DRAMMER

Description: Leo and I discuss last week's major attack on DNS, answering the question of whether or not the Internet is still working. We look at Linux's worrisome "Dirty COW" bug, rediscovered in the kernel after nine years. We address the worrisome average lifetime of Linux bugs; share a bit of errata and miscellany; and offer an in-depth analysis of Drammer, the new, largely unpatchable, Android mobile device Rowhammer 30-second exploit.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-583.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-583-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here with the latest security news. He's going to talk about that, of course, widespread Internet DDoS attack from last week, and have info on a brand new exploit. He was embargoed last week and can talk about it now: Drammer. And we'll show you how to test your Android device for it. It's coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 583, recorded Tuesday, October 25th, 2016: DRAMMER.

It's time for Security Now!, the show where we cover your security and safety and privacy online with this guy right here, Steve Gibson of the GRC Corporation. Live long and stay safe. There he is.

Steve Gibson: Yes. So I promised last week that I already knew the title of today's podcast, and I was under embargo. I was brought in by the guys who did the Flip Feng Shui some time ago, a slightly different crew that also involves three researchers from Santa Barbara in addition to the guys at the VU University in Amsterdam. I was informed of what was going on under the agreement that I would abide by their disclosure timeline. And of course that was before the Internet melted down at the end of last week, which actually affected me in two interesting ways that I'll share when we get to that. So there was some competition for, like, what's the week's biggest story?

So anyway, I settled on Drammer, which is the name they've given to this next-generation Rowhammer DRAM memory bit-hammering exploit. But this is significant because they answered the question of whether this could be done on ARM platform mobile devices in the affirmative.

Leo: Uh-oh.

Steve: Which was not believed to be possible. And remember that I described the Flip Feng Shui as a sublime hack, I mean, just - and I'll quickly review it when we start talking about this at the end of the podcast. But this is equally sublime. I mean, these guys are good. So I'm just so happy that they're on our side, and that they're working to reveal flaws and responsibly disclose as they have to Google. Google informed their upstream partners - or downstream partners, I'm not sure which way the stream flows - a month ago. Although while there is some mitigation possible, the underlying problem, as we've discussed when we've discussed Rowhammer for the last couple years - it was in 2014 that this first emerged as a problem - it is a hardware flaw.

And so, while you can sort of dance around and make it more difficult to leverage the flaw into an exploit, everything we see as we look at the history of these things over time is that something that looks like a problem, but, oh, it's nothing to worry about? Well, give it a year, and someone figures out how to make it a big worry. So that's the main focus. But of course we're going to talk about last week's powerful denial of service attack.

Linux has a new problem that Linus himself wrote about. It was actually a problem he took some responsibility for nine years ago, known as "Dirty COW." COW, C-O-W, is the abbreviation for Copy On Write. And it turns out there is a longstanding race condition in all Linux kernels which is really critical. Essentially, any web-exposed Linux server is vulnerable. So all of our listeners who have web-facing Linux servers need to understand what's going on and get themselves patched. Oh, and it was discovered in the wild. So this is unlike, for example, Drammer, which is a researcher-driven theoretical problem that they've made real. This was discovered from traffic analysis happening to someone's server. We also, sort of related to that, an interesting look at the average lifetime of Linux bugs. There was a study done about six years ago, and that's been updated by a Google researcher. And nothing has changed, unfortunately.

We have a little bit of errata and miscellany. And then, as I said, we'll take a real close look at the details of this latest Rowhammer exploit, Drammer, as they call it, which is quite a concern. So lots of good stuff to talk about.

Leo: It's going to be a good show. Looking forward to it. Let's pause briefly before we get to the news of the week. And the news of the week is going to talk a little bit about the danger of Internet of Things devices. And here we go with an Internet of Things device.

Steve: And it sort of brings the point to bear, along with this week's Picture of the Week in the show notes, that not all of these connected things really justify their connectivity. I mean, it's sort of a reach, for example, when you've got, like, an Internet-connected coffee pot.

Leo: Right, right.

Steve: It's like, okay, well, you know...

Leo: Okay.

Steve: Yeah, I guess I could start brewing while I'm stuck in traffic so the pot is ready for me when I get home, but...

Leo: But there are some things that are really useful, too.

Steve: Yes.

Leo: And so that's the thing is be judicious. I mean, you can't not connect a router to the Internet; right? So make sure you get one that's well designed, I guess.

Steve: Yeah, yeah. Well, and several people sent me this The Joy of Tech cartoon.

Leo: I have it here, so I'll show it.

Steve: It was titled "The Internet of Ransomware Things," where the smoke alarm up on the ceiling is saying, "30 bucks in bitcoin, or next time I smell smoke, I might just let you sleep." Or the dishwasher says, "Your dirty dishes can wait. I'm busy mining bitcoins." The lamp hanging from the ceiling and the camera sitting on the counter say, "Excuse us while we participate in a DDoS attack."

Leo: Yikes. That's a little too close to home, huh.

Steve: It is a little too close, yes. So anyway, and this is just full of - it shows a little robot vacuum zooming around, saying, "Wire my hacker $100 or I'll reverse my motor and blow dirt all over the place."

Leo: That's pretty funny. This is Nitrozac and Snaggy. We love The Joy of Tech. And of course they do the album art for this show. Nitrozac did that great illustration of Steve, which we're going to maybe have to update. I think your mustache has gotten a little grayer.

Steve: Yeah. And I always look a little snootier in that than I [indiscernible].

Leo: We could get another one. We'll go get another one. She's very kind. She updates us.

Steve: Anyway, so clearly they know their IoT problems because they pretty much nailed every bad thing that could happen. I was tempted to call this "Attack of the Light Bulbs," or then in homage to an old Internet meme that you'll of course remember, "All

your light bulbs are belong to us."

So my own intersection with last week's problem was when I received, at about 7:30 in the morning, an iMessage from Sue, my bookkeeper. She had been away from the office for about a week, traveling with her laptop and checking in constantly to deal with any sales.grc-related mail stuff. And she sent me a note saying that Eudora - yes, we're all still using Eudora - was returning "GRC.com address not resolved" error. And I thought, whoa, that's odd. And so I shot her a text note back and said, well, okay, that doesn't really sound like our problem, but I'll look into it.

And then, like maybe two hours later, I got an alert saying that one of GRC's eCommerce transactions had failed. And I do, if the GRC server is unable to connect to our merchant gateway through which we process customer credit card transactions, it waits like a few seconds and tries again. So it does 10 retries and then declares a failure, and then returns a response to the user through the web page that something outside of our control is preventing us from successfully completing the charge. You know, it's not you've got the zip code wrong or your street address doesn't match. It's something else. So that happened. Someone was determined to buy a copy of SpinRite, so I had to listen to 30 individual sets of - or three sets of 10, so 30 total error messages. It generates the Star Trek hailing whistle, and then a voice says...

Leo: [Whistles]

Steve: ...something about - exactly - eCommerce transaction retry, and then does that 10 times, and then eCommerce transaction failure. So I always know what's going on. So then I started seeing the news about a DNS-related outage. And that, of course, put all the pieces together for me. That explained why Sue, wherever she was, she was down like in Texas or somewhere, where whatever DNS server her location was using was unable to obtain the IP for GRC. And suddenly I thought, ah, I'll bet you that that's what's happening because of the coincidence that GRC's eCommerce system was unable to obtain through DNS lookup the IP for the merchant gateway. But I was able to get it from here as a Cox subscriber.

So I looked up the IP, jumped over to GRC's server, and dropped an entry into the hosts file. And as we all know, the hosts file, even today, is - and this is on a Windows 2008 R2 server, so a recent server - it looks in the hosts file first. Well, what I found, interestingly, was I had already commented out the line I was going to put in. In other words, this had happened previously. So all I did was I removed the pound sign from the from of that line because the IP had not changed from whenever it was I had done that before. And then immediately eCommerce transactions started to process again.

So we had a big DNS problem. And of course everybody knows about last week's massive outage. Now, I think what surprised people, though, was, well, several things. Our listeners know enough to wonder why DNS caching didn't mask the problem. And I will explain why that is. But essentially there was a company based on the East Coast, Dyn, that is the authoritative nameserver for, as we learned, many major providers. Meaning that that service - and, I mean, Dyn is a major DNS provider.

Leo: So are they one of the 13? Or are they a subsidiary to the main phone books?

Steve: So, yeah, they're not the root servers.

Leo: The root servers, yeah.

Steve: They would, for example, they offer the servers that a domain registrar points to. So, for example, if you look up Twitter.com's registrar, whoever they're using - very much the way, for example, I'm using Hover right now. So for the domains I have at Hover, I've given them the Level 3 servers that GRC uses. And so those are the authoritative domain nameservers for GRC. That is, that's where GRC.com's DNS records are. And so it turns out that many of these large companies have outsourced their DNS. You know, being me, of course, I don't. I do my own. And in fact I actually have the root server, and Level 3's are slaves. So they actually handle all of the traffic, but they obtain what's known as the GRC.com zone record from my master server, which is a FreeBSD Unix machine.

So these major companies just thought, you know, there's no value to be added by DNS. And they also wanted, you know, I mean, they're much larger than GRC. I'm fine having two strong Level 3 DNS servers serve GRC's DNS. But Twitter and Amazon AWS and huge, huge Internet presences, they really need multiple physical presences, you know, multiple points of presence - so West Coast, East Coast, international and so forth - so that the world's DNS queries are not all just pounding on one or two servers, but are being distributed by essentially the equivalent of a content distribution network for name services. And that's what Dyn offers. The problem is...

Leo: So if I'm using Hover - I didn't go down because I was using what they chose. But you were doing your own.

Steve: Correct.

Leo: Got it.

Steve: And so, yeah, so you could use Hover's DNS; or, as I do, I could point my Hover registration record at Level 3's DNS. And instead they, like Twitter, have pointed their registration's record at Dyn's DNS servers.

Leo: Oh, so it's because you were using Level 3 that you got hit.

Steve: No. It's actually that whoever - well, for example, my merchant account company was using Dyn because they're all...

Leo: So your GRC wasn't down.

Steve: Correct.

Leo: But you couldn't take a credit card.

Steve: Correct.

Leo: Got it.

Steve: And wherever Sue was, her DNS provider was somehow affected by this, too. So anyway, it was a wakeup call that so many major sites could be taken down by an attack on one part of the infrastructure. And we've often talked about, because we've talked about DNS often, it is UDP protocol, which is meant to be lightweight. And what that means is that it is inherently spoofable. TCP connections cannot be spoofed because you need that three-way handshake. Packets have to be able to make a successful roundtrip from each endpoint to the other and back in order for a TCP connection essentially to be completed and connected. But that's an overhead that simple lookup services minimize, so they simply just send off an unverified UDP packet towards its destination. And if they get a response, that's great. It was very quick and inexpensive. If they don't, they wait a minute, well, not a minute, but a few seconds, and then they try again. So they take responsibility for trying to get a response.

But what that also means is that, for example, they're able to deliberately rewrite the outgoing packet's apparent source IP to mask their actual source IP. And if those illegitimate source IPs are allowed out onto the Internet, and that's something we've talked about often, the whole issue of egress filtering, where ISPs currently do allow packets that cannot have, I mean, whose packets clearly are known, could be known to the ISP to be falsifying their return address, essentially, their source IP. All the ISPs know if packets are leaving that could [audio glitch], if there is an illegitimate source IP. They arguably should be dropped onsite. They should just simply be ignored. But that's not happening today.

So there is actually - there is an RFC and a proposal, BCP 38. And the RFC is 2827, which is titled "Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing." And there's even an acronym for a movement that is trying to get itself going called SAVE, which is Source Address Validation Everywhere, the implication being there are even chokepoints where - and I've talked about this, too. A router in the middle of the Internet has a routing table that tells it which connections to send packets to based on where they're going. But that can also infer which connections packets can have come from. And so even routers in arbitrary locations could be made smarter and to recognize packets which cannot be valid. But, again, nobody's doing that.

And we talked about this last week. It was funny because you and I were talking about IoT and about source address spoofing. And I mentioned that, unfortunately, there were other attacks that did not rely on spoofed source addresses. And so those would pass right through such filters. And in fact, the attacks last week did not rely on whether they spoofed their source addresses or not - and they may have because they were DNS attacks - but they didn't need to.

Okay. So Dyn wrote something, sort of a, not saying very much, corporate-level speak. In their blog posting under "What We Know," they said: "At this point we know this was a sophisticated, highly distributed attack involving tens of millions of IP addresses. We are conducting a thorough root cause and forensic analysis and will report what we know in a responsible fashion. The nature and source of the attack is under investigation, but it was a sophisticated attack across multiple attack vectors and Internet locations. We can confirm, with the help of analysis from Flashpoint and Akamai, that one source of the traffic for the attacks were devices infected by the Mirai botnet. We observed tens of millions of discrete IP addresses" - assuming those were not spoofed because, of course,

if they're just pseudorandom, then they could easily be, not tens of millions, but 10 pretending to be more. But we now know that this is a huge botnet.

Okay. So what was interesting, as I dug into this, is that I looked at some of the traffic that this Mirai was using, and it looked disturbingly familiar because I realized that it was using a weaponized version of the technique I used six years ago when I wrote GRC's DNS Benchmark and GRC's DNS Spoofability Testing. For the Benchmark, I explicitly need to bypass caching, or selectively bypass caching, because the DNS Benchmark wants to determine how fast your local DNS caches are, but it wants to separate that from the speed of the authoritative nameserver that is the ultimate source of the DNS information.

The way I do that is I create made-up domain names, pseudorandom gibberish, for example, dot something dot com. And so the idea is, since that pseudorandom gibberish cannot yet exist in the local DNS resolver, it is forced to shrug and forward it to the authoritative nameserver to see if that's a domain that the authoritative nameserver knows about. So this is a deliberate cache-busting technique. And, unfortunately, it's what Mirai uses, whereas I was very careful never to use this to overload any DNS server. And in fact, one of the things that the DNS Benchmark does, it wants to determine the reliability, the response reliability of the servers that it is benchmarking.

So I do not want to clog the channel which would cause a false positive packet loss. So I meter these queries out, only allowing one to be in flight at any time to any given server of the batch that are under test. Of course, Mirai wants to attack remote DNS servers. And, so, see, what it needs to do, it must bypass its local ISP cache because, if it simply asks for Twitter.com, well, if the local cache run by its ISP wasn't already aware of Twitter.com - and, I mean, it almost certainly would already have it in its cache. But if it wasn't, then it would have to make a query to Twitter.com's authoritative nameserver in order to determine what Twitter.com's current IP is.

And so that's where it goes to the root, to the com, the dotcom servers, essentially, in order to look up Twitter.com. Then it goes to Twitter.com's server to see what Twitter.com's current IP is. So but the point is that, if say an aberrant DVR - because we know that a lot of these were commandeered DVRs. If the DVR made a query once, it might force the local DNS server to look up Twitter.com. But now it's in that local DNS server's, that local resolver's cache. So all subsequent queries for Twitter.com would not bother the actual Twitter.com DNS server because they're now cached.

And so what I did many, many years ago, and even do now for the spoofability testing, is I deliberately make up fake domain names, subdomains, in order to get queries from - in order to send queries through the cache to get to the actual servers. These guys are doing it to attack the servers. So that's a cache-bypassing attack which, as we've seen, is super effective. Now, our listeners wondered why a DNS server going down, an authoritative DNS server, why wouldn't that be buffered by valid caching for valid domain queries? And the answer is it certainly was.

But the problem is, not only do you have the IP address in a domain record, but the domain owner is able to set the cache duration. As part of the DNS record, they're able to specify, once this is queried, how long they want that result to survive out in the Internet's caches. And, unfortunately, not everyone has those set optimally. There are instances where you would deliberately want it to be shorter. For example, when many years ago I was moving GRC from behind my two T1s, where it lived for a long time, over to Level 3, that meant that GRC.com had to - I was going to completely change all the IP addresses from the block that I had with Cogent who was my bandwidth provider, over to a block that I had been assigned by Level 3.

So what I did a couple days before this planned changeover, and everyone will remember that it was - "planned" is being kind because suddenly I lost my connection. And I called them up. I said, "What happened?" And they said, "Didn't you get the mail?" I go, "No, I haven't had that email address for 10 years." So, anyway, it was a little bit of a fire drill around here. But the point is that, knowing that it was going to happen, I shortened GRC's DNS to 15 minutes, and I did it a day before.

So what that meant was that essentially I drained the cache, or at least I drained the long expiration cache out of the Internet. And it took, for example, if it had been previously set for eight hours - I don't remember now if it was eight or 24. But I had to wait at least - I had to keep the old IP at least as long as the cache had been, so that all DNS servers on the Internet would have come back and refreshed themselves. But when doing so, they would be refreshing themselves with the 15-minute, the very short-duration DNS record, so that would then force them to come back every 15 minutes and get an update.

So what that meant was, when I did bring up the new servers at Level 3 and then changed GRC's DNS, within 15 minutes the entire Internet was informed of that change because I had planned it that way. And so there wasn't a big outage. If my DNS, for example, had been set to 24 hours of caching, and something had forced me to make an unplanned emergency change, it would be staggered through the 24 hours because DNS resolvers all over the Internet would have different comeback times. But the oldest of the caches wouldn't have expired for a full day before it got news of the new IP.

So the point is that, if these major companies had their caching set to 24 hours, there still may have been some outage, but it would have been spotty. So the point is that the authoritative nameservers were effectively DDoSed off the air. So nobody was able to refresh their cache with new information. But if these major, like Twitter and others had set very long records, and they could do that if they had a known stable IP address, then they could better survive a few hours here and there of a shortage. Some people would see an outage. Others, their local resolvers would have been able to keep the cached information for a much longer period of time and would have never had to come back and update themselves.

Anyway, so that's the whole story of what happened. Stepping back from this a little bit, I was reminded of a comment that our friend Bruce Schneier made recently regarding IoT device security. He said: "The market can't fix this because neither the buyer nor the seller cares." Now, of course, that's not true of our listeners. But look at all the DVRs that are out there.

Leo: I think people are caring a little more than they did before.

Steve: Yeah.

Leo: They'll care. XiongMai, that made most of these devices, has already started to update them and do recalls and stuff. They are a Chinese company.

Steve: Well, and to threaten suing Brian Krebs and others.

Leo: XiongMai is suing Brian Krebs?

Steve: Yes, saying that he is maliciously reporting and blaming them for the whole attack. They're denying that they are as responsible as they are. So it's like, okay, well, you know, play the game.

Leo: That's a problem.

Steve: Yeah.

Leo: That does validate what you just said. They don't care.

Steve: Yeah. So I should mention also that we've talked about the Shodan search engine, which is scouring the Internet and logging anything it finds. There is a simple IoT scanner which makes a query of Shodan for a user's IP to see if any given user's IP exists in Shodan's database.

Leo: Oh, that's nice.

Steve: It's iotscanner.bullguard.com, I-O-T-S-C-A-N-N-E-R dot B-U-L-L-G-U-A-R-D dot com. It gives you a nice presentation and just explains that, if you click this button, it'll check Shodan for you to see if you are listed. I pressed it, and I wasn't. I passed it on to my buddy who's gone IoT crazy, although he's got the three dumb router configuration, and he's isolated. And, good, you are not public on Shodan. You can do a deeper drill as the next phase; but they warn you that, if you do, that may bring you to Shodan's attention, where you may not have been there otherwise.

Leo: Why would I not want to be brought to their attention? I am now, apparently. I just pressed the button.

Steve: Yeah. And I think that probably, you know, they're doing enough job of scanning. And, good, no vulnerability found again.

Leo: Now, this is here, of course, and we got Astaro routers in between us and security boxes in between us and the outside world.

Steve: Well, so of course, that's the problem, too. It's why I referred to, someday, considering doing a comprehensive 65K, or 64, depending upon how you count, port scan because what a lot of these IoT devices do is they leverage universal plug-and-play in people's routers in order to open up incoming ports to themselves. I mean, that's the only way something out on the Internet can find your DVR, if the DVR has, as part of its setup and configuration, opened up an incoming port to itself. And unfortunately, ShieldsUP! doesn't currently find that. And it's very labor intensive. It's not something

you can scan for inside your network. You need somebody to scan for you on your behalf outside your network. And so it might be worth having our listeners use this iotscanner.bullguard.com, just to make sure that there's nothing that they've got flapping in the breeze publicly that they're not aware of.

Leo: You can't provide it with an IP address. It just checks the network you're currently on.

Steve: Correct.

Leo: Be nice if I could say - because I'd like to check my home address right now, but I'll have to [crosstalk].

Steve: It would be nice, although, of course, that's subject to abuse.

Leo: Right.

Steve: I do the same thing with ShieldsUP!. I will test where you are connecting from.

Leo: Right.

Steve: But I do not let you put an arbitrary IP address in because, of course, people would then be using me to scan other people.

Leo: Right, right, exactly.

Steve: That's not...

Leo: Check and see how safe they are.

Steve: Yeah, no.

Leo: Yeah, I don't blame you.

Steve: So anyway, you know, there's nothing we can do except protect ourselves. And I don't know how this gets resolved. As Bruce said, the problem is, you know, you go to Amazon, and people are clicking buttons to download light bulbs and baby cams and all this stuff. And they're fun, but they're creating a problem, not only for - I was explaining to my friend, who's like, he's always showing me his pictures around his home when we're out grabbing a bite. And I go, "Uh-huh, well, that's not good," you know.

And at this point these devices are of interest to attackers for use in commandeering them for attacks like we saw last week. That's sort of the low-hanging fruit. Eventually, they're going to get around to poking around inside people's networks using, I mean, this absolutely can be done. If you've got these DVRs which have had malicious software installed in them, this Mirai botnet, then that's - and it does exist only in RAM, as we know. If you reboot these devices, that flushes it out. But if they remap a port and go public again, they're infected within five minutes because there are so many of these things now scanning for others.

But what this means is that you've got remotely controlled malware on your internal home network, and it has full visibility to the net it has access to. So there's never been a better case for network segmentation that we've been talking about now for quite a while. You have to have these, you know, your high-value computing resources separated from stuff that you can't and no one can reasonably represent are secure. And ultimately we're going to see people beginning to have - right now it's just using these things for attack. They're going to end up becoming points of intrusion. It's inevitable. So it's called...

Leo: Lovely news.

Steve: Yes. It's called "Dirty COW." Linus wrote...

Leo: This is not what Donald Trump called Hillary Clinton, I don't believe.

Steve: No, that was "nasty woman."

Leo: That's right, okay.

Steve: Dirty COW. So on last Thursday - he does pronounce his name "Linus." I always feels self-conscious saying "Linus."

Leo: Does he?

Steve: Yes.

Leo: Everybody's saying, "It's Linus, it's Linus." It's Linus, huh?

Steve: I made a point once of listening to him say it of himself. I found some audio of him pronouncing his own name. It's Linus. So he wrote: "This is an ancient bug." He called it "ancient." Okay, well, maybe in Torvalds' timescale.

Leo: It goes back to almost the beginning; right?

Steve: Well, nine years.

Leo: Okay. That's a long time.

Steve: So, but, yeah, okay. "An ancient bug that was actually attempted to be fixed once" - he says in parens - "(badly) by me 11 years ago in a commit," and he gives the commit number. And it was called "Fix get_user_pages() race for write access." Then he says: "But that was then undone due to problems by a subsequent commit which was ('fix get_user_pages bug')." So he made a fix that he's now unimpressed with, which was then fixed. And that fix broke the kernel in a way nine years ago that allows this race condition.

And you know, but a lot of our listeners don't, a race condition is sort of a famous problem in computing in general, and actually before that in electrical engineering. It's any situation where two things are not supposed to happen at the same time. That is, typically, they're trying to be mutually exclusive. So that, for example, each one checks to see if the other one has happened. And, if not, then it happens, and it blocks the other one. And this is a problem. I've been dealing with it for years. For me it's just now at an instinctual level I'd handle this because in multithreaded code you've got multiple threads of execution. And when we've got multiple cores, we actually have multiple processors running around in our code at the same time.

There are some things, for example, say that you had an increment, where you want to increment a value in memory. Well, you do that by reading the current value into a register, incrementing the register, and then writing it back. But say that one thread read the value from memory, and at that instant there was a context switch, so that thread was paused, while another thread ran. And that other thread read the same value into its context. And then the context switches back. And the first thread increments that and writes it back. Then the second thread increments its copy and writes it back.

Well, notice what happened. We intended to have two increments, but only the second one survived because of a race condition where they both had essentially nonexclusive access to a single location in memory. And this causes all kinds of problems in computing because, for example, you have things like reference counts in garbage collection where a variable is dynamically allocated, and you need to keep track of how many different places it's referenced, and you increment that going upwards. And when you no longer are using that variable, you decrement it going downwards. The last process or user to cause it to be decremented to zero allows that to be freed. Well, if your counting has been faulty as a consequence of race conditions, all kinds of havoc. That's where you get memory leaks, and buffers that are deleted when they shouldn't be, and nightmares.

So this can happen in electronics, where you have two edges of two different signals that are not supposed to happen at the same time, and it can happen in OS kernels. So there is, has been for nine years, an unappreciated race condition in the Linux kernel. And this was discovered by a Phil Oester, who is a Linux developer who routinely logs all HTTP traffic to his web servers and analyzes it for forensic purposes, just to kind of keep an eye on what's going on. And so a few days ago, or last week, he's looking at his logs and processing them however he does because no doubt that's a lot of data, which is why most people don't keep track of every single query, and he discovers this thing being done to his Linux servers in the wild.

So this is a high-severity, nine-year-old bug in the Linux kernel. Any system running Linux on a web-facing server is vulnerable. Somebody knows about it and has been

exploiting it. It allows an attacker to gain write-access to memory that should be read-only. And it's a big concern because it's not difficult to develop exploits that work reliably. It's located in a section of the Linux kernel that's part of virtually every distribution of Linux that's been released for the last nine years. It has been fixed.

And so Red Hat is aware of it. I saw an announcement from them talking about, you know, this is a zero-day. Update your Linux kernels. So everybody who's running Linux on web-facing systems needs to get themselves updated because no doubt bad guys, I mean, now the information is out there. This is where we see a quick escalation in exploits, and it's not difficult to do. Which means people are going to be looking for unpatched Linux kernels. And now we have the problem of little, small, micro Linux kernels in IoT devices that are never going to get patched and that have been sitting on networks. Maybe they're in closets; maybe they're people's routers. But they're certainly web-facing if they're exposed to the Internet. We're in for some interesting times, Leo. Wow.

Leo: Yeah. I mean, it's web servers, we should point out. So if you're running Linux at home, unless you're running a server on it, you're not...

Steve: No, it's web-exposed servers. So anything with an open port is a server by definition.

Leo: Okay, all right. Okay, all right.

Steve: Yeah, yeah. So it's not just classic web servers. It's anything that is responding to unsolicited incoming traffic, like all those DVRs that have opened a port and are being scanned for and located.

Leo: So it could be a remote exploit. But you would have to, well, go to ShieldsUP!. Make sure you don't have any open ports. You'd have to have a port open for something on your Linux box. Which many people do. I mean, if you allow SSH into a Linux box, if you...

Steve: Precisely, precisely.

Leo: Yeah.

Steve: Yeah. I mean, this is bad.

Leo: Or a mail server.

Steve: Oh, and it takes five seconds, by the way. I forgot to mention that.

Leo: Oh, well. You didn't mention that. Oh, never mind, then. Nothing to worry about here. All right. On we go.

Steve: So he spells his name "Kees," but it's pronounced "case." Kees Cook is with Google. He describes himself, saying: "I work for Google on ChromeOS security. Previously, I worked for five years at Canonical as an Ubuntu security engineer. My work is to stay alert, curious, and creative while keeping one step ahead of the bad guys. While I'm not working, I have been known to play with MythTV and generally poke around at video formats."

So last week he wrote in a blog post: "In several of my recent presentations, I've discussed the lifetime of security flaws in the Linux kernel. Jon Corbet did an analysis in 2010 and found that security bugs appeared to have roughly a five-year lifetime. As in, the flaw gets introduced in a Linux release, and then goes unnoticed by upstream developers until another release five years later, on average. I updated this research for 2011 through 2016 and used the Ubuntu Security Team's CVE Tracker to assist in the process.

"The Ubuntu kernel team already does the hard work of trying to identify when flaws were introduced in the kernel" - which is really neat to know, like, where this came from, which whenever I find a bug in my own code, I just - everything stops, and I say, okay, how did this happen? Because I want to work on the process rather than just fix the problem. I want to work on perfecting the technique of not having this happen again. So I'm delighted that they're looking back to determine where this came from to see if they can, you know, if there's anything they can do to improve the process. So, he says: "So I didn't have to redo this for the 557 kernel CVEs since 2011."

Leo: Wow.

Steve: Uh-huh. So the numerical summary is critical flaws were two, and they were fixed at 3.3 years of age. High vulnerabilities were 34 of them fixed at 6.4 years of age, on average. There were 334 medium concern vulnerabilities fixed at an average age of 5.2 years. And then 186 low-impact vulnerabilities fixed at an average of five years. So he says his comes out to roughly five years lifetime again, so not much has changed from Jon's analysis in 2010.

And he says: "While we're getting better at fixing bugs, we're also adding more bugs." And of course that brings us back to that T-shirt that we talked about a few months back of the 99 bugs on the wall, take one down, whatever it was, fix it, 113 bugs. Or 99 bugs in the code, and then it's 113 bugs in the code because this stuff is complicated. And so often now things we do have unintended consequences.

So he says: "And for many devices that have been built on a given kernel version, there have not been frequent, or sometimes any, security updates, so the bug lifetime for those devices is even longer. To really create a safe kernel, we need to get proactive about self-protection technologies. The systems using a Linux kernel are right now running with security flaws. Those flaws are just not known to the developers yet, but they're likely known to attackers." And we just discussed exactly such a story, where this race condition was discovered from its use in the wild. "But they're likely known to attackers, as there have been prior boasts/gray-market advertisements for at least" - and then he notes a couple common vulnerability numbers.

So anyway, from my standpoint, I mean, we recognize now that very complex software is very difficult to make perfect. And we also know that security is, by its nature, in our current model, and they say it because I don't think it always has to be this way, but our current model, exactly sort of as he implies, is the weakest link, that is, security is a chain. And the strength of the overall chain is limited by the strength of the weakest link. And every single one of them has to be strong. And the way our systems are designed today, unfortunately, that's the nature of it. I mean, so you could argue we're fighting a losing battle. The systems are inherently insecure, that is, their design promotes problems. So it's only by being as perfect as we can about not inducing problems which are sort of natural, that we get security.

So, again, it's an uphill battle. And now, with the IoT revolution and the fact that so many people are using tiny Linux microkernels, pulling from the free open-source codebase in these devices, they're going to have these problems, and they're never going to get fixed. I mean, even older mobile phones with Android are not going to get fixed. They're not going to get patched. And I've heard you talking, Leo, about for example the Pixel and how good it is, that it's a phone you love, and Google is 100% behind it, and we know that it's going to be kept current.

Leo: Well, for a couple of years anyway.

Steve: Yeah.

Leo: I mean, that's the only problem. Even Google doesn't - like the Nexus 6 is already out of support.

Steve: Right.

Leo: Or will be in a couple of weeks.

Steve: And in fact it was the 5 and the 6, it was the 5 specifically, but also the 6, that we'll be talking about shortly, that the Drammer guys found as exploitable. So as soon as those phones are no longer maintained, they become platforms, yeah.

One little bit of errata from last week. I got a kick out of this. Someone shot me a tweet after the podcast on a CNET article whose headline was "Tech luminaries laud Dennis Ritchie five years after death."

Leo: Did they mention you?

Steve: No, no, no. I'm not a luminary. But the subtitle was: "Well-known tech figures appear to have forgotten the father of the C programming language died years ago, falling victim to social media's 'second death syndrome.'" They go on. "Some of tech's biggest names are paying tribute this evening to computing pioneer Dennis Ritchie. Ritchie was an internationally renowned computer scientist who created the C programming language. He also made significant contributions to the development of the

Unix operating system, for which he received the Turing Award in 1983.

"The problem, especially if you look at it from Ritchie's perspective, is that he's been dead for five years - exactly five years. The time gap seems to have escaped some of the biggest names in tech, including Google CEO Sundar Pichai, who late Wednesday tweeted out Wired's five-year-old obituary on Ritchie..."

Leo: That's where it started. So you know what? That's where it started. Everybody copies Sundar, yeah, yeah.

Steve: "...thanking him for his immense contributions." And then: "Om Malik, a partner at True Ventures and the founder of tech site GigaOm," who of course is a wonderful and frequent guest on your TWiT show...

Leo: Yeah, we love Om.

Steve: "...retweeted Pichai's tribute before soon recognizing his mistake and tweeting an apology for 'adding to the confusion and noise.'" Which of course I also did last week. So I thought that was an appropriate [crosstalk].

Leo: You did for moments, and then you corrected yourself. You know, in a way it's fitting because one of the reasons I remember this vividly is because he died the same week Steve Jobs passed away. And Steve Jobs, of course, got all the ink. And I received numerous emails, and there were many tweets from people who said, "How come Steve gets all the credit? He created nothing. And Dennis Ritchie passes away, and no one notices." And, of course, neither is exactly true. But, sure, five years later...

Steve: Well, it's pop star versus, you know, our star.

Leo: Right.

Steve: You know, I mean, I'm waving my K&R book around last week.

Leo: Yeah. Well, I think in a way it's kind of fitting that now all the noise about Steve Jobs has faded away, we can give Dennis the tribute he finally deserves.

Steve: Indeed.

Leo: Mm-hmm. I was wondering where this started, and I bet you it started with Sundar. It had to be somebody so influential that everybody else would say, oh, and retweet it. And I saw those retweets; and I thought, this is weird, because I remembered it. And so that explains it. It's somebody so kind of godlike that no one

would even say, "Well, let's see if that date's right."

Steve: You just don't bother doing a double-take, yeah.

Leo: Right, right.

Steve: Yeah. And in fact, even if you had doubts, you accuse yourself of being wrong.

Leo: Right. Oh, I must have forgotten, or I must be mistaken.

Steve: Yeah. I was thinking about somebody else. And of course we did lose Dave Bunnell in this past week.

Leo: And Beranek of BBN (Bolt, Beranek and Newman).

Steve: Yes, yes.

Leo: Who's an interesting story because he was an acoustic engineer, formed BBN in Boston, and was invited by ARPA to design the early Internet. And BBN did, in fact. And he passed away. He was 102.

Steve: Wow.

Leo: So it shows you. Invent the Internet, live a long time. David Bunnell, sad to say, merely 69, which...

Steve: I know. And do we know anything...

Leo: As we get into our '60s, that sounds too young.

Steve: Hey, I'm there, baby. You know, I'll be crossing in...

Leo: Steve, they'll be talking about us soon enough, believe me. We're getting into that territory now; you know? But Bunnell was, in many ways, I did not know him.

Steve: Instrumental. He was key in the PC industry.

Leo: Yeah. He founded PC Magazine and PC World, arch rivals; MacWorld magazine; and was the inspiration, mentor for many of the people we work with in the tech industry. Harry McCracken wrote a great article about him last week, too.

Steve: You remember PC Mag? It was every two weeks, and it was an inch thick.

Leo: You had to read it. And you had to read it.

Steve: And it was, like, 95% ads. I mean, it was the biggest cash cow the industry has ever seen.

Leo: No, one bigger. Computer Shopper Magazine.

Steve: Oh, okay. Right. That horrible oversize newsprint thing.

Leo: It was huge.

Steve: You had to wash your hands, yeah.

Leo: It was tabloid size. It was four inches thick. I think it did have the merit of only coming out once a month. But I got it, I loved it, also from Ziff-Davis. I got it because that's where, you know, you could peruse the ads. Now, with the Internet, none of these have survived. Nobody prints the stuff anymore.

Steve: I got an award once from David Bunnell.

Leo: Did you. So you knew him.

Steve: Oh, yeah, I knew him. I designed and inserted the highest pulling ad in the history of PC Magazine.

Leo: What?

Steve: And they gave me an award for that, yeah.

Leo: Wow.

Steve: It was a simple page. The headline was "Hard Disks Die."

Leo: Oh, I love it.

Steve: And then the subhead was "Ever Wonder Why?" And it was just a little bit of prose with some cool photos. But it, I mean, it was expensive as all get-out because it was in PC Mag. But it pulled more than any other ad they ever ran.

Leo: Nice.

Steve: Well, because people understood that that was the case back then.

Leo: Yeah, yeah.

Steve: I just did want to mention, just complete random miscellany, but this AT&T/Time Warner merger.

Leo: Wow, huh?

Steve: You know, I'm a free enterprise entrepreneur capitalist. But I recognize that our system has a problem, which is that power begets power. And when companies acquire an advantage, they use it to their own further advantage. I guess, in reading into this a little bit as I have over the past week, there has been some suggestion that, if this is allowed to proceed, then maybe some concessions can be obtained, like unbundling of cable packages and other desirable things. But, boy, I just, you know, here I am stuck with only one viable source of Internet connectivity, as an example of how the system fails the consumer when companies acquire too much domination in a market. And this scares me as much as the, what was it, the Comcast/NBCUniversal acquisition.

Leo: It's very similar, isn't it.

Steve: Yeah, yeah. It is a carrier acquiring content.

Leo: Yeah.

Steve: And it's like, okay, well, let's hope our Network Neutrality legislation finally does come down doing the right thing because this is worrisome. You know, when a carrier has an interest in some content over competing organizations' content, that's just scary.

Leo: Not good.

Steve: Also, a little more on topic, at least relative to this podcast, This Week in Energy Storage. I knew of this last week, but we were too crammed full of stuff. And so I

thought, okay, I'm just going to save this one for this week because this isn't particularly time sensitive. But I just wanted to put it on our listeners' radar. You know, as everyone knows, mobile energy storage is an interest of mine. We've talked about super capacitors and battery technology and, you know, nanotubes and all that. This comes out, this was a press release last week from the very well-known Oak Ridge National Laboratory. They, as often happens, accidentally discovered what they believe is an efficient process to turn - are you sitting down, Leo?

Leo: Yes.

Steve: CO2...

Leo: Oh, I saw this, yeah.

Steve: ...into ethanol.

Leo: Huge. I hope this is not bogus. I hope this is real.

Steve: This sure has all the smell of real.

Leo: Yeah.

Steve: And it uses inexpensive components at room temperature with a little electricity to catalyze a reaction.

Leo: So why is this a big deal, Steve?

Steve: Okay. So as we all know, we believe there's a problem with increasing levels of CO2 in the atmosphere.

Leo: It's not belief anymore. It's demonstrably true.

Steve: I'm just - right. I'm trying to be politically...

Leo: No need to be politically correct. It's demonstrably true.

Steve: So, and we know that that tends to have a heating of the planet effect. There's been a lot of talk about carbon sequestration, that is, trying to trap the CO2. And then they want to, like, pump it back down into the dirt. And it's like, well, okay. But that just seems to be expensive, with no return for the investment other than keeping green people happy.

So what these guys have, and I'll read from the Oak Ridge National Laboratory's official release: "In a new twist to waste-to-fuel technology, scientists at the Department of Energy, the DOE's Oak Ridge National Laboratory, have developed an electrochemical process that uses tiny spikes of carbon and copper" - both abundant - "to turn carbon dioxide, a greenhouse gas, into ethanol. Their finding, which involves nanofabrication and catalysis science, was serendipitous.

"ORNL's Adam Rondinone, lead author of the team's study published in Chemistry Select, said: 'We discovered, somewhat by accident, that this material worked. We were trying to study the first step of a proposed reaction when we realized that the catalyst was doing the entire reaction on its own.' The team used a catalyst made of carbon, copper, and nitrogen and applied voltage to trigger a complicated chemical reaction that essentially reverses the combustion process." I just love that phrase. "With the help of the nanotechnology-based catalyst which contains multiple reaction sites, the solution of carbon dioxide dissolved in water turned into ethanol with a yield of 63 percent. Typically, this type of electrochemical reaction results in a mix of several different products in small amounts." So this is both very efficient and very selective.

Adam continued: "We're taking carbon dioxide, a waste product of combustion, and we're pushing that combustion reaction backwards with very high selectivity into a useful fuel. Ethanol was a surprise," he says. "It's extremely difficult to go straight from carbon dioxide to ethanol with a single catalyst. But this does that."

Leo: Amazing.

Steve: "The system operates by dissolving CO2 in water at room temperature and, with a bit of electricity, produces ethanol."

Leo: Wow.

Steve: So, again, when I see something like this, I think, this is why we absolutely have to keep raw science funded.

Leo: Yeah. You never know.

Steve: Exactly. I mean, we got the Internet from it. And now here's a potential win because this makes it then feasible to just have big CO2 scoops, you know, suck it in. CO2 dissolves easily in water. Run it through this process, out comes gas.

Leo: Wow.

Steve: I mean, hello.

Leo: Hello.

Steve: I hope Elon knows about this. Maybe he can just build this onto one of his next ideas.

Leo: Pretty amazing.

Steve: And finally, I just want to say that last night I hit 24%, because I looked down at the little bar on my Kindle, in Peter Hamilton's "Night Without Stars."

Leo: Oh, okay, okay, okay.

Steve: So, and you know, I should know better by now because he divides this into what he calls "books." And the books have chapters.

Leo: Yeah.

Steve: And so Book One, so I go in, you know, last week and start learning about a whole bunch of new people, and I kind of get up to speed on them. And then I go to Book Two. It's like, okay. And now I've got a whole - I changed location and new people. And it's like, oh, okay.

Leo: I just got up to speed on the first set.

Steve: Exactly. So then I got to learn all these people. And then I switch. Then I get into Book Three. It's like, oh, no. And yet another name I haven't seen before, and a whole new location. I'm thinking, oh, come on. And then, wham. It just took off like a rocket ship.

Leo: Yeah.

Steve: And I remembered having exactly this experience with "Pandora's Star," where it was interesting; but it was like, okay, where are we going with this?

Leo: Right.

Steve: You know, I mean, and because many books are written as sort of in a first-person narrative, where you're just, you looking out of the eyes of the protagonist as he walks around and talks about what's happening, or it's being described or something. Peter doesn't write that way. Peter writes big, complex, swooping, sweeping, you know, stories. I mean, he's a real storyteller. And so he needs to set up the pieces and the people and, you know, establish. And so it took 24% of this book to establish what's going on. And then it just, like, I mean, I kept looking at the clock last night thinking, oh, I've got to get up early to work on the podcast all day today. Anyway, so for what it's worth, oh, boy. You know, we find out what happened with Bienvenido, which is where

we left off on the first half of this duopoly here, duology here, whatever the heck it is. And it is really looking like another fun book.

Leo: Oh, boy. I can't wait.

Steve: Yeah. Twenty-four percent and fasten your seatbelt.

Leo: It's my next book, yeah.

Steve: Oh.

Leo: I have it on Audible, of course. I listen. Yeah, that's awesome.

Steve: Yes, yes. So Colin Wills in Dorking, England, shot me a nice note. And he's a listener, so he knows how to put in the subject line, "SpinRite saves a guitar lesson." And he did a little smiley-face. So he says: "I worked from home today using my bitlockered" - and I kind of like that, "bitlockered," I've been bitlockered - "using my bitlockered Windows 7 work laptop so that I could take my daughter to her guitar lesson. But last night I thought I was going to have to go into the office and that we would have to skip the lesson because my laptop wouldn't boot. It had recently hung at some point during the boot, but had recovered at a second attempt. But last night it repeatedly got stuck. Being a Security Now! listener almost from the start, I immediately thought of SpinRite." And of course that's why I tell everybody about it every week, so that you immediately think about SpinRite.

Leo: No fool, you.

Steve: Instead of, you know, instead of, like, oh, no, what do I do? Who could not know at this point? And he says, parens, "(Actually started around Snowden but also recapped and now just past the Sugar Hill at 353 and reading Taubes.)" He's referring to Gary Taubes's book.

Leo: Yes. "Why We Get Fat."

Steve: He says - exactly. "The work laptop is locked down such that I cannot boot my SpinRite CD, so I transferred the hard drive to my personal laptop and spinrote," he says...

Leo: That's a new verb.

Steve: Spinrote.

Leo: Spinrote. New tense.

Steve: Yes, "...it quickly at Level 2. Of course, the fact it was bitlockered didn't matter, and SpinRite cut through the disk like a knife through full fat butter and completed after about an hour. After that, the work laptop booted, I worked from home, and we made it to the guitar lesson. Thanks for the great product. I am really, really looking forward to v6.1." Colin, not as much as I am, and all of our SpinRite listeners are. So Colin, thank you for sharing the news and helping me further plant the "Aha, I know how to fix this..."

Leo: Aha.

Steve: ...in our listeners' minds.

Leo: Aha. All right, Steve. I sit here poised with my finger over the Install button.

Steve: So what Leo is referring to is the fact that these guys who have revealed this new Rowhammer version of an attack that they named DRAMmer, or Drammer - it can stand for DRAM Hammer, or also Deterministic Rowhammer because that's one of the innovations of their particular implementation of this. They have produced a side-loadable tester that runs in Android devices. They had been expecting Google Play to host it for formal availability, but it has not yet appeared in the Google Play Store.

So I tried it last night. I meant to try it on a Kindle Fire. But I did try it on an older Samsung device. And when I clicked the link and tried to run the .apk file, I got the notice that the device is not configured to allow unauthorized software to run. So, of course, I clicked the settings and went in. And I said, yes, I do want to drop my shields for a moment. So then I allowed that to happen and ran the APK. It doesn't have a fancy UI. You can see it allocating some memory. And then it produces a log if, over time, it manages to induce bit flips in the DRAM of your particular Android device. I'm a trusting person. I trust these guys. So I don't have a problem. It's coming from their site, vvdveen.com/drammer/drammer.apk.

Leo: Oh, man. I'm trusting you on this one because this is against every instinct I have. But you trust them. You know them.

Steve: Yeah, if there was any way to, well, I mean, I'm in a dialogue with them.

Leo: Now, we should say that the tester is not open source, because otherwise they'd be publishing exploit code.

Steve: Correct.

Leo: Right. So should I do it?

Steve: Yes, do it.

Leo: So this is Pixel. This is the new Google phone which, in theory, and it has all patches applied, would be as secure as it could be. Now, note, even after you turn on the "install from third-party sources," which it warns you against, you'll get this further warning: "Installation blocked. Drammer. This app contains code that attempts to bypass Android security protections." I'm encouraged, by the way, that you see stuff like this.

Steve: Yes.

Leo: One would hope that it would see that in other applications. You could say "okay," and it won't install it. But if you say "more details," it'll say, even if you've heard of this app or the app developer, it's still dangerous to install an app from an untrusted source. They warn you three times. Now I'm going to say "install anyway."

Steve: Yeah, because these people are not untrusted. I mean, they are, you know, they're on our side. I mean, they told Google about this.

Leo: I think it installed. Did it install? Let me do it again, just to make sure. More details. Install anyway. There we go. Okay. App installed. Now I'm going to open it. It downloads something. Should I do - "Check whether your device is vulnerable to the Rowhammer bug." Relaxed or aggressive? Should I go all the way?

Steve: Yeah, crank it up.

Leo: In for a penny, in for a pound. And then you can send statistics or not about the test result. Of course I'll help them out. And then I love the button, "Hammer time." Now, you're seeing - I'm seeing it, the log.

Steve: Yeah. And I didn't get it. I didn't get the log on mine.

Leo: Well, it just crashed is what happened.

Steve: And did it say "flipped" in those entries?

Leo: Well, it crashed.

Steve: Okay.

Leo: It's crashing out.

Steve: Okay.

Leo: I don't know if that's a good thing or a bad thing.

Steve: And what I'm wondering is, I see there's a little block that are, like, memory allocations, which I'll explain in a minute.

Leo: Yeah, yeah, yeah, yeah, [crosstalk]. Low memory, boom.

Steve: Okay. Okay.

Leo: Is that good or bad? I don't know.

Steve: Oh, it means nothing. It just means that it's raw code that they put together and that hasn't been widely tested. Although, as I'll explain, they've tested it on a whole lot of devices. So there may be something about the Pixel which is a little bit different. And I imagine it will evolve. And maybe that's why it hasn't appeared in the Google Store. They might be waiting to tweak it a little bit.

Leo: Let me turn down aggressiveness. Would that make it maybe more reliable?

Steve: Oh, that might - how about put it back where it was, just a little, like only half an inch over from the left.

Leo: Right. It crashed again.

Steve: Okay.

Leo: But I'm going to keep turning it down until it doesn't; right? Boom.

Steve: Well, it's probably...

Leo: I don't know if it's actually crashing because I see it in [crosstalk].

Steve: Well, it is deliberately coming up against an out-of-memory condition. And I'll explain why they're doing that.

Leo: Oh, okay.

Steve: And so that's on purpose. But it's a little - oh, now it didn't.

Leo: No, it's just slower.

Steve: Okay.

Leo: It's going to take a while. But I saw the same thing. This is, by the way, all the way relaxed. And it's reading from page zero and page X, X=0 to 128. So, and then eventually it's going to say "exhausted," and then it's going to say "out of memory," and that's when it usually crashes. So what is it doing here?

Steve: Okay. So the background. Our listeners who've been following the podcast for two years will remember that we first discussed what was initially a theoretical "this doesn't sound good" problem. And as so often is the case, things start out as, oh, that's not good, but it seems like it'll be hard to exploit. Well, a few months ago we talked about the Flip Feng Shui. And in fact that was the title for that podcast, where this same group there at the VU University in Amsterdam, led by Professor Herbert Bos - there's a team of researchers that work with him, and a trio from UC Santa Barbara in this case. These are the guys who figured out how a process running in a shared hosted environment could arrange to flip the bit in the public key in an adjacent process, rendering that public key factorable, and then factor it and immediately use that modified public key to securely access, with full privileges, that other process.

So to do that, they had to do some amazing gymnastics, essentially, very clever, where they were able to use the fact that these virtualized environments do page coalescing, or deduplicating, which is coalescing, where any duplicate pages are being looked for constantly by a background thread. And if they're found, the page tables which describe each process's mapping of logical to physical memory, the page tables are modified so that both instances of logical memory in the two processes refer to the same physical memory. And this is a highly effective memory-saving technique that modern virtual environments provide.

So these guys were able to leverage, to take the raw capability of, okay, if we pound on memory, our own memory, which is the only memory we have access to, we can flip a bit in it. And so you first think, well, okay, how's that going to be useful? Well, they turned it into something that was an actual attack. Now they're back. So they call this one Drammer. And as I mentioned, I think of it as DRAM Hammer. But on the other hand, all Rowhammers are DRAM Hammers because this is a fundamental problem with DRAM.

Just to remind our listeners, DRAM is arranged in a grid array. And the nature of the dynamic RAM is such that rows, entire strips, strip rows are refreshed and read at the same time. And it turns out that designers have shrunk the cell size of DRAM, as they're always doing, in the same way they pushed the margins out of hard drives, and then they pushed the margins out of lithium-ion cells, trying to squeeze as much usable surface area as they can at the cost of manufacturing tolerances being reduced. They also squeezed DRAM cells down to where they, oh, look, they still work when they're this small. It's like, uh-huh, unless you create a lot of noise in the neighborhood. And so that's what this does. This creates a noisy neighborhood. And if you have particularly weak cells that are not being refreshed at a high enough rate, you can cause them to misread. You can flip the bits from zero to one or one to zero.

So the question had been, that had never been answered before - because all of the Rowhammer attacks to date have been on x86 and Intel x64 platforms, never on the ARM platform. And of course most mobile devices are ARM-based because it's so efficient with battery power for the computing that you get out of the power consumption. So they decided to tackle the ARM platform. The first problem - oh, and in a realistic environment, not just some raw ARM chip on a board, but an existing smartphone, Android-based phones and tablets, Android-based ARM tablets or ARM devices.

So from the beginning of their abstract they said: "We show that deterministic Rowhammer attacks are feasible on commodity mobile platforms and that they cannot be mitigated by current defenses. Rather than assuming special memory management features, our attack, Drammer, solely relies on the predictable memory reuse patterns of standard physical memory allocators. We implement DRAMmer" - or Drammer, I don't know how they pronounce it - "on Android/ARM, demonstrating the practicability of our attack, but also discuss a generalization of our approach to other Linux-based platforms.

"To support our claims, we present the first Rowhammer-based Android root exploit relying on no software vulnerability, and requiring no user permissions. In addition, we present an analysis of several popular smartphones and find that many of them are susceptible to our Drammer attack. We conclude by discussing potential mitigation strategies and urging our community to address the concrete threat of faulty DRAM chips in widespread commodity platforms."

So the first thing they needed to do was to verify experimentally that the DRAM memory in existing ARM-based smartphones could be subject to Rowhammer. There was some suggestion in the researcher community that DRAM required faster repeat writes to DRAM than the memory controllers in these devices might support. So it might be that, while the DRAM itself could be faulty, the DRAM controller wouldn't allow a high enough bandwidth writing to inflict the Rowhammer fault on the underlying hardware. So they first had to determine whether that was possible or not.

Now, in order to pound on DRAM, you need to get the cache out of the way. We've talked about caching, how for example in modern Intel architectures you have a hierarchy of caches - the L1, the L2, and the L3 cache. And so any attempt to write is typically written through to the DRAM. But if you write it again, the cache realizes you're writing the same thing you just wrote, so it discards it. And so that write doesn't get propagated down into the hardware because the cache is already synchronized with what's in the hardware, and so it saves that write cycle, which is also a benefit for performance in a busy system.

So in order to bypass the cache on the Intel platforms, there is a cache flush instruction which is available from userland, from Ring 3, from where normal user code executes. And remember also that, in the intervening time, Google found that there was a Rowhammer vulnerability that they were able to inflict in Chrome because their NACL, their native code execution, had not ruled out the use of the Intel cache flush. Well, they simply changed that native code interpreter in order to make that no longer possible. So that solved the problem. That is, if you can't flush the cache, you can't do Rowhammer because the cache will get in the way. In a very similar way to the DDoS attacks we were talking about at the top of the podcast.

So the first problem was that, whereas the cache flush is a non-privileged instruction on the Intel platform, it is privileged on the ARM. So this is, like, whoops, wait a minute. That means that an application running in user space has no ability to flush the cache through the instruction that is present in the ARM architecture, but only available to privileged code. So the first thing they did was they said, okay. Let's just not worry about doing this from userland. Let's first figure out whether the DRAM controller can permit

this. So they wrote a kernel loadable module and their own kernel code just for the moment, to be able to have access to the privileged cache flush instruction. They pounded on DRAM, and they got bits to flip all over the place. So they said, okay. If we can come up with a way of doing Rowhammer without the cache flush instruction, which is not available to us from userland, then we've got something to work from.

Turns out, again, these guys, I don't think anything could stop them. As I said, I am so glad they're on our side because it turns out that there is a way to pound on memory without having to flush the cache. And that's from direct memory access. DMA inherently avoids the cache because you need to keep the view of different DMA'ing processes coherent, meaning that you can't have any situation where the cache might contain data which has been modified, but not yet written to DRAM.

So DMA, Direct Memory Access, is a cache-bypassing process. So they were able to come up - oh, and DMA is available to userland processes, all kinds of things, things doing video and webcam apps and even audio. I mean, all kinds of things need high bandwidth access to main memory for reading and writing. So that's something reliably available, certainly on the Android platform, and probably everywhere. So they explained that they consider Drammer to be a particular instance of the larger Flip Feng Shui technology. So for Flip Feng Shui in general to be successful, three requirements have to be met. An attacker needs to be able to hammer sufficiently hard one way or the other, hitting the memory chips with high frequency, because no bit flips will occur if the memory controller is too slow. And now they know that they're able to do that.

Also, physical memory needs to be manipulatable and manipulated so that exploitable data can be located in a vulnerable physical page. And we saw them do that in a different instance using this memory deduplication on a VM platform. They don't have that environment in Android, but they needed some way to do that. And physical memory mapping must be determinable to allow what's known as double-sided row hammering, that is, where you literally, when you hammer a row, you are writing to the row of cells adjacent to another row. Double-sided row hammering typically ping-pongs between the row above and the row below the target hammered row, really just hammering the crap out of it. So, but in order to be able to do that, you have to know how the actual bits on the chip are organized. And they designed a heuristic that allowed them to figure out on any given platform the row length in bits so that they were able to relate logical addresses to actual physical rows on the grid of DRAM. Like I said, you know, you want these guys on your side.

So none of those three requirements were initially met with any of the previous research that had been done on Rowhammer. They had to fulfill those from scratch in order to move this over into the ARM platform and make it workable. So with all of this background, here's what they do. They allocate a single physically contiguous block of memory. Now, the good news is allocating memory for DMA does that because DMA solves two problems for them. Not only does it bypass cache, but it also allows for the explicit allocation of physically contiguous memory because you need physically contiguous memory because the DMA is writing to the physical memory, which the process sees a logical view of through the memory mapper that exists in the system, the memory management system.

So it's necessary to have the logical to physical mapping and physically contiguous DMA pages because the DMA just works with a physical counter. And when it's streaming data in or out of memory, it just does that by incrementing the counter byte by byte, or block by block, as it's writing, and just going for however long you tell that the buffer is. So they allocate a big contiguous block of memory. Then they deallocate it and immediately reallocate a ton of tiny blocks of memory. So, and this is the deterministic aspect of what

they've done. So because we know how Android manages memory, and how its memory allocator works - I mean, it's open source. It's public domain. You can probe it. And so that's a given.

So they allocate a single huge block of memory. Then they release it and immediately basic request all of that memory again in tiny blocks. And what happens is the way Android manages memory is it uses page tables which will be located out in the physical space in between blocks of pages that the page tables describe. So it's a way of essentially forcing a layout of memory which is deterministic and well known and gives them the property that they want, which is these page description tables exist at essentially knowable periodicity within this large region.

Now, here's the kick. And the trick is the page tables are exclusively the property of the kernel. They are kernel memory management structures which are absolutely off limits to any user code because the users just see an abstraction of memory through the memory manager. The OS is managing the actual physical allocation. But you absolutely cannot allow a process to mess with it own memory management because that would be giving it the keys to the kingdom. That's something that only the privileged OS is able to do.

So what these guys figured out a way to do was, after templating, they allocate this big block of DRAM. Then they hammer the crap out of it in order to find all the flippable bits within the region of memory that they have access to. And they call that "templating." They template the memory. And by the way, the templating code is up on GitHub in public domain for people to experiment with, if anyone's interested. So they template the memory. Then, knowing how the memory is laid out, they are able to use the Rowhammer on - oh.

So they template the memory, fragment it, deliberately fragment it, which is going to sprinkle page table management structures throughout that fragmented region, which they technically no longer have access to because that's OS. Those are OS-owned blocks which they can't see. But because they know where it must be, they have been able to arrange for those management blocks to fall in areas with soft flippable bits. So then they hammer, as they have already proven they can, on a row adjacent to a row of memory that is in one of these page tables, which flips the bit and maps the page table into user memory.

Once again, they flip a bit - now, again, because this is a hardware flaw, the operating system just thinks they're a strange app. You know, it's writing a lot to this couple of locations. But who cares? Well, that induces an unseen bit flip in the hardware, in the memory management system. And they've engineered that bit flip so that the page table then appears in their user memory.

Now, they have just been given something no application should ever have, that is, an OS memory management structure that they have full read and write access to because they tricked the hardware and mapped that into their own process memory space. So that means they can point entries in the page table which they now control to any physical memory in the system, including the kernel memory. So they do that to scan through the kernel memory, looking for the well-known process management structure, the process credentials, where the UID and the GID and other common Linux structures are stored. They find the structure for their process, that is, the structure in the OS which is managing the rights and privileges of their process, and they simply give themselves root access. They manipulate their own credential structure to grant themselves root access. And they now have full access to the entire system.

So they start as an application like any other that needs zero permission; and, simply by

manipulating a weakness in the hardware, they get control of critical kernel structures, give themselves root access, and then they have the keys to the kingdom. And the bad news is there's no clear solution for this. On their page they show a range of existing current generation smartphones which are vulnerable, how many have been tested, how many of those have found flippable bits. Most have; a few haven't.

Google has known about this for a month. The problem is there's no clear solution. When we were talking about mitigating this back in the Flip Feng Shui episode, we talked about the next generation of DDR memory, which would notice that access had been particularly heavy in a given region and increase the refresh of any rows that might be susceptible to Rowhammer attacks, sort of a dynamic refresh. Another approach is just to crank up the refresh rate. That, as we know it, does take a hit on performance, and it does consume more power. And mobile devices are, if nothing else, very sensitive to power consumption.

So basically they have demonstrated that a two-year-old theoretical flaw does affect a wide number of existing ARM-based mobile devices; that unprivileged software can give itself root access, and it takes about 30 seconds overall to do this; and it's not clear how this thing gets mitigated. I mean, the problem is it's not a flaw in any code. It's a flaw in the hardware that we are trusting not to have its bits flipped. And they've shown they can. Beautiful piece of work.

Leo: Now, I reran the thing with the relaxed mode all the way up, and it did complete, but it didn't give me any report. So I don't know what to think about Drammer and the Pixel.

Steve: Okay, yeah.

Leo: I mean, it passed? It doesn't mean anything.

Steve: Yeah. The fact that - it shouldn't have crashed. Mine didn't crash.

Leo: Sounds like it's safe.

Steve: Yeah. And they don't have a big UI on it yet. They are presenting this toward the end of this week, which is why it came out of embargo, and the world knows about it. Dan Goodin wrote a great piece in Ars Technica. And I started getting, as I knew I would, tweets about it yesterday when the news broke. So we'll keep an eye on it. I don't know how soon Google's going to have patches out. And it's not clear how they're going to mitigate this.

Leo: Yeah. Apparently the 5X is not vulnerable. So I wonder, is DDR4 vulnerable? Does that make it invulnerable because it's DDR4? Or does it matter?

Steve: Both are.

Leo: Three and four? Yeah.

Steve: Yeah, they did find some DDR4 that is vulnerable.

Leo: This is DDR4.

Steve: Yeah. And there is a next-generation DDR4 which does incorporate adaptive refresh. And so that's probably what's going to happen. But the problem is look at all the hardware that's already out there.

Leo: Right.

Steve: I mean, you know, this is...

Leo: Unpatchable. I mean, unfixable.

Steve: It's not clear how you fix this.

Leo: Right.

Steve: Yeah.

Leo: Well, it looks like, what I would guess from the warning I got is that Google at least is scanning for software that tries to do stuff like this, or tries to do something, anyway. Because I did get this "unsafe software." You have to override this to - not merely that it's untrusted source, but that it's literally unsafe. So that's encouraging. Do you get that on your 5X?

Steve: No, I didn't.

Leo: Yeah. I wonder if that's something new. Now, you said your 5X was vulnerable; right?

Steve: No. In fact, it was a Samsung. It's an old Samsung tablet.

Leo: Oh, you did it on an old Samsung.

Steve: It's an old Samsung tablet.

Leo: They said 12 of the 15 5's were vulnerable. I don't know. It's unclear. They only tried one 5X. So more to come, obviously.

Steve: Well, yes. And they refer to their Nexus 5 as, like, their old beat-up war horse.

Leo: Right, right.

Steve: I think, you know, this is the one that they subject to everything. And so one question they had was whether this might be usage fatigue. You know, might there be a more prone tendency for bits to flip on, like, well-hammered DRAMs?

Leo: It's worned out. It's all worned out.

Steve: Although there is an electrochemical process that we know of to describe that. You know, for example, we know exactly why flash RAM fatigues and dies over time. We know why we need wear leveling and to limit the number of writes for flash. DRAM doesn't have that characteristic. So I suspect it isn't usage base. But it certainly could be also the case that even the same model phone in batches is made with slightly and subtly different DRAM. I mean, because they just say we need 4GB of DRAM. And they buy some batches from this supplier, and some batches from that supplier. So even things like little process-to-process variations, I mean, because we're down at the margins. But as this demonstrates, margins matter.

Leo: Yeah. All right, ladies and gentlemen. There you have it, the scoop of the century.

Steve: So for our listeners who probably, I think without fail, want to run this DRAM test, if you just google Drammer, D-R-A-M-M-E-R, you'll find the article and the page. It's in the show notes this week. And it's from these guys. I trust them. And all it's doing is seeing whether it's - it doesn't do any exploiting. It just looks for flippable bits.

Leo: Yeah.

Steve: And in fact, if somebody finds them, send me a tweet. I'd just love to know that some listeners found flippable bits in their phones.

Leo: And of course I immediately uninstalled it and rebooted, just in case.

Steve: Yeah, good.

Leo: Just in case. And I'm deleting the APK right now. Well, thank you, Steve. I

don't know. My results were inconclusive on the Pixel. But I'm sure we'll learn more in time.

We do this show every Tuesday at 1:30 Pacific, 4:30 Eastern, 20:30 UTC, if you want to watch live at TWiT.tv and be in the chatroom at irc.twit.tv. But if you can't, don't worry. On-demand audio and video also available at Steve's site, GRC.com. Actually, he does audio. And he has something unique there. He has transcripts. You'll also find, of course, SpinRite, the world's best hard drive maintenance and recovery utility. You'll find many freebies. Steve's very good on that. ShieldsUP! might be a useful thing if you have an IoT device running. I'm sorry, no, no, ShieldsUP! would be for the COW, the Dirty COW. Test your network for a Dirty COW.

Steve: I can't believe they called it that. Oh.

Leo: And of course everything else. It's a great site: GRC.com. But do come to our site, TWiT.tv/sn. If you want to subscribe or download video, we have that. And any podcast app will have the latest version. Subscribe. That way you get them all. Complete your set, all 800 and, what is it, no, 583 versions of this show. We're going to keep doing it till we get it right. Or till the Internet is safe, whichever comes first. No, that's going to be a long time, Steve. Thanks so much.

Steve: Yes, we're not going to run out of anything to talk about. We'll do a Q&A next week unless all of the world's toasters decide to gang up.

Leo: Revolt.

Steve: And attack the Internet and bring the world to its knees.

Leo: By the way, you can ask Steve questions: GRC.com/feedback. But he's also on Twitter @SGgrc, and you can DM him there and ask a question there. He uses Twitter questions, as well.

Steve: And we will see you next week. Thanks, Leo.