors at the same time on SMM.

(*) The OS could overwrite 0x30000 and gain access to SMRAM. "Why 0x30000?" is near the top of my list of x86 unanswered questions, even higher than "What the heck was CR1?".

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 22:08 UTC (Tue) by **luto** (subscriber, #39314) [Link]

True, although adding a whole microcontroller would cost more than just an I2C flash chip. The only service that I think is really needed is some NVRAM that can be written outside SMM to queue up variable writes. I don't know whether current Intel chips can lock down part of boot flash while leaving some unlocked.

> The OS could overwrite 0x30000 and gain access to SMRAM. "Why 0x30000?" is near the top of my list of x86 unanswered questions, even higher than "What the heck was CR1?".

I've never understood why the reset/SIPI vectors and the default SMBASE don't point to addresses that are unconditionally in the boot flash. I guess that all that's really needed is the ability to securely run something from flash before any OS code can possibly execute on a newly hot plugged CPU. Is this not actually possible without kicking all CPUs into SMM before powering up a new CPU?

Replacing x86 firmware with Linux and Go
Posted Nov 22, 2017 0:02 UTC (Wed) by **pbonzini** (subscriber, #60935) [Link]

It's certainly possible, the problem is that 0x30000 doesn't point into flash. :(

Defaulting SMBASE to 0xA0000 would have made a lot more sense.

Replacing x86 firmware with Linux and Go
Posted Nov 22, 2017 0:04 UTC (Wed) by **pbonzini** (subscriber, #60935) [Link]

I forgot a sentence: 0x30000 doesn't point into flash, so you need to bring everyone in SMM when the newly-hotplugged CPU does SMBASE relocation.

Replacing x86 firmware with Linux and Go
Posted Nov 22, 2017 6:45 UTC (Wed) by **MakeHinduGreateAgain** (guest, #119801) [Link]

I thought the better description is that install my customer SMM handler in Linux payload (As https://github.com/rminnich/linux/commits/monitor). The RAS related function code is resident in SMM and some of the registers are SMM access only if my memory is correct.

And a little anecdote is default UEFI SMM handler always bright up other core APs into SMM synchronously (the CPU troop is spining and waiting). If you have more AP cores, the more latency you will get. (Although there are some security implication here). A modern server machine has at least twelve cores. There will be some performance and latency issue.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 4:36 UTC (Tue) by **jhoblitt** (subscriber, #77733) [Link]

What is the status of UEFI with ARM? Is that a done deal for the "server" class ARM systems that are being launched?

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 12:10 UTC (Tue) by **joib** (subscriber, #8541) [Link]

Yes, ARM servers will feature UEFI and ACPI. See https://lwn.net/Articles/584123/

The relevant ARM standards are called SBSA (Server Base System Architecture) and SBBR (Server Base Boot Requirements).

Interestingly, it seems OpenPOWER uses something pretty similar to NERF called 'petitboot'; a simple (well, compared to UEFI at least) firmware that loads a Linux kernel + userspace from the flash rom, that system then kexec()'s the final distro kernel.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 14:09 UTC (Tue) by **mirabilos** (subscriber, #84359) [Link]

Would be happy to replace the ME on my IBM Thinkpad… but only with something that then provides seabios to the to-be-booted operating system on the main CPU, so I can boot DOS, OS/2, old-style BSD, etc. as well.

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 14:38 UTC (Tue) by **merge** (subscriber, #65339) [Link]

That's no problem. I always configure coreboot to use the latest SeaBIOS as payload, so far.

In my case, it'd be even better use GRUB instead, and don't install GRUB via Debian on disk. I simply haven't had the time to dig in. Are there any nice posts about coreboot+GRUB and how to install and configure Debian without GRUB, and how to maintain grub's config files on disk?

I guess there are even less GRUB releases as there are SeaBIOS releases that would make me want to re-flash :)

Replacing x86 firmware with Linux and Go
Posted Nov 21, 2017 14:42 UTC (Tue) by **merge** (subscriber, #65339) [Link]

sorry, my comment was off-topic. I assumed you run coreboot.

Replacing x86 firmware with Linux and Go
Posted Nov 22, 2017 14:46 UTC (Wed) by **cortana** (subscriber, #24596) [Link]

If you run the installer in advanced mode, you can choose to not install any boot loader. Otherwise you can simply the `grub-pc` or `grub-efi-amd53` packages.

Replacing x86 firmware with Linux and Go
Posted Nov 22, 2017 8:26 UTC (Wed) by **nim-nim** (subscriber, #34454) [Link]

It's not extensible but it ships with a compiler… WTF.

The Go ecosystem produces wonderful apps, but they need to fix their ship to production story. Piling up massive amounts of intertwined Go modules with no release discipline, massive forking and wheel reinvention, last-mile compilation of this mass in a single static binary, only goes so far. Security researchers are going to have a field day if Go software escapes the world of GitHub-plugged devs.

Go is only escaping black marks because there are few public Go deployments where everything is not continuously rebuilt (Not that the result is secure, it's just too mutating to be easily audited, so researchers have chosen to look elsewhere so far. Remember when they decided to look at Java and uncovered pervasive years-old vulnerabilities?).

Copyright © 2017, Eklektix, Inc.
Comments and public postings are copyrighted by their creators.
Linux is a registered trademark of Linus Torvalds