iOS Security Part 2

Description: (Part 2 of 2) On the heels of Apple's major update to their iOS Security whitepaper, Steve and Leo catch up with the week's top security news, including coverage of the interesting discoveries from the past week's 14th annual CanSecWest and Pwn2Own hacking competitions. Then, having come up for breath after last week's Part 1 episode, they take a second deep dive into everything we have learned about the inner workings of iOS. Most is good news, but there's one bit that's VERY troubling.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-447.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-447-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is back, the latest security news coming up, including results from the Pwn2Own contest. And then we'll get back to iOS security. Steve's very impressed. More about how the iPhone 5s keeps you safe, next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 447, recorded Tuesday, March 18th, 2014: iOS Security Part 2.

It's time for Security Now!, the show that protects you and your loved ones online; your privacy, too. And here he is, the Explainer in Chief, Steve Gibson. Hi, Steve. Good to see you.

Steve Gibson: Hey.

Leo: I didn't say anything last week because I thought maybe he just didn't shave.

Steve: Well, there have been scruffy times. And with an HD cam you really can't away from scruffy any longer. But, yeah, this is an experiment that was triggered by Harrison Ford coming out onstage during the Academy Awards. And I thought, huh. He was sort of unshaved chin. And I thought, well, that's kind of the coloration I would have. Now, I'm not comparing myself to Harrison by any means, and I don't think anybody else will make that same mistake. But still...

Leo: Well, you're a damn fine-looking man. And I think that...

Steve: Too bad we're 500 miles apart, Leo.

Leo: I almost thought that you were going to do - I thought it was going to be a goatee. But I've been informed - by the way, here's Harrison Ford with his look. I think, put on some sunglasses, age about 10 years, you could be.

Steve: And get some hair on my head. That'd be good.

Leo: Yeah. Oh, he does have quite a bit of hair up top, doesn't he.

Steve: He's still with Calista. That's cool.

Leo: Yeah, Calista Flockhart, his wife, yeah. And his son Shaquille O'Neal. So...

Steve: And was he married before? I don't know the story of Harrison.

Leo: I don't follow that. This is - I'm not Perez Hilton.

Steve: Neither of us.

Leo: I dimly remember that before Star Wars he was working as a carpenter. But so I imagine he probably had - they call it the...

Steve: I think I remember that, too.

Leo: Yeah, the original wife, and then the Hollywood wife.

Steve: And he was really fit as a consequence because he was out in the sun and hammering and sawing things.

Leo: But that's not what we're here to talk about.

Steve: No, we're actually going to do a podcast, believe it or not.

Leo: Yeah, we're going to continue. It's Part 2.

Steve: Yes. As I thought, we did not have a chance to finish Part 1 last week. And there will be nominally even a Part 3 because one thing I want to talk about, which these first two parts actually set us up for really well, and it's a topic we have never addressed, and that is jailbreaking. If all of this is so good, then what is jailbreaking? And it's not a topic in the nine-plus years we've been doing this that we've ever come across. So I think we will pause for a Q&A because I'm sure there'll be Q&A from these last two podcasts. And then I'm planning to talk about jailbreaking in 449. But for now...

Leo: Oh, I like that. We had a guy on here last week when we were talking about security who was saying, I'm a jailbreaker, I do jailbreak software, and was adamant it did not compromise security. So I'm very curious about your opinion on that. Apple, of course, says it does. But they have other reasons for not wanting you to jailbreak.

Steve: Yeah, true, because it breaks their control. But I would say even that, I mean, we've seen that, were it not for iOS security just being as soup-to-nuts as it has been designed, we know there's tremendous pressure on getting in there. So that's what's going to happen.

So anyway, this week we did have the 14th annual, we've never failed to cover this because it always produces some interesting security news, the 14th Annual CanSecWest security conference up in Vancouver, which has the famous Pwn2Own contest. And out of that came some not surprising Pwn2Owns, but also a disturbing revelation that exactly fits with this podcast because - and many people were tweeting me about this, worried and wanting to know what it meant because it talked about - it was a presentation disclosing the so-called "early random" pseudorandom number generator, newly devised for iOS v7, which is unfortunately a little more pseudo than we would like. So we'll talk about that.

Also just a little blurb about how cloud storage costs have just collapsed, thanks to a recent announcement by Google and what they're going to be charging. An interesting 10-point plan to thwart NSA surveillance. A little update on SQRL. And then we'll plow into the second half of our iOS security analysis.

Leo: Wow. A jam-packed day, but we can do it.

Steve: It occurred to me that one of the reasons why a VPN would be advised, or it's a nice feature that they accept bitcoin, is anonymity. You don't want to necessarily give a provider who you're going to be using to obscure your identity, your identity right upfront. I'm sure they would be responsible with it. And I know that their terms of service are. But just in terms of prudence, if you're wanting to establish an anonymous relationship, then you want to do anonymous payment. And bitcoin is the way that happens now.

Leo: Yeah. Of course, I mean, they have logs, which they destroy regularly. But if somebody, if law enforcement got to them soon enough before the logs were erased, and they knew who you are, then I guess that law enforcement could figure that out, unless you're using an anonymizing payment system like bitcoin.

Steve: Yeah, a buddy of mine did get a letter from his ISP saying, uh, we've been informed that you downloaded something or other, some movie. And he was upset that I hadn't protected him. I said, okay, we'll talk.

Leo: Set him up.

Steve: Yup. So we're going to be talking for some time, and I think this is an entirely valid subject for the Security Now! podcast, about the end of Windows XP. Dvorak, our great friend John, had an interesting PC Mag column titled "The End of Windows XP." And his premise is that Microsoft is making a huge mistake; that killing Windows XP is wasting billions of dollars that Microsoft could be earning. He notes that, and this is a well-known statistic, that about 500 million Windows users are still using XP. Half a billion people are using XP. So although 7 has finally become the majority Windows platform, I think XP peaked at about 800 million at its, well, I just said it, at its peak. And it's only dropped off from 800 to about 500 million. So it is still a significant platform. That's 29%, says John, of the computers in the world. So nearly one third of the computers in the world are on XP. And we've got 20 days to go until April 8th. And this isn't going to change in that period of time.

So anyway, John's point is that many do not want to upgrade to anything new. As he says, they are happy campers. And he wrote in this PC Mag article, he said: "Upgrading the Microsoft OS is a needless exercise in agony." Of course, I should say he's assuming that security patches continue, which is a different issue than what we'll be largely talking about. But he says: "I'd [speaking of himself] still be using an XP machine for my podcasting if the machine itself had not crapped out. What's the point of changing? For prettier icons?"

And so anyway, thinking about it, John's clearly right. Because I'm sort of in the middle of this and am taking a somewhat contrarian view to this idea that, oh, my god, the sky is falling. I'm seeing the incredible anxiety that is being created in the marketplace by the end of this drip-drip-drip IV update feed that we've all been on for 12 years with XP. And what that means is there's money to be made. So what John proposes is, if Microsoft just sold updates for a dollar a month in some fashion, that would be $12 a year times 500 million, potentially, I mean, a lot of these people aren't going to be paying Microsoft. They're in dark corners of the globe. But still, a chunk of money. And we know that Microsoft is still producing the XP patches because they can be purchased under corporate agreement under "paid extended service." So, wow. I guess, I mean, clearly there's a computation that Microsoft is making, figuring this will force people to pay a couple hundred dollars to move to Windows 7. And that's got to be Microsoft's goal. And we'll be covering this for months because there will be stories of various sorts. There's even a hysterical one that we'll talk about.

But I put together what I call "Five Steps for XP Usage Past April 8th." So the first is, now, I did in my notes put run as a standard user. Someone tweeted and reminded me that that term was developed after XP. "Limited user" is the term under XP. So run as a limited user. Remember, 100% of the IE exploits, which is the main way malware gets in, were blocked during all of 2013, if you just weren't an admin user. 100% of them. And what is it, in the 90s, of like the other...

Leo: It's 92% for operating system exploits, yeah.

Steve: So that provides a huge amount of protection. Of course, remove Flash and Java. If you haven't done it before, do it now. But I'll also note that, remember, it's only the operating system that's going to stop getting its IV feed. All the other things - Flash, Java, Chrome, Firefox - they all continue to get updated. So, and those are your contact with the external environment. And those companies, Mozilla's going to stay on the ball. You'll be able to run Firefox v212 on XP, and Chrome and so forth. So all the other pieces continue moving forward. And if you do look back over the exploits, primarily they've been browser, or they've been Java. So there truly is a lot that the user can do to protect themselves.

So No. 2 is remove Flash and Java. No. 3 is use Chrome or Firefox, never IE. And XP hasn't been able to run those later versions of IE. I think XP stops at 9, which is - I don't use IE. I use, as everyone knows, Firefox for my browser. But I've got IE9, and I can't go any further because it's not available for XP. Again, that's another little pressure point that Microsoft puts on us. Also, Office exploits are a problem. I would drop Office and switch over to one of the open offices, and everyone says LibreOffice is the one. Apparently it forked off of OpenOffice some time ago. And I guess Sun or Oracle got bored and decided, okay, fine, we're going to let go of it. So LibreOffice is the one you want. And of course keep that current. And finally, behave yourself online. Don't do dangerous things.

So we'll certainly be following this as we go. And so the danger that I've already seen in tweets with my saying the world doesn't come to an end when the IV gets cut off is I've been responding to people who are our podcast listeners, who aren't trying to use their XP machine in open WiFi in order to surf to dark places which they're not willing to do at home, for example. That's not safe because that's where the next month's zero-day exploits will begin to appear. And with this huge install base of XP, as you mentioned last week, completely correctly, Leo, there will be bad guys looking to leverage what is learned about the platform still being updated, but which are no longer available for XP.

So, and what we do see is many of these problems affect the entire, like IE, all versions that are supported back to 9, which would be the one that you got on XP, which is why you don't want to use IE - not that you really ever did - in a safe way on a Windows XP machine. So an example of the kind of article that, you know, I look at it, and I just sort of shake my head. This was in a regional newspaper, AZCentral.com. The headline was "95% of ATMs could face hacking threat." And quoting from this article, it said: "Banks and other businesses have less than a month to get their plans in place before the computer operating system powering about 95% of the country's ATMs becomes vulnerable to hackers and computer viruses." Nothing could be further from the truth. I mean, absolutely baloney.

Then they quote Ken Colburn, president of Data Doctors, who says: "If you're still running your equipment on Windows XP, you're open to a lot of new threats. All of the devices out there still running Windows XP have to get patched up and changed fairly quickly, or they're going to be exposed to hacks." Colburn continued, saying: "There's still a large number of people out there that just don't realize how big a security threat this is." And I'm one of them. He said: "After April 8, these hackers can come knocking, and you're going to be defenseless." And, I mean, this is this strange notion that somehow, without the IV feed into our OS, it crumbles. It starts to decay. It starts to die in some way. Which is just not the case.

And finally, Scott Kinka, chief technology officer of Evolve IP, was quoted in the same article, saying last Tuesday, in an article for ATM Marketplace, which is obviously a vertical marketing rag for ATM purchasers, the article was titled "One Month From Today: XP Armageddon. Surprisingly, only 15% of financial institutions are expected to react

before the April 8 cutoff, according to a recent ATM industry association survey." Okay, and that doesn't surprise me. Why? You couldn't have an example of a safer use of an operating system, any operating system, than in an ATM. It's a classic embedded application. Nobody is surfing the web. Nobody with the ATM is going to hostile websites. And again, nothing is a hundred percent. But an XP installation being used in an ATM, first of all, it's probably actually the XP Embedded version of XP.

Leo: Actually it's not.

Steve: Really.

Leo: Yeah. I talked with a guy who just retired from Diebold, who said it is Windows XP for Embedded Systems. That's not the same as Embedded XP. It's the same bits as Windows XP.

Steve: Okay. So...

Leo: And Diebold was very, very, very concerned about this, looked into the idea of a volume license. Because if you're a volume licensee you can pay for extended support. So their determination was most big banks are going to be considered volume licensees, and they're going to pay for extended support. They will get support. But they were worried about small banks. And Diebold was considering perhaps licensing it, then reoffering those updates to the smaller banks. And I'm not sure what they decided to do.

Steve: My take is that it's a little bit like having to take everybody's shoes off while we board planes now. That is to say, if a company was still using XP, didn't have the security license from Microsoft, and had a major problem, then they'd be in trouble, like somebody's shoes exploding a second time if we weren't all having our shoes checked. My point is, I don't really believe there's any danger. I absolutely don't. There is no rational way that an embedded operating system with a screen and a keyboard is somehow suddenly going to be prone to attack in two months if it isn't today. I will happily be very surprised if I'm proven wrong, but you guys deserve to know what I think. That's what I think. I just - I don't see it. But I can, from a political standpoint, see the need to appear to be doing everything you can for your end-user security, even if it's complete nonsense.

Leo: There you have it. You and I disagree a little bit. Not completely, but in terms of degree. I think we'll find out in a month.

Steve: Yes. I'm just going from the technology. It's not like the OS, I mean, it is incredibly mature. It's a rock-solid OS. And it's only the exposure to...

Leo: Wait a minute. Steve, it is not rock-solid OS. That's absurd.

Steve: Sure. Well, it's...

Leo: It's not a rock-solid OS.

Steve: Well, okay, then there is none.

Leo: Well, maybe not. But, yes, that's probably the case. It may be more rock-solid than some, but it's not rock-solid.

Steve: Vista, Windows 7...

Leo: There's exploits every single month. They're critical exploits.

Steve: For all of them. For every operating system.

Leo: Well, yeah. But if they're no longer being patched, then it no longer is rock-solid. Am I wrong? They're rock-solid-er if they're getting patched.

Steve: It's not the operating system that is the problem, and it's the operating system that is not going to get patched.

Leo: You're saying it's the user that's the problem.

Steve: The user's a big problem, and the way these things get in. Clicking on links in email. Clicking on links on web pages. Malicious ads being served. None of these things apply in an embedded operating system like you have in an ATM. Anyway, yes.

Leo: Or in a Target point-of-sale system.

Steve: We will have an interesting adventure for the next few months.

Leo: Yeah, we'll find out.

Steve: We're going to learn a lot one way or the other. I'm happy to go on record. So I'm on record. Let's see how it turns out.

Leo: Okay.

Steve: So, speaking of rock solid, at the recent CanSecWest 2014 annual security

conference held in Vancouver last week, starting just after our last podcast, every major browser fell to the Pwn2Own hacking contest, every one of them. There were zero-day exploits found against Chrome, IE, Safari, Firefox, and, additionally, Flash Player and Adobe Reader. There were three successful attacks against Firefox on the first day, and then another one on the second day. A French team from Vupen hacked Google Chrome, exploiting a use-after-free vulnerability that affects both the WebKit and the Blink rendering engines. The researchers then bypassed Chrome sandbox protection to execute successfully arbitrary code on the underlying system.

Another anonymous researcher presented a Chrome remote code execution exploit the next day, on Thursday, but the contest judges only declared it a partial win because some details of the hack were similar to those of a exploit that was presented earlier at Google's own hacking contest, which runs in parallel to Pwn2Own during the CanSecWest conference. And another researcher with the Chinese team Keen combined a heap overflow vulnerability along with a sandbox bypass to achieve remote code execution in Safari. And then he and a fellow researcher demonstrated a remote code execution exploit in Adobe Flash Player.

So basically all of these pieces were demonstrated to have problems which were at the time unknown to the software's publishers. Remote code execution exploits were demonstrated, and of course these will all be fixed immediately.

Leo: This, by the way, supports the XP Armageddon thesis next month. The reason that these have not been revealed is that all of these security researchers hold onto them until Pwn2Own.

Steve: Yes. Yes.

Leo: And it just confirms the fact that people find exploits and hold onto them for an opportune moment.

Steve: Right. So XP, using IE, at Starbucks, on open WiFi...

Leo: Bad, bad idea.

Steve: Surfing porn sites and downloading...

Leo: You know what, by the way, it ain't porn sites. It could be my site. Sites are infected routinely with malware all the time.

Steve: That is true.

Leo: And it's a misstatement to say you're on the dark side of the 'Net. It could be a church site. In fact, it often is. It's just a website that isn't properly secured. Those are the vectors.

Steve: Correct. And even, as we know, even ads.

Leo: Yeah.

Steve: They have JavaScript-containing ads...

Leo: Yeah. It's Yahoo!. It's Yahoo!

Steve: Ads get passed.

Leo: So you don't have to be doing deep and dark dirty deeds. You can just be doing normal stuff.

Steve: Yeah. So we set a record at CanSecWest this year. The prizes won, because they pay heavily for these, the prizes won during the second and final day of the competition put the total contest payout at a record - sitting down? - $850,000.

Leo: And now you know why they hold onto those exploits.

Steve: Yes, yes. And in fact it was noted that Firefox was the most attacked and exploited browser at this year's Pwn2Own with those four new vulnerabilities. And so someone noted that: "Though IE, Chrome, and Safari were all attacked, and all were exploited, no single web browser was exploited at this year's Pwn2Own hacking challenge as much as Mozilla's Firefox. A fully patched version of Firefox was exploited four different times by attackers, each revealing a new zero-day vulnerability in the open source web browser. When asked why Mozilla was attacked so much this year, Sid Stamm, who's their senior engineering manager of security and privacy, responded: 'Pwn2Own offers a very large financial incentive to researchers to expose vulnerabilities, and that may have contributed in part to the researchers' decision to wait until now to share their work and help to protect Firefox users.' The Pwn2Own event pays researchers $50,000 for each Firefox vulnerability, whereas Mozilla now pays a researcher only $3,000 per vulnerability."

Leo: Better get to work.

Steve: So if you've got one...

Leo: I'd keep it.

Steve: Are you going to tell Mozilla, or are you going to sign up for next year's...

Leo: That kind of makes me mad.

Steve: I know.

Leo: Because Pwn2Own is setting up a situation where companies have to overpay for exploit - to reveal exploits.

Steve: And Mozilla is an open source, please send us money, crowd-funded, great group. But they can't afford to pay $50,000 per zero-day vulnerability. And the other thing this does is remember that the point is that we know, because actually from a lot of Brian Krebs' great reporting in the past, that zero-day vulnerabilities affecting Windows XP are going to be selling for a premium, for a good price, because they will be ways for people to get malware into this half a billion XP machines that are being used insecurely after the patches for those are no longer available.

Leo: Where does the money that Pwn2Own pays come from? That's a lot of money.

Steve: It is. I know, $850,000.

Leo: I mean, where are they getting that money?

Steve: You have to wonder, too - what was I going to say? Oh, they're like, they don't know how many zero-day remote-code execution vulnerabilities are going to happen. So, you know, yikes.

Leo: Well, now you can pretty much assume they're giving away all the money.

Steve: Yeah.

Leo: From now on. All the money. Wow.

Steve: So the other thing that was revealed during this CanSecWest conference, a paper was delivered by some neat iOS security researchers. They did a prior paper about iOS 6, where they're really kernel-hardening, kernel-attack guys. And what made the headlines - and again, the headlines were overblown. It's funny, I was just reading Matt, Matthew, I'm forgetting his last name.

Leo: Honan?

Steve: Not Honan. Our security researcher in Chicago.

Leo: I'm drawing a blank.

Steve: Last name begins with "B," I think? I can't get it. Anyway, he was saying the same thing. I was catching - oh, I know. I was looking through his blog, looking for what I haven't been able...

Leo: Green.

Steve: Green. Oh, yeah, Green, right, Matt Green. Thank you. Looking through his...

Leo: Thank the chat room, yeah.

Steve: ...through his blog, trying to find a reference to the TrueCrypt first phase of audit stuff, but still haven't been able to find anything, so I don't know where that reference was. And he was saying the same thing, that he looks at the headlines of the stories covering anything to do with security. And they, I mean, the headlines, I guess they're trying to draw eyeballs. They want to have links that draw clicks from users. So they're just over-the-top headlines. And that's been the case with this. I mean, which is not to say - and we're about to describe it in detail, so I don't want to minimize that this is bad. This is not good, what happened in iOS 7, but it's also not the end of the world.

So here's the deal. We've talked mostly in the context of Windows, back when Microsoft was finally getting serious about hardening Windows. This is when Vista was new, and we were doing the podcast, and we were talking about many of the mitigations. And this is the key word, to understand that these are attack mitigations. For example, ASLR, Address Space Layout Randomization. The idea of that is that, if you simply load the operating system, just like at the bottom of memory to give you sort of a good visual, you just sort of stack it, all the pieces, always in the same way, always in the same place. Then, if a hacker can somehow break through the security barrier between being in the user land or user mode and the kernel, then they get a huge boost from their knowledge of where everything is.

For example, oftentimes an exploit doesn't give them much leeway. For example, they can write a few bytes onto the stack, which when the routine returns, it executes those, but they only could, like, write seven or eight bytes. So that's enough to, like, to jump anywhere in the system. And so they find some code in the kernel that does what they want. Like it grants administrative privileges to the calling process. And so, if they know where that code is, and they can just somehow wangle a jump to there, then suddenly they can get a privilege elevation and turn their process that was in restricted privileges into full admin privileges, that kind of thing. So the point is that, if instead the operating system deliberately scrambles itself up while it's booting, it arranges to never put itself twice in the same order, then that's a tremendous mitigation against the attack, against exploiting the attack.

So I want to separate these two concepts because this is important, just in general in terms of security theory, but for this conversation. The vulnerability is the ability to overrun the stack and write some data. But then the question is what can you do with that? So that's really the bug. Then the question is how do you exploit it? And so mitigation is the second part. It's obfuscating things and doing whatever you can in

terms of operating system architecture so that, if something can gain a little foothold, it isn't able to roll that into a big exploit. So that's the important thing.

So contemporary operating systems do this because the architects recognize we're going to do everything we possibly can to not have any mistakes, to not have anything that would give any unprivileged code access to the kernel. But if we fail in that - and, for example, we just saw an example during the same security conference of failure in that. If we fail in that, we want to make it absolutely difficult for that mistake to get rolled into something really big, a complete collapse of the security model of the operating system or whatever. So how do we do that?

What iOS 6 had is a number of these mitigations, randomizing the logical to physical map. If you have a 64-bit processor, which now we do, you certainly don't have 64 bits' worth of RAM. Nobody does. That's 16 billion billion bytes of RAM. It's twice 32. 32 is - I'm sorry, it's 16 billion billion. Yeah, that's what I said, twice 32, 32 bits gives you 4 billion, so we square that. So that means that you have this massive address space where you can sort of put yourself wherever you want to. So randomize where you sit within this incredibly large address space using what's called "logical to physical map randomization." So that's one thing you can do.

Then we've talked before, there are things you can do about the various sorts of dynamic memory, the stack is one, where you can put a cookie on the stack, basically a random number. And we did a whole podcast on this years ago where, if code overruns it, then it was like, if something overruns the stack, then it basically will overwrite a sentinel which you have written there. So the idea is, going into a function, you put a little "canary," sometimes it's called, on the stack. And then, on your way out of the function, before you use the stack, so-called "popping the stack," you check to make sure that sentinel is still there, which is to say, nothing in your function or that your function called overwrote that. So that's another thing you can do. And there's more. There are ways to protect the heap and allocate memory in zones and protect it.

So the OS designers have been very clever about this. But anything that is known to the attacker, for example, if you always wrote the same canary pattern, the same cookie on the stack, then the attacker can figure that out. And they can arrange for their overwriting of the stack not to disturb that cookie, or to overwrite and in that one spot put the same cookie. So you need randomness. For all of this you need randomness. And you need what's called "early randomness." That is, eventually, once the operating system gets going, then you're going to start collecting entropy. You've got your hardware random number generator in the Secure Enclave. It's going to be spitting out noise from electrons to reverse tunneling through diode junctions, cool quantum stuff. You're going to have this huge machinery going. But this is all before then.

Yet even before then you still need randomness because this is what attackers - attackers will take advantage of a lack of randomness even there. So Apple has something they call the "early random number generator." They thought they were improving it when they went from iOS 6's style to iOS 7's. And what's clear to me, in looking at the analysis of it from this paper, is no crypto person was consulted.

Leo: Oh. But they were doing so well.

Steve: I know. I know. It must be they're big, and the crypto people were on the other side of the quad or something. I don't know how you explain this. This was done by somebody with the best of intentions, who had no cryptographic security training.

Whoever this was used a brain-dead pseudorandom number generator. And I've talked about them before. That's called a "linear congruential pseudorandom number generator." It's what we used on the Apple II when we were shuffling the deck of cards.

I mean, it's like it's the most godawful thing. You take a number. You multiply it by a constant, and you add another constant, and that gives you your next one. Oh, and you also allow it to overflow, so it's done mod whatever the size of the word is on the computer. So if you had a 32-bit machine, you just take a seed, and you multiply it by a constant, which is typically some prime, and you add maybe another prime to it. And that gives you your next one. And that probably is bigger than 32 bits, so you throw away the stuff that wrapped off. And so, and you just do that over and over and over. And it steps you through the address space. Good ones will visit all, for example, on a 32-bit machine, all 2^32 states.

I used to pride myself, because in the old days, for things that were not cryptographically secure, I would use these to generate randomness. I actually taught myself Windows with a cool screensaver called ChromaZone, and I needed randomness. And so I used one of these. And I spent some time choosing those two constants, the multiplier and the addend, in order to come up with really good random numbers. I mean, really good within this definition of it just sort of looking like noise to get a random star pattern, for example, or random motion of something.

But, I mean, nothing about this is cryptographically secure because, think about it. Basically, if you ever capture its state, if you ever capture that value, and the constant is fixed, the multiplier and the addend are fixed, they're in the code, you now know every future one it will ever produce. Forever. And it turns out it's not difficult to run it backwards. We can do that, too. Which means we can run it backwards all the way to the original seed. So if it's possible to capture its state, you have all of the past and all of the future.

Well, because these are known to not be very good, and it wasn't - for example, the low order bits are, like, useless. I mean, they just form a very simple cyclic pattern. And so the developer of this threw them away. He knew not to use the least significant bits. But that meant he needed more than this generator's width was. And what he did was he used four. So he did it, threw away the least significant bits, took the most significant bits, put them as the first chunk, then iterated it and did that again. Second chunk, iterated it, did it again. Third chunk, iterated again. Fourth chunk. So it turns out that's even worse because now, in a single snapshot, a single random number from this thing, he's given you four states in a row. And it turns out with a simple analysis that eliminates all the uncertainty. So, and then it turns out - so remember, so we have that. The question is, what if this leaks something?

Well, it turns out the early randomness leaks everywhere. It's being used by the operating system. Just sort of get me a random number, get me a random number, whenever it thinks it needs a random number. And so it's not only in the logical-to-physical map randomization in the stack check canaries and cookies and in the address space layout randomization of the kernel called KASLR, Kernel Address Space Layout Randomization. It's not only there. It would kind of be better if it was only there. Unfortunately, they just spray the random numbers from this thing everywhere while they're starting up. So that it's simple for code to get samples of that, determine what the random number was at some point in time, and then go backwards into time to the beginning and then go forwards, recreating what the kernel was getting while it was getting it, and then essentially render all of its uses of that random generator moot. Whoops. So Apple was not happy. And I'm sure there will be a 7.1.1 probably before long because...

Leo: Oh, are there exploits out there already?

Steve: No, no. And so that's the point. Remember we want to differentiate. We want to separate mitigation from exploit. So what this means is, if there is a problem which is found in 7.1, that is, something exploitable, then this dramatically hurts the operating system's mitigation of the damage that can be done because now that everybody in the world knows about this, everyone will work out how to essentially derandomize everything that the 7.1 kernel does while it's starting up. So this is not good. And I don't know if this actually was about 7.1. There's a chance that 7.1 already changed this. I don't know for sure. I think I saw 7.0.3 mentioned, and some change to something, some reference to .06, which was remember the mysterious one that we suddenly got that fixed the SSL certificate problem, where they weren't checking SSL certificates at all.

Anyway, so that's what that's all about. It doesn't itself represent a problem. But first of all, the fact that this - oh, the other thing I should mention that sort of creeped me out about the design was that some bits from the low order were shifted up and XORed to, like, scramble things more. You just put your head in your hands. It's like, oh, goodness. I mean, no security person would ever believe that has any effect at all. I mean, just, like, none. No one who was doing crypto work would just be tripped up by that. So that's an inexperienced programmer who tried to do a good job, and unfortunately came up with something really, really bad.

Now, I don't know why they didn't use the cool hardware. It may be that this is a bit of a problem of the Secure Enclave is it's busy booting up, too, while the main processor core is booting up, and so there just isn't access to random numbers then. I don't know what the interdependency is. Maybe they could boot the Secure Enclave first and then pick up really good entropy from it. That would be nice. Who knows? But certainly Apple has the ability to change these things. This does not feel like it's boot ROM-ish because it's from iOS 6 to iOS 7 this got broken. iOS 6's had some problems, but they weren't nearly as bad as basically creating a purely deterministic pseudorandom number generator that is not very pseudo at all. So anyway, that's what that is about.

I did note just yesterday, and I guess this was news from late last week, Google changed the pricing on cloud storage. I created a shortcut to a nice ReadWrite.com article, bit.ly/sn-google, all lowercase, sn-google, a bit.ly link, bit.ly, which sort of shows the lay of the land. And essentially the 15GB that anyone using Google Drive gets for free is still free. But they dropped the price of the 100GB plan from $5 a month to $2. So, substantial drop there. And they dropped the price of the 1,000GB plan, that is, the 1TB plan, which used to be $50 a month, to $10. So you can get a terabyte of Google Drive storage now for 10 bucks a month.

And the economics of this are clear because I just checked, I was curious, a 4TB Seagate desktop hard disk drive, SATA II or III, 6Gbps, native command queuing, 64MB cache, a state-of-the-art 4TB drive, costs $155. So that's about $40 per terabyte. So that says that, if Google were using those, four months of the $10/month subscription for a terabyte - no, wait a minute, I got that wrong. I didn't do my math right ahead of time. $55 was the drive for 4TB. So 1TB costs $40. Oh, yeah. So in four months of the 1TB plan you - yes, that's right. In four months of the 1TB plan they break even, and most people are going to do more than that.

So then the question, of course, is what is everybody else, all the other cloud storage providers, going to do? And since the economics support it, I think we're going to see a shakeup in the market of pricing and everybody coming down. And it was this, and the

fact that we haven't revisited the topic for some time, that led me to tweet and also right now to announce that we will be revisiting the topic of TNO cloud storage solutions with an updated cloud storage roundup. That was one of our very popular podcasts from a couple years ago where I went through and looked at, I don't know, it was like 12 or 13 of the cloud storage providers. And some of them got dinged pretty badly for their lack of security. Others did well. And I'm looking both at providers and at third-party solutions, like Boxcryptor and Tresorit, and I'm trying to think of the guy who did scrypt. I'm blanking on it. Anyway, so the idea being that you can either use a client in your machine, which I actually prefer, or maybe a turnkey package from the remote provider. Tarsnap, that's what I was trying to think of, Colin Percival's solution, which is also very nice and TNO.

And finally, Wired, someone tweeted, and I appreciated this, said, oh, look, the guy at Wired has clearly been listening to the Security Now! podcast. Wired had an article just out which was under "wishlist." This is Kim Zetter's piece titled "A 10-Point Plan to Keep the NSA Out of Our Data." First point, end-to-end encryption. So, yep, we're completely on track there. No. 2, bake user-friendly encryption into products from the get-go. Exactly right. This is not something the end-user has to worry about. It's very much like we need to move websites from non-SSL to SSL. Simple thing to do. The browsers are ready. The servers are ready. People just have to buy certificates and switch their users over. And then you can be in an open WiFi hotspot, and your data at the website won't all be completely sniffable, which otherwise it absolutely is.

No. 3, as a matter of fact, make all websites SSL/TLS. It's interesting, too, the Defuse Security guy, Taylor Hornby, whom we know as FireXware, and we've spoken of on the podcast from time to time, had an interesting piece. He put it up, I think maybe last week, where he was proposing that it's time for browsers to make it a little more clear when a site is not secure. That is, they do indicate with the lock or the key or, in the case of EV certificates, greenness, they clearly indicate, go out of their way to indicate when a site is providing you with security. But they're very low key about not being. And Taylor proposed, and I think I like the idea, like show a red broken lock to just say, uh, this is not secure, to convey to the typical user, be careful here. I guess the counterargument would be that that would - that makes it look like something is proactively broken. And from the standpoint of someone who believes you really should have secure communications, you could argue that something is proactively broken, or at least broken. But I thought that was an interesting idea.

No. 4, enable HTTP Strict Transport Security. And of course we've talked about that. That's a feature where the web server can declare itself to be always accessible over SSL, which the web browser will cache so that, if the user inadvertently puts in non-S on HTTP, or clicks a link in email that's non-S, or in any way suggests to the browser that they go to this site not over SSL, the browser's cached memory of that site's prior statement using HTTPS, HTTP Strict Transport Security, allows the browser the permission to autonomously upgrade those connections so that any attempt to connect to the website gets upgraded to SSL, which thwarts another whole class of attacks that we've spoken of over the years.

No. 5, encrypt datacenter links. Google knows the importance of that now from the NSA spying on their backbone, essentially. No. 6, use perfect forward secrecy. That's the idea that you are always generating new keys for your conversations, rather than always using the same key. And it's funny, too, because the term is "perfect forward secrecy," but it actually refers to the secrets from the past. That is, perfect forward secrecy means you negotiate a new key so that, if anyone got a hold of your current key, they could not decrypt previous conversations. Of course, there aren't yet any future conversations. But if they had that key, and you didn't change it, they could certainly go ahead and decrypt

those as they occurred.

But, yes, we always want to be negotiating ephemeral keys, generate keys per conversation, and all of the good TNO end-to-end encryption, for example, chat clients just do that as a matter of course. They establish the identity of each other to get authentication, and that allows them to securely negotiate a key which cannot be intercepted by a man in the middle because that man in the middle cannot authenticate themselves as either end. You need authentication in order for a man in the middle to make any sense. I mean, protection against man in the middle to make any sense.

Also 7, secure software downloads. Always download software over SSL, authenticated with the website's name, never not. Because otherwise you just don't know what you're getting. No. 8 is a good one, too. We don't talk about this enough, I think, reduce storage and logging time. I've mentioned this in years past, the notion that we should expire the content of databases, rather than them just living on for decades and decades. It's difficult for a company to demonstrate a valid business purpose for really old information. And people should just know that anything really old is just sort of the Internet forgets about it, rather than right now the Internet holds onto it fanatically because it might have some de minimus value. So we really should reduce that.

No. 9, replace Flash with HTML5. Amen. Long overdue because Flash, even last week at CanSecWest, again, more source of exploits and problems. And I really like No. 10, and you will, too, Leo: Fund a global account to support community audits of open source code. Because, of course, as we've said, open source code demonstrates the goodwill of the people coding it, that they are explicitly saying we have nothing to hide. But the fact that it's open doesn't mean that it's secure. Apple's famous mistake, the goto fail, it was sitting there in an open source repository for quite a while and didn't help anybody. But if the code were audited, I mean, somebody was looking at it, asking line by line, what does this do, what does this do, does this make sense, it would have been spotted. And so I'm really glad that TrueCrypt is being audited now so we will know about it. But the notion of there being a formal code auditing system of some sort that is funded really makes, I think, a lot of sense.

Leo: Yeah, and none of these other proposals make sense unless you do that since most of it would be done with open source. You couldn't, for instance, even Item 1, end-to-end encryption, or bake user-friendly encryption into products, if you can't validate that - why would I accept a third-party vendor's encryption scheme if I can't validate it?

Steve: Right, right.

Leo: Doesn't do anything.

Steve: It's one of the nicest things that Joe Siegrist at LastPass did when I was doing my vetting of it, was that he was able to prove what he was claiming by creating a test site, where you could do these things, and the web page showed you that it was working, and it was very simple JavaScript that anyone could take a look at. So those kinds of steps go a long way towards validating the security which the document says is present.

Leo: Yeah. I don't think any of this is going to happen, but I think it's all a great idea. But who knows, yeah.

Steve: Well, but we're moving forward. I mean, like many people are using TextSecure and Threema because they want secure, end-to-end encryption. My site and other sites are using HTTPS, Strict Transport Security. Unfortunately, we just don't throw a switch and make it happen. But we're definitely going there.

We've had a great reaction, very, very, I'm so pleased about the reaction to the SQRL translation project. Last week, when I mentioned it on the podcast, we had 34 translated languages and 80 volunteer translators. Today, this morning when I looked to put the notes together, we were at 48 languages because so many people wrote and said, hey, I speak Swahili. I speak, I mean, name your language, and I would like to do a translation. 12 people, no, 14 people, 14 people asked for additional languages which they speak and would be willing to translate from English. So we went from 34 to 48. And, boy, have we got the bases covered now. And we've jumped up from 80 to 213 volunteer translators are logged in. So that's really great, too, because we really need, where we can get it, more than one person claiming that this is a translation into Hindi, which by the way is one that was added last week. I don't speak it, and I have no way to verify it. I mean, I am going to trust people.

And if there are any problems with the translations, for example, someone said, well, this is not saying what you think it says, Steve, I'll immediately pull it. But it's great to use essentially a community effect to allow multiple people to be all interacting and agreeing on a single translation. So having more people willing to vet translations - and I imagine, once they actually exist, I'll be able to get some people to say, hey - I'll be able to say, hey, we really need some for Italian. If there are any English and Italian speakers, please check this, just read it over to make sure that it makes sense. So anyway, to everybody, I really want to say thank you. It's going to make a big difference.

Meanwhile, I'm cranking away on code. I've got the app is open and started. And I'm having to do the so-called unicode, or "wide char" as Windows calls it, programming for the first time, where the user interface, all of the UI needs to be representable and displayable in 48 different languages. So that's not something I have ever had to do before. So it's taken me a little bit of time to build a new foundation of tools to work with that, but that's what I'm in the process of doing.

I did want to take a moment to talk about the logo for SQRL. I purchased full, unrestricted commercial rights to that logo that I'm using. I call him the mascot, that smiling little squirrel head. Anyone can see it if you go to - if you look at the UI pages on GRC, I've got his little face on all of the UI, or the SQRL, the Crowdin.net, C-r-o-w-d-i-n dot net slash - I think it's page six, Leo, when you go to the SQRL page. Yeah, Crowdin.net/project/sqrl. Then you'll see the little guy. And I'm happy with it. I mean, I'm not in love with it. It didn't cost me very much. So if anyone wants to come up with something...

Leo: I think it's a chipmunk.

Steve: Well, okay.

Leo: No, I'm kidding. I'm teasing you.

Steve: I don't know if there is a difference. It might be. Anyway, my feeling is it's friendly. This is not tech-y. I didn't want to, like, blur in a bitmap or anything.

Leo: I like it. It's nice.

Steve: The world is going to use this. The problem is, when I reduce it to 16 - oh, and here's our trash pickup.

Leo: Right on time, yeah.

Steve: Right on time. When I reduce it to 16x16, it creates a blurry icon. And so I've got now our little squirrel friend sitting in the tray of Windows because, as I said, I have code, something's running now. And just, if you know what it is, okay, that's our little squirrel. If you don't know what it is, it looks kind of like a little brown blob. So I imagine within the sound of my voice that we have people with some artistic talent. And so if you took the image, for example, from the Crowdin.net translate page - you can also get to it just by saying translate.grc.com, although, as I said, I'm going to be replacing that, I just haven't. I've been focused completely on getting the code written. But translate.grc.com will immediately take you to that page. I would love to have hand-rendered, rather than algorithmically reduced, 32x32 and 16x16 icons for that little guy. That would be really tremendous. So if anyone is interested in doing that, that would be great.

Leo: Sure you can find somebody.

Steve: And a podcast follower, a frequent tweeter and friend of the podcast, Christian Alexandrov, we've visited him a few times - you'll remember, Leo - in Sofia City, Bulgaria. He's the guy with the dentist whose computers are breaking all the time.

Leo: Yeah, that's right.

Steve: In this case, he sent on March 16th, so what's that, two days ago, he said: "After my hard drive failed on me more than a year ago" - so this is probably early in his experience - "I used SpinRite to bring it back to the world of the living. After that, I started running SpinRite regularly, once every month on Level 4. Now my hard drive is stronger than ever, and SpinRite helped my drive to locate and put its weak sectors out of use, leaving only strong sectors in use," which is true. "My hard drive," he writes, "is stronger and healthier than it was when it was new. Running SpinRite regularly keeps my drive in good shape. No hard drive problems ever since."

And it's funny because this was in the mailbag. And in thinking about it, it is difficult for me to understand, if you ran SpinRite for maintenance periodically, how you can have a hard drive problem. SpinRite is going to find the problem before it can manifest as a

problem with readability because SpinRite can perform so much recovery on sectors which are beginning to get weak. And if you run it often, you just - you can't have a problem. And I recognize that, for SpinRite 6, it sort of works against people who want to run it often because it is not running as fast as v6.1 will. And as everyone knows, everyone gets 6.1 as soon as I get back to it, once SQRL is put to bed. Which will, among other things, be way faster. Yet 10 years of SpinRite 6.0 is in the world, and people are using it periodically. So it absolutely does keep your drive from dying. And I appreciate Christian reminding us.

Leo: And we thank the NSA for removing all of Steve's information in a timely fashion. We continue on with Security Now!, Mr. Steve "Tiberius" Gibson. And the garbage trucks have moved on. Good timing.

Steve: They have. Perfect. So as we said last week, the challenge which Apple has taken on and accepted is that of creating something which has really never existed before, a large and powerful and flexible digital ecosystem, which is what iOS is, for a computer. We've got - all the other systems are open. And while Apple, of course, famously gets some criticism for the fact that they're closed, it is that closure which is the only thing that prevents bad guys from installing malware on iOS devices.

And as I said, you have to know, we do know, that there is tremendous pressure to take over people's iPhones. I mean, there's a massive install base of iPhones. They're a huge, juicy target. So if it were possible to get people to somehow hurt themselves, the bad guys would be doing that. So the only way not to have that be possible is to create a closed system where you have an app store, like iTunes, which is curated, where to the best of their ability Apple looks at the applications, verifies that they are from known developers. Which is one of the things that I don't think I explicitly mentioned, but part of this notion that - we were talking about how the kernel gets going, and the kernel has to be signed.

Well, the same is true for apps. So every single app which is submitted is signed by a certificate which Apple issues a developer. So Apple has an identity for every single developer who is producing apps which have a chance of appearing in the iTunes store. So it is a closely controlled system. And what we learn mostly by experience, and we're now at major v7, we were just talking about a mistake that Apple made for the startup of the kernel which hurts the strength of their exploit mitigation during boot time, which I imagine they will get fixed quickly. So it's a matter of iterating and continuing to improve the integrity of this soup-to-nuts lockdown of the system.

So what I learned in reading this paper that we began discussing last week is that you don't achieve that without a virtually obsessive focus and concern for security. As we've often said, a secure system is a series of links of different components. And the system is only secure as the weakest link. So all it takes is one mistake anywhere in this interlocking chain from the time the power turns on and the code in the ROM begins to run and get itself going and then reach out into the file system and load the kernel and verify its signature and go from there. This is all interlocked, and each step protects all of the following steps.

So it takes, I think, hardware support. I don't think you can do this without hardware support of the right kind, which we really now have in the iPhone 5 technology, with the Secure Enclave technology, I guess that's only 5s, and a good source of random numbers. We just saw that there's still a problem with boot time randomness, the need for it, in order to thwart the amount of damage that bad guys are going to be able to do

if they do find a vulnerability somewhere. Apple's going to have to come up with a better source of boot time randomness than iOS 7.1 has. But we do know that, once they get going, we've got the Secure Enclave producing really good, high-quality, not pseudorandom numbers, but true random numbers from quantum properties happening at the chip level.

So the thing I like about what I've seen is, and we'll see some more examples of that in a second, is Apple's design is absolutely user-centric. I saw at every step of the way it was a concern for the user's security and privacy. Apple needs what they need in terms of not having their software stolen, so that's one thing this does is it protects Apple as well as it protects the user. But they've really gone above and beyond in protecting the user. The crypto, where it exists, is unobtrusive. People don't know it happens.

I saw a tweet from someone listening to last week's podcast who said just holding his iPhone now he felt better. I mean, he felt good, like it was a little crypto vault. I mean, it is. It's an incredible piece of technology that we just easily take for granted because it's like, oh, yeah, look, it works. And bad guys can't get in. But oh, my goodness, what it takes to make that true. And the architecture throughout really evidences a total respect for the user's privacy and security. Apple doesn't receive any piece of information that isn't truly necessary for the delivery of the service they are delivering.

And as we'll cover in a second when we talk about iMessage, we're going to get to that now, that there are some things they're doing which are arguably not secure. But again, they've made the ease of use versus security tradeoff, favoring ease of use. But I've seen nothing gratuitous. Nowhere are they just sending some stuff off because, well, it'd be nice to have that. The machine has a unique identity that is fused in at manufacturing time in the Secure Enclave, by the Secure Enclave, and no one knows what it is. Apple doesn't know what it is. You can't ask it what it is. All that you can get is the effect of that key by asking it to encrypt and decrypt and sign things for you. It's just - it's really a beautiful piece of work.

So we covered the secure boot chain last week, where everything is digitally signed, all the way from boot outward. We looked at secure update security, the idea that devices request an update package with their own ID as part of the request, and that the request is signed by Apple with that ID embedded, and the device will only install it, will only accept it as an update if it's got its ID in it, and if Apple has signed it. And that prevents downgrade attacks. That prevents any older version from some other device, for example, being installed in a newer device and then winding back its security, making it vulnerable to things which Apple has already fixed.

And of course we talked about the Secure Enclave, which is a completely separate, logically separate coprocessor. It's on the same silicon chip as the A7 application processor, yet the only communication they have is they're able to share buffers and sort of give each other the heads-up that there's something that the other needs to take a look at in one of the buffers. So a so-called "semaphore communication" using basically sort of mailboxes to send things back and forth.

So we know that we have hardware-enforced protection throughout this thing. We've got the Secure Enclave, the unique ID that never leaves the device. And the other thing nice is that all of the iOS data that exists in memory is cryptographically tied to a particular device's ID inside the Secure Enclave. So even if the keys were divulged, only that device can use them.

Apple gives the example that, because the key hierarchy - and we'll be talking about the file system key hierarchy in a second - because the file system key hierarchy is protected

by a key in the Secure Enclave, which is "wrapped," as Apple uses the term, which just says it's just encrypted, essentially the key for decrypting the key hierarchy, I'm sorry, the file system hierarchy is encrypted using this unique ID. Even if the memory chips were physically removed and put into a different device, nothing would work.

So the memory chips and the processor, if you were going to do something, you can't just take the memory out. The processor is the only thing that ever knows how to decrypt what's stored in the memory. And the memory is never written unencrypted. Remember, it's got that AES-256 DMA encryption engine sitting there, right in the connection between the processor and the memory, so everything is encrypted and decrypted on the fly as it passes back and forth. And then, apart from this unique ID - there's actually two. There's something called the UID and one called the GID, which is the group encryption key, which is, Apple's document says, common to all devices of the same generation.

Other than those two, there's the unique device key and the group ID key, all other cryptographic keys are created by the system's random number generator, the good hardware random number generator in the Secure Enclave. The hardware random number generator seeds an algorithm based on the CTR-DRBG algorithm, which is an NIST standard known strong pseudorandom number generator, seeded with a true random number generator, to generator good entropy. This is done because sometimes there's a greater need for randomness, for entropy, than the hardware can generate. Hardware entropy generators normally have sometimes way less than lightspeed rate at which they're able to produce entropy. These little electrons are only pseudorandom or only - I'm so used to saying "pseudo" - only truly randomly crossing this diode junction at a certain rate. And so that's being sampled at a certain rate.

And then there's some other stages that a hardware random number generator goes through in order to balance it because typically the hardware itself will have some bias. It'll be producing a lot more ones and zeroes. Even though they're random, they're not exactly equal. So there's a whitening process and various sort of post-processing that happens before you finally get true random numbers out the other end. But it could be that your software desperately needs them faster than the hardware can produce them. So it is completely acceptable to use the hardware to produce the seed for a very good cryptographic-quality pseudorandom number generator, and that's CSPRNG, cryptographically secure pseudorandom number generator.

And then what you typically do is there's some limit to how much data you can take out of it before the cryptographers start worrying that, if somebody looked at all of that, they might be able to start guessing what was coming or what had happened previously. So the idea is you only can take so much before you reseed. And so the cryptographically strong pseudorandom number generators will produce volumes of cryptographically secure numbers. And then normally they're being constantly reseeded as the hardware has generated enough new true randomness that it's able to say, okay, here's - start over. And that way at no point are you producing too much randomness from a purely software algorithm that is inherently deterministic. You just want to make the sample that is used between seedings small enough that no one analyzing it can figure out what the determinism pattern is for its generation.

So what about device locking and unlocking? On devices with the latest, the A7 processor, the Secure Enclave holds what they call cryptographic class keys for data protection. And Apple, as we'll come to when we talk about the file system, has four different classes of keys which are chosen to protect files based on the way the file will be used. For example, some files need to be used while the device is locked. If you were downloading something and locked the phone or your pad, iOS will still be able to write

to that file so you can do downloading in the background. There are applications where it makes sense to have things survive locking, and others where it absolutely doesn't. And there are some flavors of that, too. So Apple divides these up into protection classes.

So as I was saying, on an A7-based processor, the Secure Enclave holds the cryptographic class keys for data protection. When a device locks, the keys for data protection class Complete - that's one of the classes, they call it the "Complete" class. So when you lock the device, the keys are discarded. They are overwritten and discarded. And files and keychain items in that class, that were encrypted under the Complete class, instantly become inaccessible until the user unlocks the device by entering their passcode. So this is cool because what this is saying is that the keys are always kept encrypted. And the unlocking process provides the information for decrypting them. But RAM holds the decrypted key, and the encrypted key just stays there sort of as your backup. That's your non-running copy. So if you lose RAM, you lose your only copy of the decrypted key for that class. And locking the device immediately wipes the RAM, and so everything protected under the Complete class becomes unreadable. The keys are gone. There's no way to read it.

And then here Apple explains, "until the user unlocks the device by entering their passcode." We're going to come to that in a second because this is interesting because it turns out it really does matter how good your passcode is. However, there's a caveat to this lock and delete. On iPhone 5s with Touch ID turned on, the keys are not discarded when the device locks. Instead, the keys are encrypted with a key that is given to the Touch ID subsystem. So when a user attempts to unlock the device, if Touch ID recognizes the user's fingerprint, it provides the key for unwrapping the data protection keys, and the device is unlocked.

So Apple explains: "This process provides additional protection by requiring the data protection and the Touch ID subsystems to cooperate in order to unlock." And we talked about how Touch ID and the Secure Enclave end up having an on-the-fly negotiated key which allows them to securely exchange data through the A7 processor, even though it's unable to read it. But so the point is the decrypted keys will get encrypted if you're using Touch ID, and only the Touch ID recognizing your fingerprint is able to decrypt them. And so that's a caveat to them otherwise being discarded when you lock the device. Then Apple explains that the decrypted class keys are only held in memory, in RAM. So they're lost if the device is rebooted.

And you'll notice something. This is an example of the security working sort of behind the scenes. Those of us who have rebooted our iOS devices or our iPhone 5's, for example, who have Touch ID, will notice that we can't use Touch ID after a reboot. And it's not that Apple doesn't want us to. It's that RAM only holds the encrypted value under Touch ID, which Touch ID is able to decrypt. But if we reboot, we lose RAM. So again, for the integrity of the security of the system, Touch ID cannot be used after a reboot. You have to first, once, put in your passcode in order to decrypt the statically encrypted keys into RAM. Then, when you lock the device, Touch ID can encrypt those pending a subsequent unlock.

So again, I mean, this has been well thought out and is really bolted down. And I didn't realize something also, maybe it's in the user specs or people know about this, but the decrypted class keys, which are kept only in memory, are lost if the device is rebooted. But additionally, the Secure Enclave will discard the keys after 48 hours or five failed Touch ID recognition attempts. Now, we know about the five failed Touch ID recognition attempts, that is, you try it five times, then you have to enter in your passcode. It turns out, again, it's not like saying, oh, we want you to enter in your passcode because you missed it five times. It's you have no choice because we don't know what the keys are. I

mean, Apple, they're not in the phone. They're encrypted in the phone. Only Touch ID is able to decrypt them. And if you can't make it happy, nothing is happy, and you've got to enter your passcode. And Leo, were you aware that there was a two-day limit, if you didn't use your phone within two days then Touch ID would no longer work?

Leo: No. No, it works. It's the same thing on a reboot. If you haven't used it in a certain amount of time, or you reboot it, the first time you use it you have to enter in the code.

Steve: Correct, yes.

Leo: That's - yeah.

Steve: Okay. And now we understand why. It is security. It's Apple saying, okay, 48 hours, that seems a little suspicious. Most people are going to be using their phone daily. And so it's a nice tradeoff. If the phone were ever not used for some reason for two days, then you've got to go back to ground zero. You can't use Touch ID.

Leo: Or if you turn the phone off, or if it crashes.

Steve: Exactly, all because it loses RAM, and the decrypted keys are only - they only exist in RAM. So actually there's - I made a note here as I was reading this because this gives a user a bit of a clue about how to increase their own security. If you wanted the most security, then rebooting your phone flushes those RAM-based encrypted keys out so that Touch ID, even if someone fooled Touch ID, they could not crack into your phone. But also it turns out that Apple actively uses your passcode as part of the decryption of the master key. So the quality of your passcode really does matter. It has to be a high-quality passcode.

Leo: So you shouldn't use a four-digit passcode, then.

Steve: Well, it would be better not to. Now, again, if you're using an iPhone 5s with the Secure Enclave, you've got hardware protection. So it's able - and there's no way to brute-force that. Apple does slow down the processing using an iterative key lengthener or strengthener. So that prevents guessing. But it turns out, if you ask for a complex passcode, you know how it gives you the full alphanumeric keyboard? It turns out, if you only use numbers to create a longer numeric-only password, next time Apple prompts you, it gives you the 10-key pad. So it doesn't give you the whole big keyboard again. So it's sort of another sort of nice compromise. You might want to use a numbers-only passcode, but not be forced to use the normal alphanumeric keyboard.

And so Apple recognizes on input, they have no way of knowing it afterwards because they've hashed it like crazy, and so they have no idea about the password, but while you're inputting it, if you only touch on numbers, when Apple then prompts you, they set a flag saying give them the numeric keypad. And it's just much faster to enter in a longer, numeric-only passcode.

Leo: And given that, I mean, I understand a four-digit passcode isn't very safe. But given that you can, and I would suggest people do, turn on the thing that after 10 tries it resets the machine and erases everything, that that's sufficient. If you can only do it 10 times, I mean...

Steve: Now we know it doesn't set a flag after 10 times. I mean, all it has to do is scrub the keys that it has in RAM. The only way it's able to operate is because it has those keys that it has not forgotten. And when it loses those keys, when the Secure Enclave scrubs those keys, it's over. Nothing can decrypt the contents of your RAM. That I am absolutely sure about.

They said of passcodes, they said: "By setting a device passcode, the user automatically enables data protection. iOS supports four-digit and arbitrary-length alphanumeric passcodes. In addition to unlocking the device, a passcode provides the entropy for encryption keys, which are not stored on the device. This means an attacker in possession of a device cannot get access to data in certain protection classes without the passcode." So it's not like the passcode is checked against something and then a key is released. The key is synthesized from the passcode.

Leo: It's tangled with the on-chip ID.

Steve: Yes, it's tangled, yes.

Leo: So that's an important distinction. It's not erasing any data. By just forgetting the key, the data's effectively erased because it's effectively encrypted.

Steve: Yes, it is always encrypted.

Leo: Same way you wipe an encrypted hard drive. You don't need to wipe it.

Steve: Correct.

Leo: Just delete the keys, and there can't be any data.

Steve: So data protection classes, I've mentioned before, all files, when a new file is created on an iOS device, it's assigned one of these four classes. And a random number is chosen to encrypt that file. So we have per-file 256-bit AES keys chosen at random which is added to the file's metadata. Metadata, of course, is like filename and date written and created and so forth. So that key is in the metadata. That's protected. That's what I was talking about when I talked about the file system key that encrypts all the metadata. So without that, nothing in the file system is visible - the hierarchy, the names of the files, and the keys to access the files, which are individually encrypted with randomly chosen keys. I mean, it just - it's a beautiful hierarchy of interlocking encryption.

So the one class I already talked about, the Complete Protection, is the class where the key is wiped when you lock the machine. So if something has Complete Protection, when the device is locked, the key is scrubbed from memory, and it doesn't exist anywhere in the device. So all of your memory is immediately protected.

Then they have one called "Protected Unless Open." Apple explains that: "Some files may need to be written while the device is locked, for example, a mail attachment downloading in the background," the example that I cited before. This is achieved using asymmetric elliptic curve crypto using ECDH," Elliptic Curve Diffie-Hellman, over the same curve I chose for SQRL - this is another one of those odd, yes, the people who really are focusing on security are choosing the same solutions - over Curve25519.

"Along with the usual per-file key, data protection generates a per-file public/private key pair" which allows Apple to hold the file open, even when the other keys are scrubbed, in order for background work to be possible. And as soon as the file is closed, that per-file key is also wiped from memory. Then they have the third class is protected until - they call it "Protected Until First User Authentication." They explain: "This class behaves in the same way as Complete Protection, except that the decrypted class key is not removed from memory when the device is locked. The protection in this class has similar properties to desktop full-disk encryption and protects data from attacks that involved a reboot. This is the default class for all third-party app data not otherwise assigned to a data protection class."

So they're saying protected until first user authentication means that, unless you restart the device, that is, remember that restarting, or as you said, Leo, if it crashes and you have to reboot, anything that wipes RAM takes this away. But there are apps that do want access to their data while locked in order to do background things. So they would say - they would explicitly say don't give me complete protection for these things because I need to access these even when the device is locked. But even so, if a reboot occurs, this key gets wiped. And of course, because you no longer have things running in the background after a reboot until you've unlocked and restarted.

So again, this is why I say, if you really want - if you're going to, like, store your device for a while, your iOS device, that's one of the things I learned from sort of seeing how this architecture works. Doing the turnoff, turning it off explicitly, powering it down, that really puts you into, like, the ground state for iOS protection.

And the last class, the fourth, is "No Protection." And they say: "This class key is protected only with the Unique Device ID and is kept in Effaceable Storage," as they call it, which they're securely able to wipe, even though it is a nonvolatile NAND technology. "Since all the keys needed to decrypt files in this class are stored on the device, the encryption only affords the benefit of fast remote wipe." So No Protection means the files are always available, but only because there is a nonvolatile key that is kept in the Secure Enclave. If that is ever lost, which is what the secure wipe does, then, pow, the No Protection file class is gone. And that's, for example, what the file system uses so that you can instantaneously wipe the file system. And that file system contains all the keys for the files. So if you wipe the file system decryption, you have like a nice chaining cascade that means that everything downstream of that is unavailable also.

On the side of app security, I mentioned that only known registered developers that have been vetted by Apple that have credentials receive certificates that allow them to sign their code to cryptographically state this is from them. That's submitted to Apple. Apple verifies their signatures and then signs it themselves so that their devices will accept it. So again we have a nice chain of security based on a root certificate and then children from that.

Apps are inherently sandboxed without any access to other app resources unless that's originally arranged for by each end. The apps are assigned a randomly named file system directory. It's a bizarre-looking thing. If you've ever seen like the name of an iOS file system entry, it's just gibberish. And so, again, that's further strengthening so that bad guys don't have any known points in the file system tree that they're able to access. Address space layout randomization is enabled for all Xcode-produced code, which is the developer tools that Apple makes available for creating iOS apps. So the apps themselves are scrambling their bits so that they're taking advantage of address space layout randomization not being in the same place all the time.

And iOS takes advantage of the ARM processor's Execute Never feature, which is a bit which restricts where code is able to execute. In other words, the stack is nonexecutable; the heap, where memory is allocated, is nonexecutable; data segments are nonexecutable. And that goes a long way toward preventing code from doing all the games it used to be able to play before the hardware was enforcing a refusal to allow the processor to execute instructions which were meant to be data.

Leo: Can I ask you a question?

Steve: Yeah.

Leo: It was about a year ago, and I'm looking at this article on CNET by Declan McCullagh, that documents leaked out that Apple had a waiting list to decrypt iPhones. And the presumption was, while law enforcement can't examine the contents of an iPhone, Apple has the capabilities and, in fact, has a waiting list of requests, which they service.

Steve: Yeah. Unfortunately...

Leo: What you're describing sounds like it would be impossible to do that.

Steve: The problem would be if someone protects themselves with a weak password or the four-digit code. Although maybe Apple has designed themselves to a point where they can't. The Secure Enclave, I think you're right, Leo, what I described where the unique ID never leaves the Secure Enclave, it may no longer be possible for that to happen.

Leo: And this is something that they started doing more recently.

Steve: Just the 5s. Just the 5s.

Leo: So this is a year-old article. So maybe that's the case, that this has changed. Because I wouldn't be surprised if Apple's response to a waiting list of law enforcement asking to unencrypt iPhone data would be to, well, let's make that impossible.

Steve: Yes. Yes.

Leo: Let's not do that anymore.

Steve: Yes, and users benefit. I mean, because Apple can't say no if they have the ability to do it. So they said, okay, we don't want the ability to do it. We don't want users knowing that we have the ability to do it. I don't think they have the ability to do it.

Leo: And that's, again, everything we're describing is specific to the newest iPhone, not older iPhones.

Steve: Yes, yes. Under "Accessories," add-on accessories, because, again, security has to go push itself all the way out to the entire perimeter, Apple writes: "When an accessory communicates with an iOS device using a Lightning connector cable or via WiFi or Bluetooth, the device asks the accessory" - that is, the Apple device, the Apple iPhone or iPad - "asks the accessory to prove it has been authorized by Apple by responding with an Apple-provided certificate, which is verified by the device. The device then sends a challenge, that is, the iPhone or iPad, sends a challenge, which the accessory must answer with a signed response." So individual accessories that plug into iOS devices have to have crypto in them and a unique certificate that they use to sign a challenge from Apple.

And then, get this. I'm thinking, wow, how do you enforce this? I mean, how do you make that practical? This process is entirely handled by a custom integrated circuit that Apple provides to approved accessory manufacturers and is transparent to the accessory itself. So essentially Apple has closed down the hardware attachment ecosystem to the point where you have to, if you're going to make high-end accessories for Apple devices, you need to get chips for them, one per device. So they totally control the attachment device market and put these chips in your devices, which is the only way to get the device to authenticate itself to your Apple device. Wow. And absent that authentication, that is, if the accessory does not provide that authentication, its access is limited to analog audio and a small subset of serial UART audio playback controls. So basically, headphones is the only thing. Headphones have no chip. And so headphones with stop, fast-forward, play, and so forth, that's all you can do.

Leo: Again with the analog hole. You can't get around it.

Steve: And Leo, we've run out of time again.

Leo: You're not going to do a Part 3, are you?

Steve: I can't believe it. But I've saved the best part for last.

Leo: We're still waiting, about this elliptic curve problem. We still wait. We wait on.

Steve: Okay, next week, next week.

Leo: That's all right. That's good. Yeah, fascinating stuff. Of course all of this is immediately eliminated by a simple addition of a goto fail line in some code. But, no, actually it's not, is it.

Steve: No. That was application land problem, as opposed to, I mean, it was a library in the OS. So all applications that depended upon that library were subject to that attack.

Leo: The SSL library.

Steve: As far as we know, no one ever took advantage of it. It was something that was fortunately discovered and fixed quite quickly, as soon as Apple realized their mistake.

Leo: There you go. There you go.

Steve: So next week, the crown jewels of the iOS security adventure, which continues.

Leo: It's a great subject. And it really is an object lesson in how this stuff ought to be done.

Steve: Yeah, it's fascinating.

Leo: Steve Gibson is at GRC.com. That's where you'll find SpinRite, the world's finest hard drive, maintenance, and recovery utility. It's also where work goes on on SQRL. And if people want to give you a squirrel, a 16-bit squirrel...

Steve: Yes, right, right, right, I completely forgot that. You can reach me through the mailbag at SQRL. In the SQRL pages there is a feedback page. There's no way to submit a binary, but tell me you've got something, and I'll send you an email address that you can use.

Leo: Should they tweet at you? Could they tweet it at you?

Steve: Yeah, you can also tweet because I watch my Twitter feed very closely. But for people who aren't tweeting...

Leo: Who's not tweeting? Everybody tweets.

Steve: Eh, there's some old farts. I mean, I wasn't before I was.

Leo: We took a while to get him there, but we got him there: @SGgrc. That's the Twitter handle for Steve, @SGgrc. He has 16Kb audio at the site. He also has very nice professionally transcribed transcriptions at GRC.com. You can go there to post feedback. Eventually we'll do another Q&A session.

Steve: That's two weeks from now.

Leo: Two weeks, at GRC.com/feedback. And we have full-quality audio and video on our site, as well, TWiT.tv/sn for Security Now!, one of the oldest URLs on TWiT.tv because this is the second oldest show on the network.

Steve: And I have one piece of news, believe it or not.

Leo: What's that?

Steve: I just got notified, Firefox v28 is released.

Leo: Well, there you go.

Steve: Just this instant. So I don't even know what it is, but I'm going to go get it, and so should everybody else.

Leo: Go get it.

Steve: It probably fixes the Pwn2Own problems.

Leo: I would guess that'd be the first thing they'd address.

Steve: Yup. Yup.

Leo: You can - let's see, what else? You can watch us do the show live, that's always fun, 1:00 p.m. Pacific, that's 4:00 p.m. Eastern time, 20:00 UTC, every Tuesday on TWiT.tv.

Steve: To hear us talk about my evolving beard.

Leo: Yeah, we do stuff before and after the show that's unique. It is not available on any podcast. Unless somebody else makes one. I always - I fear that someday somebody might make a show of the stuff between the shows.

Steve: Outtakes.

Leo: Yeah. That would be bad. What else? I guess that's it. We'll just have to take a break, adjourn now. Before You Buy is coming up. Steve, thank you so much. We will talk again next week.

Steve: Thanks, Leo.