SECURITY NOW!

Transcript of Episode #532

Verifying iOS App Conduct

Description: Leo and I discuss a very busy week of interesting - and somewhat distressing - security and privacy news. Then we explore the fundamental problem with iOS application security enforcement which is going to take Apple some time to resolve.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-532.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-532-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. Lots of security news. We're going to talk about everything under the sun, and then Steve's going to take a look at and explain why iOS is inherently insecure because of a design decision Apple made when the first iPhone came out. Wow. iOS App Security, coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 532, recorded on November 3rd, 2015: Verifying iOS App Conduct.

It's time for Security Now!, the show where we talk about your security online with this guy right here, the Explainer in Chief. Steve Gibson, hello.

Steve Gibson: Hey, hey, Leo. Great to be with you again.

Leo: It's good to have you with me.

Steve: Yeah, as we approach the end of this year. We're now in November, and Halloween passed, and Thanksgiving three weeks away, and I'm watching the bubbling politics of the United States presidential election. It's quite a circus, quite entertaining this year. All kinds of stuff happening.

Leo: Well, today's a good day to talk about that because our good friend Drew Curtis, the founder of Fark, is running for Governor of Kentucky. Today is Election Day.

Steve: Yup.

Leo: If you're in Kentucky, get on out. Actually, if you're anywhere where there's an election in the United States, and there are many jurisdictions where there are, go on out and vote. And if you're in Kentucky, do yourself a favor and vote for Drew. All I have to say about that.

Steve: Good, yes.

Leo: I have an opinion. Actually, we had a great interview with Drew on Triangulation, if you want to know more about him.

Steve: So our main topic is, as I promised, verifying iOS App Conduct. I first had "behavior," App Behavior. And I thought, no, it's really the conduct. I thought "conduct" is like a better word for what we're aiming at. And as I promised, I completely absorbed the recently released research paper and came away deeply disturbed because Apple has essentially painted themselves into a corner due to the total history of the iOS platform. And that'll be the topic, once we catch up with the news and other miscellany.

Leo: And this is in response to a malware threat on iOS.

Steve: Correct.

Leo: That took advantage of this.

Steve: Yes. There was a Chinese advertising SDK that we talked about last week, which was found to have been used by a number of Chinese apps. And it had behavior that the apps that used the SDK likely did not know about because it was sending information back to that Chinese advertising vendor's servers directly. And it was doing this against Apple's rules, which meant that it had, like all of the apps which had used this software development kit as one of the libraries for their own use, they all slipped past Apple's iTunes Store vetting process. And what I have learned is the vetting process is fundamentally broken, and it cannot be fixed. It will take abandoning the Objective-C platform. It cannot be fixed. And essentially they've been relying on undocumentedness, you know, obscurity, in order to sort of be secure.

Leo: Wow.

Steve: Yeah, it's kind of eye-opening. So I think our listeners will find it interesting from sort of a, what's really going on there, and also just sort of there's a lot of interesting sort of fundamental security lessons built into this.

But we have a lot of news. There was a brief glitch in uBlock Origin's expected availability on the Chrome Store. Symantec has screwed up certificate issuance big-time. The Hacking Team has returned. There is a Tor Messenger which just entered alpha, or I'm sorry, first beta. The U.S. and U.K....

Leo: Let me just interject. When we talk about that, I will admit to doing something really stupid that you will laugh at with the Tor Messenger, but I will tell you about that when we get there.

Steve: Cool. The U.S. and U.K. appear to be taking diverging crypto and cybersecurity paths. We'll briefly cover that. JavaScript 6, or technically ECMA, ECMAScript, has been ratified, and I thought I'd give us a quick little peek at that. We won't go into it in depth; but, boy, it's acquired a bunch of new features. And it really is just sort of a mess.

But for those who love it, I mean, it is the application programming language for browsers on the Internet, although it's got some competition coming up that we've talked about in the past. Threema, the messaging client or messaging system, has received an independent audit. We've got a bunch of miscellany from NASA and Star Trek and "Fargo" and more. And then we will plow into the details of the corner that Apple painted themselves in on day one with iOS apps.

Leo: I can't wait to hear about that. All right. Let's get to the security news.

Steve: So New Scientist tweeted a picture that one of my great followers forwarded to me, that I got a kick out of. My caption for it was, "Has the Internet of Things (IoT) Gotten Out of Control?" And so here we see a list of features for the SmartSpoon. And so its elegant, ergonomic design has fingerprint recognition, WiFi, Bluetooth, and USB connectivity. Autosyncs with other SmartCutlery. You can recharge it, of course, with the SpoonHub. The Spoon location is made easy with FindMySpoon, the smartphone app; and you may track, download, and analyze your spoon usage with SpoonStats. So it's like, I'm sure this will be popping up on Kickstarter any minute.

Leo: Oh, it's huge, the new quantified cutlery. It's already a big deal.

Steve: Oh, boy. So, okay. So we've talked about uBlock Origin, which is our, the podcast's, or at least mine, and I know that you're using it, chosen - essentially it's more than an adblocker. It's an HTML filter because it gives us so much control over what our browser is doing with the page it receives relative to other domains that it may or may not pull stuff from. And five days ago Google posted or sent a note to uBlock Origin's developer, who goes by the handle "Gorhill." And we'll remember that I sort of have characterized him, just based on his writings, as a hybrid, like, what you would get if you merged John Dvorak and Richard Stallman, you know, just sort of, you know...

Leo: Wow.

Steve: ...a unique character. So Google sends him a note saying, "Dear Developer, your Google Chrome item, uBlock Origin, with ID" - and then it's got some huge random junk of lowercase alphabetic stuff, which is the ID for his app in the store - "did not comply with the following section of our program policies." And then they quote themselves, saying: "Where possible, make as much of your code visible in the package as you can. If some of your app's logic is hidden, and it appears to be suspicious, we may remove it."

Now, Gorhill, cranky as he is, replies to this, and he posted this on his GitHub log or blog there. And he said: "This amounts to, 'There is somewhere one or more pieces of code I don't understand, but I won't tell you what it is. Your challenge is to find what I'm talking about and modify it so that, in my next review, maybe I will understand it.'"

So then some time passes, and this caused a great deal of concern that uBlock Origin - of course, now, understand that we're kind of watching this because we know that Google is basically funded from its advertising revenue, and uBlock Origin, one of its main benefits is it's an easy-to-use adblocking, ad controlling add-on. So there's some tension, we would imagine.

So then - and I got a lot of tweets about this, a lot of concern. The entry in Gorhill's posting goes on and on and on with people upset and responding and so forth. Then Google comes back and says: "Dear Developer: We apologize that the update was rejected due to a snag in the review system. The updated item will be available in the Chrome Store within 30 minutes. Thank you for your cooperation." Google Chrome Web Support Team.

Leo: And in fact I don't think it was ever taken down.

Steve: It was never actually removed, correct. And then Gorhill finally apologizes to the list, saying: "Sorry for all this. That really got me worried. If this happens again, I'll wait a bit more for feedback from the Chrome Store before reporting it here. Unclear, though, whether making such an issue widely known sooner than later helps with its resolution, or at least a faster one."

So, I mean, there were a lot of people who apparently wrote letters to the Chrome development team complaining about this and saying, hey, you know, this is all on GitHub. It's all there. All you guys have to do is look at it if you really care to. So, I mean, one of the problems that Apple has that we'll talk about is also the problem that Chrome has, is that there's an amazing influx of apps for Android and for iOS. And so they necessarily depend to a great degree on automation to do sort of first-pass verification of what the apps do. And then somehow it falls to a team in order to look at it more closely.

But, boy, that's not a job I would want to have. I mean, it's - so many examples of what we've talked about through the years past have been how difficult it is to look at some code and determine whether it might have some unwanted conduct. It's just next to impossible to do that. So both of these ecosystems face a challenge I would never want to have. And it's one they don't have any choice but to have because they're wanting both to offer their users some guarantee of security against a world unlike what we've had before.

I mean, you know, we never had in the Windows or the Mac traditional application world, we just - we had the presumption that apps we installed were not going to be malicious. We had some control over them by not running ourselves and thus the apps we spawned as root, so hopefully the damage they could do was minimal. But generally, if something was malicious, we'd remove it and let the world know as best we could. But now we're in a whole different world, where apps cost a few dollars. People say, oh, I'll try that for a day, and click on stuff. And in fact there was some piece of news where thousands of apps that were free, it tied into Baidu, I think. Is that pronounced right?

Leo: Baidu, Baidu.

Steve: Baidu.

Leo: Yeah, the Chinese search engine.

Steve: Exactly. It was free apps that they were making that had some unwanted conduct. And them being free was part of the hook. It was because, I mean, who's not going to get something that's free. Unfortunately, they were misbehaving themselves. So, boy, I mean, it's a big problem, and I don't see how we solve it.

Leo: You know, the thing I loved in this whole Gorhill thing was the way you interpret this very much had to do with your opinion of Google.

Steve: Uh-huh. Yup.

Leo: Like, you know, it confirmed everybody's firmly held belief that Google is evil, if you had that firmly held belief.

Steve: Right.

Leo: And I think it's completely possible that that's true But I think it's also very possible that, literally, this was a bug. It spat out what was essentially an incomprehensible email, which kind of leads me to think it was a bug, and they fixed it. And that happens all the time. It happens on Apple Store, as well. So it doesn't have to be malfeasance for this to have happened.

Steve: Correct. I agree. And it might well be that, like, this was the automated phase.

Leo: Right.

Steve: And that most people who are trying to push something creepy through, they just, oh, well, okay, that didn't work. And they're never heard from again. And so it might be that good apps that should not be denied, there may be a false positive rate. And when that happens, if the developer stands up for himself and says, hey, wait a minute, what's going on, then that escalates it to the next level of examination. Someone looks at it and goes, oh, yeah, that was, you know, we didn't mean to reject that, sorry.

Leo: Whoopsies, yeah.

Steve: So Google has taken what has surprised the industry with the strength of their response, although I can't say that I think it's over the top. And that is that Google has declared that they're basically putting Symantec, one of the major security certificate authorities after they bought - they bought VeriSign; right?

Leo: Yes.

Steve: And VeriSign was like, you know, I was once upon a time using VeriSign.

Leo: Me, too. That was the one; right?

Steve: Yeah, exactly. And, I mean, I happily escaped to DigiCert, where I'm way happier. Anyway, so here's, to get a sense for what has upset Google, a little after the middle of last month, or actually, no, now we're in November. So it was September 18th, so maybe, what, six weeks ago. Symantec, in a rather clueless blog, I mean, I looked at the title of it after I knew what their behavior had been. They titled it "A Tough Day as Leaders." And I thought, oh, god.

Leo: Oh, you jerks. You're jerky.

Steve: Get over yourselves.

Leo: You don't really want to combine an apology with a boast at the same time. They don't really go together.

Steve: Exactly. And you didn't build this, you bought it.

Leo: Yeah.

Steve: So they bought VeriSign. So, yeah, a tough day as leaders, it's like, okay.

Leo: Please.

Steve: So they say: "We learned on Wednesday that a small number of test certificates were inappropriately issued internally this week for three" - three, now, listen to these numbers because this comes back to bite them bad - "three domains during product testing. All of these test certificates and keys were always within our control and were immediately revoked when we discovered the issue. There was no direct impact to any of the domains and never any danger to the Internet. Further, we are in the process of proactively notifying the domain owners and our major partners.

"In light of these events, we must reassert our commitment to stand behind our values and our position as a trusted industry leader. While our processes and approach are based on the industry best practices that we helped create, we have immediately put in place additional processes and technical controls to eliminate the possibility of human error. We will continue to relentlessly evolve these best practices to ensure something like this does not happen again.

"In addition" - okay, now, this contradicts what they just said. "In addition, we discovered that a few outstanding employees" - now, I don't know if they meant "outstanding" in the sense that they were outstanding employees, or...

Leo: They're out standing in the unemployment line, I hope.

Steve: Yeah, "that a few outstanding employees" - meaning that at that time they still were - "who had successfully undergone our stringent onboarding" - to create some yuppie verb...

Leo: I hate that. Ugh.

Steve: I know, "our stringent onboarding and security trainings, failed to follow our policies. Despite their best intentions" - okay, how does that work? - "this failure to follow policies has led to their termination after a thoughtful review process."

Leo: What, their best efforts not to get fired?

Steve: Please don't sue us.

Leo: Please don't fire me, please, I don't want to...

Steve: Yeah.

Leo: Oh, lord.

Steve: "Because you rely on us to protect the digital world..."

Leo: Yes.

Steve: Parens, which we helped create...

Leo: The entire world relies, yes.

Steve: I put that in there.

Leo: Yes.

Steve: "...we hold ourselves to a 'no compromise' bar for such breaches."

Leo: Oh, that's a relief, yes.

Steve: "As a result, it was the only call we could make." Boo hoo. Okay. "As much as we hate to lose valuable colleagues" - oh, they were more than employees. After their "onboarding" they became valuable...

Leo: Please don't sue us.

Steve: ...became valuable colleagues. "We are the industry leader in online safety and security, and it is imperative that we maintain the absolute highest standards. At the end of day, we hang our hats on trust, and that trust is built by doing what we say we're going to do."

Leo: Well, their hats must be on the floor by now. Geez, this is the worst.

Steve: Okay. So remember, this was a tough day as leaders, is this.

Leo: Days as leaders, yes.

Steve: Now, over at Google, somebody was spitting their coffee out, and his name is Ryan Sleevi. He blogged Wednesday. He said, and I have the link in the notes, he said: "Following our notification, Symantec published a report in response to our inquiries and disclosed that 23 test certificates had been issued without the domain owner's knowledge, covering five organizations, including Google and Opera. However, we were still able to find several more questionable certificates using only the Certificate Transparency logs and a few minutes of work. We shared these results with other root store operators on October 6th to allow them to independently assess and verify our research. Symantec performed another audit and, on October 12th, announced that they had found an additional 164 certificates..."

Leo: Well, it's close to three. Close. It's within an order of magnitude.

Steve: "...over 76 domains, and 2,458 certificates issued for domains that were never registered. It's obviously concerning that a certificate authority would have such a long-running issue, and that they would be unable to assess its scope after being alerted to it and conducting an audit.

"Therefore, we are firstly going to require that, as of June 1st, 2016" - so they get seven months, eight really - "all certificates issued by Symantec itself will be required to support Certificate Transparency." Meaning that everything they issue is logged publicly. "In this case, logging of non-EV certificates would have provided significantly greater insight into the problem and may have allowed the problem to be detected sooner. After this date, certificates newly issued by Symantec that do not conform to the Chromium Certificate Transparency policy may result in interstitials or other problems when used in Google products."

So this is an unprecedented slap, saying essentially that Symantec, the "leaders of the industry," behaved so badly in taking responsibility after notification that they can no longer be trusted. And we're going to require that every certificate they subsequently issue is publicly noticed, and we're going to verify it. And Chrome, the browser, will essentially blacklist any certificate that VeriSign/Symantec issues which we have not first been notified of the issuance of by Symantec.

And I don't think, again, I don't think that's too much to ask. We've seen certificate authorities make mistakes. And everyone acknowledges that, while they can be bad, what a CA has to do is respond responsibly. And in general, that's the only thing we ask for from anyone. When Joe at LastPass found some scary network activity, it was his response to that which was more than adequate and immediate, saying we don't even know if there was a breach, but something - we don't understand why there was this much network activity on someone's workstation when he wasn't there, so we're going to err on the side of caution.

Similarly, a certificate authority has to say, oh, my god, immediately do an audit, and absolutely come clean, saying, okay, thank you for catching this. This is what we found. They're all canceled. They're revoked. We pushed out revocation. We're doing everything we can. I mean, that's what you want.

What Symantec did cannot be understood because they initially said, and in their public pronouncement of their grandiosity, they said, oh, yeah, we found three test certificates that never were used publicly, never happened, blah blah blah, and then it's thousands. So, sorry, Symantec, you've lost the trust. And anything you do after June 1st, 2016, for some length of time is going to have to be, you know, every single certificate you issue, you notify. Oh, and that's not just EV, but also non-extended validation certs. So this is...

Leo: Does that put them out of the business, or, I mean...

Steve: No.

Leo: No.

Steve: No, it won't put them out of business. But it's just sort of - it's a conduct slap. It's saying we do not feel we can trust you based on your behavior. Your behavior is the only thing that a CA has. I mean, that's, you know, it's their policies and their operational security that we must trust because all of our clients are trusting all of the certificates that any of the CAs issue. And this is the weakness of the entire public key infrastructure as we have it today is that we're trusting all these individual entities. And every single one of them must never make a mistake.

Well, that's a weak system. It's the best we've been able to come up with so far, but it's accident prone. And so when an accident does happen, and then from like a major certificate authority that's issuing a ton of certificates because, as we said, VeriSign was - they were always my certificate authority until they just got so annoying that I went in search of an alternative. So, yikes. Yeah, a tough day for the leaders. Yeah, okay, Symantec.

Leo: Hard to be a leader, I'll tell you.

Steve: Oh, yeah, got a few arrows. So Hacking Team is back. We'll remember that about four months ago a hacker known only as Phineas Fisher, as he identified himself, hacked into the Hacking Team and put 400GB of their previously private internal data, including emails, customer lists, and the source code of their proprietary RCS, basically a remote access trojan, a RAT. They called it RCS, Remote Control System. And their customer list showed that U.S. law enforcement, you know, FBI, NSA, CIA, DEA, under sort of shell corporations, were actively buying this technology as part of their cyber efforts.

So, and of course we also discovered a bunch of zero days that were completely unknown. It took a while because 400GB of data is like you're drowning in wealth there. But what we found was Flash zero days, like a bunch of them, and some Windows zero days. So these are things that they were leveraging themselves and/or selling to others for their own use.

So this was a massive black eye for this group. And it's taken them now about four months, on the 19th, a couple days ago, or a couple weeks ago, rather, their CEO, David Vincenzetti wrote to their mailing list - which by the way we have, and we know everybody who's on it. We know everything. He wrote: "Most law enforcement agencies in the U.S. and abroad will become 'blind.' They will 'go dark.' They will simply be unable to fight vicious phenomena such as terrorism. Only the private companies can help here. We are one of them. It is crystal clear," he writes, "that the present American administration does not have the stomach to oppose the American IT conglomerates and to approve unpopular, yet totally necessary, regulations."

So that he is saying that Apple and Google and others are being allowed to have undecryptable end-to-end encryption. And so he's saying that you - and this is a newsletter out to their customer base, basically, to global law enforcement and governments and others who have purchased from them in the past. He's saying, look, you need our stuff. So the Hacking Team is "finalizing brand new and totally unprecedented cyber investigation solutions." And what has been reported sort of in the fog surrounding this is that some companies are remaining; others are reconsidering whether they want to have an affiliation with this group because basically their past affiliation was outed, and that hurts, and they don't want to stay there.

So anyway, I think these guys are no doubt going to have to build up another repository of attack vectors. Unfortunately, they really do appear to have skills. They had a treasure trove of amazing goodies that no one knew about in the public, and that they were no doubt leveraging into some serious cash flow for themselves. So that got lost. And of course, as all those zero days were found, the industry was patching them as fast as they were dug out of that 400GB data dump. They're going to need some more. But that's apparently what they're working on. And unfortunately, there is arguably a place for them.

I mean, we do want, we as users, and companies wanting to offer security, we want truly non-backdoored, non-broken in any way, end-to-end encryption. What that means is that law enforcement will be forced to insert shims into systems to get in before the encryption is applied because the data in transit, unless there is a way for a third party to crack that by altering keys or something, I mean, it's not beyond the realm, but in terms of like the weakest point of attack is the computer that is sending or receiving it before it gets encrypted and after it gets decrypted.

And there your classic model is go get a warrant in order to get legal authority to infect some suspect's machine with something that allows law enforcement that kind of shim. And so I think that's the way we're going to see it evolve. That means that, though, this is highly specialized. And so that's not skills that the FBI typically has, so it makes sense for there to be an underworld third-party company with the expertise to then sell that capability to law enforcement that, under court order, is able to use it. I think that's the model.

Leo: This is not the only company doing this. There must be other agencies, other groups offering these kinds of skills.

Steve: Well, yeah. And, I mean, you could see how it would appeal to a certain class of skilled hacker. They find defects in commercial products. And of course we know that that's happening all the time. All of the hacking competitions are always breaking all of the browsers, one way or another. So this can be done. And Adobe Flash is constantly being patched. It's not like it's stopped being patched. It's always being fixed.

Leo: Right.

Steve: You know? As is Windows. So, you know, we have very complex systems that have not yet been secured. And so I guess my point is that I don't even view that as necessarily nefarious.

Leo: No.

Steve: Depending upon who they sell their shims to. If they sell it to law enforcement, and assuming that law enforcement acts responsibly, legally, essentially, constitutionally, and gets a search warrant, then I'll actually like that because it takes pressure off of the end-to-end encryption system that we want to have by allowing law enforcement a legal means of achieving what they want to break encryption for, but not by breaking encryption, by getting in before the encryption.

So, Tor Messenger. You had something interesting?

Leo: So I, like you, when they announced this, I said, oh, yeah. And I took a look at it. And one of the things that they offer - I installed it. One of the things they offer is to be the rooting for more mainstream messenger services. So they support XMPP and Jabber. And I noted Facebook Messenger, and I thought, well, this would be kind of cool. I mean, obviously Facebook would still know everything you're doing.

Steve: And Google Chat.

Leo: And Google Chat, which used to be Jabber/XMPP.

Steve: Right.

Leo: So Facebook would know. But no intermediary would know. And so nobody could spy on your Facebook Messenger, which would be kind of cool. So I installed it, and I clicked the button that said "Connect Facebook." And I logged in, and it didn't work, probably because I have two-factor, and it didn't have a way of accommodating two-factor.

Steve: Ah, right.

Leo: Then later in the day I got an urgent message from Facebook on my Facebook page saying, "Somebody has tried to log into your Facebook account from Egypt. We've disabled your Facebook account, and we won't let you back until you change your password. And stupidly, I just freaked out, and I went oh, no. And I tweeted about it. I said, you know, I'm worried that Tor Messenger isn't completely secure. And then of course immediately I got tweets back saying, "You dimwit, that's how Tor works. That was you from Egypt." And I went, "Oh. Never mind." So that is one. And of course...

Steve: I love - that's a great anecdote, though.

Leo: Yeah, Facebook's not prepared for that, so Facebook said, well - and I just, I went - I wasn't very critical in my thinking, and I said, "Oh, no," and changed my password and all that. Yeah. It's what Tor does.

Steve: So, yeah. So what Tor Messenger is, which has just entered beta, is a combination of Tor for anonymity and OTR. That's the Off The Record chat protocol.

Leo: And that's - and I like that. I thought, that's cool.

Steve: Yes, it is. It's absolutely - it's public, publicly vetted. It's been checked several times. That's a win. So the combination makes a lot of sense. And Tor, I think, makes sense for chat because you're not trying to do web surfing, where, as we know, a web page goes out to 50 other domains and sucks stuff in from all over. You're just sending a couple hundred characters of chat back and forth.

Leo: It takes a little longer. No big deal.

Steve: Yeah, exactly, a little bit of latency in a chat doesn't matter. So anyway, I wanted to put it on everybody's radar. While it's in beta, it's like, okay, we'll just kind of keep an eye on it, and we'll see how it goes. But know that for some class of our listeners, they may want to know and may want to start playing with it.

Leo: And don't try to hook up your Facebook account.

Steve: And certainly Tor is well established, and OTR is well established. So, I mean, it's not like it's anything hugely new they've done. I imagine that the only problems they would have are things, for example, like needing to support two-factor, if you're trying to connect to that, and so forth.

Leo: And good on Facebook. I love it that they said, oh, somebody's logging in from Egypt. I don't think you're in Egypt, since you logged in this morning from somewhere, from Petaluma, so it must be a phony login. And from their point of view, it seemed like it was, sure.

Steve: Yeah.

Leo: I just feel like - I feel kind of dumb.

Steve: No, I love - that's a great, great anecdote.

Leo: "Don't you know how Tor works?" And I said, oh, yeah.

Steve: So we have two divergent directions which cyber products or cyber security are taking. In the U.S., the EFF Thursday posted the word "Victory," with the subhead, "State Department Decides Not to Classify 'Cyber Products' as 'Munitions.'" Now, okay.

Leo: I'm surprised that's still on the table.

Steve: I know. Aren't you? It's like, wait a minute. Have we not been here before? It's because crypto was once declared a munition that we're having problems today because weak crypto is still available in some browsers and servers which would never be there if it weren't deliberately crippled.

Leo: That's a good point. I didn't really - of course that's why the 20, whatever, 24-bit keys are out there.

Steve: Right, right.

Leo: And they call it "munitions" just to give it export restrictions. That's the point.

Steve: Precisely. Precisely. And so once upon a time, strong encryption was classified by the State Department as a munition, and you could not export it from the U.S. And so of course the problem was the Internet is global. And so the only solution that Netscape had when they were trying to do secure connections, because remember they did SSLv1, was, okay, 128-bit keys the State Department, of all things, says we cannot use on a connection leaving the country, but 40 bits we can. And that's where 40 bits came from, was that it was weak, weak...

Leo: Because it's crackable.

Steve: Yes. Because the NSA was, if they really needed to, could get into it. And back then, with computing resources and so forth, 40 was strong, but mostly it kept casual people from getting in; 128 is what you really wanted. I mean, 128 is even strong today still. So, okay. So anyway, so this little blurb is just two sentences, reads: "This week the U.S. Department of State's 'Defense Trade Advisory Group' (DTAG) met to decide whether to classify 'cyber products'" - and those are in quotes because it's sort of ill-defined - "whether to declassify 'cyber products' as munitions, placing them in the same export control regime as hand grenades and fighter planes. Thankfully, common sense won out," writes the EFF, "and DTAG recommended that 'cyber products' not be added to the control list." That's the good news. The bad news is that the U.K. hasn't gotten there yet.

Leo: No, of course not.

Steve: Now, I want to make it clear this is proposed and needs to go through Parliament yet, but is being strongly pushed by David Cameron. So the headline is "Internet Firms to Be Banned from Offering Unbreakable Encryption Under New Laws."

"Companies such as Apple, Google, and others will no longer be able to offer encryption so advanced that even they cannot decipher it when asked to under the" - this is called the "Investigatory Powers Bill. It will also require Internet companies to retain web browsing history of their customers" - meaning ISPs would be required to retain web browsing history - "for up to a year. The bill is expected to face a tough route through Parliament. But Mr. Cameron" - and I'm quoting from the Telegraph.co.uk.

"But Mr. Cameron urged critics to back the measures. He told ITV's This Morning, 'As Prime Minister, I would just say to people, please, let's not have a situation where we give terrorists, criminals, child abductors, safe places to communicate. It's not a safe space for them to communicate on a fixed-line telephone or a mobile phone. We shouldn't allow the Internet to be a safe space for them to communicate and do bad things.'"

And then Lord Carlile, who is the former terrorism laws watchdog, said there had been a "lot of demonization" of the police and security services over their intentions for such information. He was quoted saying: "I think it is absurd to suggest the police and the security services have a kind of casual desire to intrude on the privacy of the innocent. They have enough difficulty finding the guilty. No one has produced any evidence of casual curiosity on the part of the security services."

Leo: How about malicious curiosity? What about political curiosity? What about...

Steve: Yes, yes, exactly. And actually Edward Snowden did provide...

Leo: Right.

Steve: ...anecdotal evidence. Remember, he was talking about nude photos of people being...

Leo: Passed around, yeah.

Steve: ...passed around the NSA that they were sucking off the Internet from their taps. So, I mean, this is expected to have a tough rode through Parliament. But, I mean, I don't know what this means. If they were to pass this, then Apple and Google have a big decision to make. And then I don't know how ISPs can even honor web browsing history unless their certificates are forced into users' clients so that the unencrypted traffic goes through the ISP for logging. So, I mean, this has huge ramifications. I mean, maybe it just - it's like a nightmare in terms of practical implementation.

Leo: Apparently it's not - by the way, this debate is going on in the House of Lords, which makes it even more anachronistic and funny. Apparently it's not a done deal.

Steve: Good.

Leo: Lord Strasburger claimed the Prime Minister does not seem to get the need for strong encryption standards with no backdoor access. The lord said: "Cameron said three times he intends to ban any communication we cannot read. Will the Minister bring the Prime Minister up to speed with the realities of the digital world?" And then Lord Clement-Jones asked if she could absolutely - oh, my god, it goes on. And then there's the Baroness, Baroness Shields. There she is. Lovely lady, I'm sure. She says, "I can confirm there is no intention to do that; that is correct." So I don't know.

Steve: To do what? I mean, on one hand what they're saying is...

Leo: I know.

Steve: I mean, this bill says that we need to be able to see everything on the Internet.

Leo: I know.

Steve: There needs to be nothing that we can't see. And we're even going to make ISPs log their customers' browsing history for a year.

Leo: She said, the Baroness said she could absolutely confirm there's no intention in forthcoming legislation either to weaken encryption or provide backdoors. She said, even though he said it three times, the Prime Minister does not advocate banning encryption. Oh, banning encryption. Oh, well. I'm glad he doesn't advocate that.

Steve: Ah, okay. You can still have it, as long as we can, too.

Leo: He doesn't want to ban encryption. I don't know. You know, I think that part of the problem is the lords and ladies do not seem to understand what the hell they're talking about.

Steve: Well, and to our government and bureaucracy's credit, and frankly to the crypto industry in the U.S., who independently wrote several mass signed explanations and white papers and explainers and pleas, you know, to make it clear that what law enforcement wanted was not possible. Not just inconvenient, not possible. And so maybe, I hope that there are similar levels of activism over in the U.K., and that that will happen because, boy, I just don't know. I mean, you know, I don't know, like from a practical standpoint, what does that mean? You can't use a VPN?

Leo: Yeah.

Steve: Because a user using a VPN would send an encrypted tunnel past the ISP, making it impossible for the ISP to log what the user does. So what does that mean? Wow. Anyway, interesting times we live in.

Leo: Seems like it's the 18th Century, actually, with the baroness and the lord.

Steve: So back to business. We have another clever way of tracking users or their browsers. And the problem is that sophisticated systems, as we often say, are so difficult to secure. Of course there was the supercookie, where it didn't just use the cookies that your browser was sending to specific websites, but it looked at all of the headers and realized that things like the user-agent contained a long list of subassemblies and subsystems and their version numbers that your particular system had, and that that hugely narrowed down the number of people you might be.

And then you couple that with a few other things, like even the order of the headers. It turns out different makes of browser have their request headers, they emit the request headers, each of them in a unique sequence. So, oh, whoop, now we know, you know, that narrows it down again. And through a series of that you can end up really figuring out to some degree who someone is.

So that was the one example. Then, remember, there was the visited link hack, where because browsers are trying to help their users, they color the links which have been previously visited differently than they color the links that haven't been. Well, someone figured out, hey, that could be leveraged in order to probe a user to see what their browsing habits have been, by off the page putting up some links, letting the browser color them, and then checking the color. So it was like, whoops, that was something we didn't anticipate.

So now we have another one. And this is an HSTS cache state hack. So, okay, what is that? HSTS, we've talked about, is the HTTP Strict Transport Secrecy, or Strict Transport Security. Now, that's the thing that, for example, GRC does, and Google does, and other security-conscious sites do, where the entire site is committed to being over TLS, only accessible via HTTPS, only accessible securely. So every response that those sites like mine and others give includes an HSTS header, a Strict Transport Security header, saying absolutely never try to access this site non-securely. And if you encounter an HTTP URL to this domain, you have our permission to silently upgrade it to HTTPS.

Now, the reason for this is there was one little hole in the security of a site. And that is the first page. And we've talked about this in the past. If, for example, you are in a public WiFi setting, and you first went to http://amazon.com, then that first page would be over HTTP. When it came back to you, all of the links might be converted to HTTPS, or your browser might even be redirected by that site to HTTPS, so that it's trying to take you, the remote server is trying to say, oh, you came in nonsecure. Let's move you over to secure. Or, you know, buttons on a form might be HTTPS.

The point is, because it was coming back in the clear, a man in the middle could strip out the S's and keep the site from switching to secure. Then the user, just assuming that the security of his login has been taken care of by the site, as we all assume, types in his username and password, which the bad guy then gets because the form has had the S stripped out of its submission URL, so it goes username and password in the clear. And now we have a problem. Now your identity for that site has been lost.

So the problem here is that first query. And so the HSTS solution was a fix for that first query problem because the idea being every single contact with the site sends back only - you, the browser, we're giving you permission, upgrade this to HTTPS. That way - oh, and I forgot to mention there's an expiration, which is typically way in the future, you know, like a year in the future. So not only does every reply to the browser reinforce that, but the browser caches that.

Okay. A clever person figured out how to probe the cache in a browser, that is, how to determine which sites a user has been to. And with a large enough set of probes, with a very high degree, you can again identify a user. So this thing is called Sniffly. If anyone wants to dig deeper into it, I've got the various links in the show notes. But the concept is that you visit a site that presents you - so you visit a site. Your browser says, "Give me this page." The page contains a whole bunch of accesses to HTTP images, that is, nonsecure images. And it also leverages something called the Content Security Policy.

The Content Security Policy is another of sort of the evolving moves to further strengthen the whole client-server web interaction. So what it does is it's able to - it specifies rules which the browser must follow for its queries. And normally you're saying, like for example, a typical content security policy would be only access JavaScript, so like *.js, over HTTPS. So it would prevent the browser from ever following a nonsecure JavaScript link. So that's an example.

Well, similarly, it's possible to use content security policy to say only access images over HTTP. So what that does is it blocks the browser from sending an HTTPS query for an image. Then when the probing site sends a bunch of image requests to known HSTS sites, the state of the browser's HSTS cache will upgrade those that are in the cache to HTTPS. The problem is that that runs, then, against the content security policy, which blocks the loading of images other than HTTP.

So what this all boils down to is JavaScript that's also running on the page is timing the failure of the image fetch. It will either be blocked instantly, in, like, less than a microsecond, which says that it has been blocked because that browser has an HSTS entry in its cache which upgraded that query, which has then been blocked; or, if it doesn't have an HSTS entry, the query will be made out over the Internet, which takes many tens of milliseconds.

And so what this all boils down to is simply by timing the length of time it takes for the query to fail, it's possible to sense the presence of an item in the browser's HSTS cache. And in fact there's a test page that I have that is on that Sniffly link that comes up and shows you where you have been and where you have probably not been. And as I looked at it, it's like, oh, yeah, it's pretty right about that. So an interesting hack.

Now, I mean, it's not something to get all worked up about. For one thing, it only works on HSTS-supporting browsers. There are some other caveats, too. It's not supported yet in Safari, IE, or Chrome on iOS. If you had the HTTPS Everywhere extension, remember what that does is that auto-promotes all queries. It tries them secure first, and then falls back if that fails. Well, that would defeat this completely. That would mess up the results. It doesn't work reliably over the Tor browser, just because of the latency. It's unable to properly make the differentiation one way or the other. And now, so Leo, there it is, there's the results. On the left-hand side...

Leo: This probably could tell me where I probably haven't been.

Steve: Right, right. Because what it's done is it sent probes to those known HSTS servers, but it's detected that you don't, your browser doesn't have an HSTS from that site, so you probably haven't been there.

Leo: I don't know if I've been to all of these sites it's telling me I've been to. Is this just for this machine and this browser?

Steve: Yes, that one instance of that browser. And it does seem wrong. I mean, I notice that...

Leo: I don't even know what some of these sites are. QLinks.com? Attracta.com? I don't think I've ever been to those.

Steve: Right, so it's not...

Leo: Might not be perfect.

Steve: I'm not all worked up over it. It was an interesting proof-of-concept.

Leo: A lot of these are right, but there's a few I don't - Gumroad.com? Have I ever been there? I don't think so.

JOHN: Could those be, like, ad networks?

Leo: Could they be ad networks? No, Gumroad is sell your work directly to your audience. I've never been to this site.

Steve: That is a good point, though, whoever it was in the background who suggested...

Leo: The ad networks, yeah, that's John, yeah.

Steve: Yes, because, yeah, exactly, your browser making third-party queries, it will be acquiring those, too. So you didn't go there, but it went behind your back.

Leo: Right.

Steve: And we know how much of that goes on.

Leo: Yeah.

Steve: So ECMAScript 6. This is the first major update to JavaScript, as we conventionally and conveniently call it, since '09. So it's been six years since we've had so-called ES, or ECMAScript 5, or the current version of JavaScript. The ECMAScript 6 was ratified in June of 2015, so a few months ago. Browsers are moving forward. There's a cool link down on my show notes, Leo, you might want to bring it up with yours, under that transpilation, as it's called. It's translation and compilation. "Transpilation" is the term that they use.

And, now, the far left column is it's just probed your browser. So it says "current browser." And so what we're seeing there is varying shades of green or red for the levels of support of these new features in various industry standard browsers. And by the way, there's one almost all green column. I was very impressed. It's Edge. Edge is the leading browser in the industry, I mean, compared to Chrome, which has been a standards leader, and Firefox. And of course IE is down in the weeds somewhere. But that's an almost all green column there, as I'm pointing to the screen. It's like, that's not going to help. It's Edge. And Safari does a fair job.

Okay. So what's new? I'm not going to go into a comprehensive enumeration of features because anybody who really cares will care about more detail than it's going to make sense for me to give. But, for example, in JavaScript, unless you declare a variable, it is way global. If you just use a variable without declaring it, it's known to everybody everywhere in the running script. The normal way of limiting that is to use the VAR, V-A-R, verb. So you'd say VAR X equals something. And that, the so-called "scope," the scope of code where that's known is limited to the function where that variable was declared with VAR. You can explicitly declare global variables by using VAR outside of any function, in which case everybody gets to know about it. The new edition is the word "let," which harkens back to BASIC, where you had let x equal 2.

Leo: And LISP.

Steve: Yup, yup. The very old languages, they use "let." Well, what "let" does is very handy because it allows for local block scoping.

Leo: Right.

Steve: Meaning that, like, within curly braces, or just within the containing block, that...

Leo: In the LISP world we have a saying: What happens within let, stays within let.

Steve: Right.

Leo: So only stuff within the let block can see that variable.

Steve: Right.

Leo: Which means we can duplicate a variable, and with no harm, which is nice, yeah.

Steve: Correct, correct. And in fact that's actually where it comes in handy. Oftentimes programmers, because they don't have very tightly locally scoped variables, they're having to use, like, IJKLMN, even in other areas of their code, whereas it'd be cleaner just to, like, reuse "I" if you had containment of your use of "I."

Leo: That's local. I mean, I think most languages handle this in some way. The problem is JavaScript, frankly.

Steve: Well, what's happening is, yeah, what...

Leo: Scoping in JavaScript's terrible.

Steve: If we were - it is, it's awful. But if we were to sort of stand back and take a 10,000-foot view of this six-year evolution, what we're seeing is bits and pieces are being pulled in from an array of existing languages. So for the first time there's an enforced constant declaration, where you can declare a variable to be a constant, and you are unable to change it. That hasn't existed before. It was up to you not to change it, but nothing enforced its fixed value. We get a bunch of handy new methods, you know, operations for the math string and array objects. There's now a support for default function parameters, which a lot of other languages have had, the idea being that, in the function declaration, you could say, okay, I want this function to take values X, Y, and Z.

But if the caller doesn't specify them, then let's have Y be 12 and Z be hello.

And so it was, in JavaScript, until now, it was awkward to do that. You'd have to say this function takes X, Y, and Z; and then, in code, you'd have to check to see if Y was undefined, and if Z was undefined. And if they were equal to the undefined value, then you'd set them to your default, which is, I mean, it does the job, but it's awful and cumbersome. And so they said, hey, you know, other languages have default values for function parameters. Let's put them in JavaScript. So now JavaScript gets that. Which is nice.

It also gets formal support for modules, which is a huge win. There's been no explicit module containment. The idea, the concept of a module is that the stuff inside isn't public. But you're able to export explicit things that you want to make public. Otherwise the work you're doing inside is private, and then you're able to import, from other code, functions that that code has exported for your use. So there's like more you have to do. You don't have to do this. So it's not like you're required to do it.

So JavaScript remains easy to use. But the problem is ease of use begins to collide with doing big things. And people are increasingly wanting to do big things. And so, for example, if your project has a whole bunch of different code, pulling in from different directions, and the variables are all global, if any of that code uses the same variable name, they clobber each other. So there's all kinds of problems as things get bigger, which people have come up with clever solutions for. There's something called "closures" in JavaScript which is a really bizarre way of encapsulating everything in a big function in order to use the functions locality to extend to all of the code. But it's just - it's really cumbersome. So these are additional nice extensions.

Also, from classic object-oriented languages, we get a formal object constructor and methods. So user-defined objects can have a constructor which is invoked when a new instance of that object is created. And then it's able to have contained methods which that object can export. And those are now formal parts of the language. So you create an instance using the new verb, and then that runs construction code of your own design. And you have now inheritance from classes. So we actually have formal classes in JavaScript 6, or ECMAScript 6. And you are able to reach back and access things in the base class.

So what's happening is current browser support is spotty, yet many companies are wanting to start using these features. So there are a number of so-called "transpilers," which translate and compile a single, not yet natively supported, JavaScript with an increasing number of these new features back into standards-compliant lower-level JavaScript that is supported across browser. The browser vendors are moving fast. As I said, Edge is like really compliant already. It was nice to see Microsoft ahead of all others in some significant and useful degree.

So anyway, I just thought I'd take a moment to mention this because it is the language of the 'Net. And interestingly, boy, if you look at the hiring boards, if you are a proficient JavaScript coder, that's now the number one language that companies are wanting in order to write web-based apps for the 'Net.

Threema got an audit. I've talked about Threema off and on for a couple years, I guess. I liked them from the first moment because they weren't free. You had to pay them a few dollars. In return, we understood their whole economic model. The other thing I liked about them is that their model was simple. Remember, these are the ones where you had, like, three levels. You had yellow, I don't remember now, red, yellow, and green, where - like a level of confidence in the identity of the party to whom you were communicating because the ultimately greatest level was using QR codes to let each of your clients see the other's private key. Or, I'm sorry, see the other's public key. The private key never leaves the client.

And so basically it's a very simple public key chat system where they don't manage, like transparently manage user identity the way iMessage does. As we've discussed, iMessage is secure from Apple as long as there's no games being played with the public keys. But games can be played with the public keys that would go unnoticed. So Threema stays out of that. They put the burden on the user, which is why I've said, if you absolutely really care, I mean, really care about security, this is something that you'd want to use.

So they've been audited by an independent Swiss IT research lab that stated in rather stiff terms: "We confirm the quality of the system as claimed by Threema in their public specification." And, they said, "Two of Threema's main promises are the whole communication - including group chats, media files, and status messages - is end-to-end encrypted." And "Threema is designed to limit users' back track" - I'm sorry - "designed to limit users' data tracking to a bare minimum."

And the audit found that both these assertions were confirmed. Threema made their source code available, their servers auditable, and their developer team provided for any assistance that was needed. So it was a complete, here's everything we're doing. Audit us as deeply as you can.

And so in their report, the auditing agency said, "Threema's concepts meet the requirements for truly secure and trustworthy messaging. The application of the encryption is correct and implemented as documented by Threema. The used protocols are free of known vulnerabilities. The app's local data is stored in a safe and secure manner. The server components only store data that is absolutely necessary for message delivery." So minimal data storage. And the servers are located in Switzerland.

So if anyone's interested, I have a link in the show notes of the PDF of the entire audit. But it passes. So it was nice to have that. Threema has a published security document that absolutely details every aspect of their protocol in, like, full specification. And I did read through it a few months back, just because I was curious, and it looked as good as I could hope it would. And they do have some new stuff for their iOS client. It is now possible to use Threema to send any type of file - PDF, animated GIF, MP3 audio, a DOC, a ZIP, et cetera, up to 20MB. Group chat now supports up to 30 members. And then they also have a bunch of other improvements and bug fixes and Android support, as well. So good cross-platform support and lots of features. So I did just want to mention it made the audit.

Okay. Quick, some miscellaneous stuff. NASA. I got a bunch of tweets from people who picked up on this, that was covered in a bunch of popular press, saying that NASA was looking for 60-year-old engineers who still remember FORTRAN and assembly language. And of course we know why our listeners sent this to me.

It turns out that Larry Zottarelli, who is the last original Voyager engineer still on the project, is retiring after a long and storied history at JPL. While there are still a few hands around who worked on the original project, the job of keeping what has become the Voyager spacecraft, which has become an interstellar spacecraft, going will now fall to someone else. And that someone else needs to have some very specific skills. And I got a kick out of the fact that it was like, "Warning: You will only have 64K of memory." It's like, wow. You know, that's hard to use. Hard to use that much memory, you know? SpinRite for the first three versions was a COM file because it was less than 64K and did everything that SpinRite did with interleave optimization and everything else.

So anyway, I got a kick out of that. It is a problem. We've seen this before. The shuttle was having this problem, and that is that the shuttle engineers were retiring, yet the shuttle itself hadn't yet been retired. And so it was just sort of a brain drain. We were losing the people who really knew how this stuff worked.

And a bunch of people sent to me, and I wanted to thank everybody, the notice that we are finally getting a new Star Trek series on CBS. It won't be premiering for a little over a year. It'll be January 2017. But I'm hopeful. Its executive producer is Alex Kurtzman. And of course he's well affiliated with Star Trek. He did the first of the refranchised Star Trek movie in 2009. He also did "Into Darkness." He did "Spider-Man 2." He was the executive producer of "Fringe" and "MI3." And also I just, in reminding myself what he did, I saw that he did "Cowboys & Aliens." And I want to say, if anybody missed that, it is a fun movie. "Cowboys & Aliens." If you didn't see it, it's quirky, but it's a solid piece of work.

And anyway, so I know nothing about what the new series will be. But I've been lamenting that we don't have any really good science fiction on mainstream broadcast television. CBS is going to bring that to us. Okay. Now, Leo?

Leo: Yes?

Steve: "Fargo."

Leo: Oh, the movie or the TV show?

Steve: It is the best series on television.

Leo: Oh, wow.

Steve: I watch everything that is, like, at the top. I mean, I've been watching "The Good Wife" since the beginning. I like "Madam Secretary." "Fargo" is in its second season, and it is arguably best than the first season that was really good. Now, I don't think I have ever seen a 9.0 on IMDB.

Leo: Wow.

Steve: But that's what it has. I want to warn people, it's not for everyone. If anyone saw the movie, you'll remember the scene with the wood chipper.

Leo: The wood chipper, yes.

Steve: Yes. I mean, that's like, you know, "Godfather" with the horse in the bed, or what's another classic one? You know, there are scenes that just live in movie infamy. Maybe the shower scene with the knife in "Psycho."

Leo: "Psycho," yeah.

Steve: But, okay. So this is not like that. But it's a little Tarantino-esque. But I just want to say, if you don't mind a little bit over-the-top gore, you know, like explicit murders and so forth, oh, my god, the writing is just titillating. The acting is superb. You probably want the first - the two seasons are not related. You don't have to watch the first season. Second season is - I think it's at three or four episodes now. But I just watch it - and it's on FX. I just watch it, and I'm just mesmerized by the - it's like art. It is television art. It's what there just isn't enough of. And so I thought, okay, it's not science fiction, but I just had to say to our audience, you know, again, with the caveat that it may not be your cup of tea. But if it is, you will be so glad you found it. And you've got a whole back season you could just overdose on, binge watch, and then the second season that is arguably better than the first.

Leo: So the reset - so this is one of those series that reboots each season with new story, new cast.

Steve: Correct, correct.

Leo: And the reboot was even better, you're saying. Which is kind of cool.

Steve: Yes, yes. I had mentioned to Jenny, who's watching it, and she said, you know, she reads extensively, like all the popular media, The New York Times and New Yorker, the L.A. Times and so forth. And she said she had seen the comments that the second season is even better. And I agree. First one, first season I loved. Second one, wow. It's just...

Leo: We'll start watching it. So you don't need to have seen the first season at all.

Steve: No, no. And so you could watch some of the second season to get a feel for it. Then you could watch the first season. You might kind of get them confused if you were interleaving them.

Leo: Oh, okay, all right.

Steve: But you do need to start at the beginning of a season, whichever one you choose, because they're definitely serially dependent. But the seasons are independent. And, oh, Leo, you will, I mean, it's art. I mean, it's just it's like the best television.

And I just wanted to make a quick mention, I 100% agree with you about Chromebook. I've had two experiences with it now, and I am very impressed with what it is. I agree with you. I've heard you say it now a number of times. For most people, it's all they need. And you turn it on, and it boots in 15 seconds, or maybe 10. I mean, it's just on. And it does everything most people need. I gave one - and I like the Toshiba Chromebook. It's a little over 300 bucks on Amazon.

Leo: Dell has a new midrange one that everybody's raving about. So I have to check that out.

Steve: How much, do you know?

Leo: I don't know. Let me look at it.

Steve: Because their hardware's pretty good, too.

Leo: Yeah, I mean, that's one of the unfortunate things is that Chrome OS is often stuck on essentially a Netbook so they can keep the price way down.

Steve: Right.

Leo: But Dell now has a 13-inch kind of business class Chromebook. I have a Pixel, the Google high-end. And, man, that's a wonderful computer with touchscreen and everything. You know, I'd love you to do a survey at some point of the security techniques they use.

Steve: In Chrome OS?

Leo: In Chrome OS.

Steve: Yeah.

Leo: Because it's really interesting, I think.

Steve: I agree.

Leo: $429, so it's a little more for the - but it's 13 inches. It's an HD display. And it's probably a little bit better hardware. In fact, I know it is. I've heard people talk.

Steve: For what it's worth, then, Toshiba is a 13-inch. There's been a refresh of it that uses a faster processor.

Leo: Oh, good.

Steve: And I saw some reviews that really liked it. I think it's maybe $329 or something on Amazon.

Leo: My experience with these is, I mean, for instance, this has got an i3 or an i5 processor. It's got 8GB of RAM. The more hardware you throw at it, the faster and more fluid these are. But boy, they've just - I think they're just great. And the security is good.

Steve: Well, and, yeah, it is, it is. I mean, it's not going to get infected. It's, you know, in the case of one person, she just needed to do cloud-based accounting with QuickBooks on the 'Net, and email. And it's like, this is all you need. And, by the way, if you step on it, I'll get you another one.

Leo: Right, right.

Steve: You know, it won't be the end of our life. So, yeah, I just - I kept seeing you recommend it, and I wanted to make a note to say, yup, I agree completely. And Mark Sidell tweeted me the news that the boiled frog analogy I've been using is a myth.

Leo: I had heard that, as well, that the frog actually does jump out after a while.

Steve: Exactly. So just, you know, under errata, I wanted to correct the record.

Leo: Who's going to test that theory?

Steve: Yeah. Now, I need another analogy, though, because it's a fabulous one.

Leo: Oh, yeah.

Steve: And we need something else like that, like - I don't have one. But I am putting out the call for a replacement for the boiled frog myth. TheAtlantic.com had an article backdated in 2006 saying: "Everyone who has heard a political speech knows this story: You put a frog into a pot of boiling water, and it jumps right out. But if you put it in a pot of nice comfortable water and then turn on the heat, the frog will complacently let himself be boiled. One standard version of the story is here. The reason it's so popular in politics is that it's an easy way to warn about the slow erosion of liberties or any other slow threat you want to talk about." Of course that's the way I love to use the analogy.

"Here's the problem," writes the Atlantic. "It just isn't true. If you throw a frog into a pot of boiling water, it will, unfortunately, be hurt pretty badly before it manages to get out, if it can at all. And if you put it into a pot of tepid water and then turn on the heat, it will scramble out as soon as it gets uncomfortably warm." Yeah, well, that's what we would do if we jumped into a Jacuzzi, too. It's like, ow, out we go. Or ow, out, as soon as it gets too hot. So, sorry about my use of a bogus analogy. I wait for someone to propose a replacement because, boy, it sure is great. Too bad it's not true. And, Mark, thank you for bringing it to my attention. And our listeners'.

And I'll just make a brief SpinRite note. In another tweet, Mike, spelled M-A-I-K - Maik?

Maik. Probably Maik Musall, M-U-S-A-L-L, he said: "SpinRite Q for SN." And this is one we get so often, I just wanted to cover it briefly. "After fixing a faulty disk, is it wise to replace it ASAP, or is it as good as a non-faulty one afterwards?" And of course that's what people want to know because they had a problem. SpinRite brought it back to life. Everything seems fine now. Now what? So the problem is there just isn't a true answer. So you sort of have to use your intuition, but that can be aided by SpinRite's SMART screen. And I've talked about that a couple times.

And in fact there is documentation on GRC about SpinRite's SMART screen. I have a couple pages where I show some examples so that people can - just sort of to help people interpret what it's showing. But the idea is that SpinRite fixes the drive so that it's okay. But it could be that it is dying, and so you've brought it back to life, but it's still going to continue the process of dying. Or it could be that, like, for example, the system got bumped when it was writing, and that moved the heads slightly off track so that that sector could not be easily read, and it took SpinRite to come along and fix it. So there was nothing wrong with the drive. There was just an instance of an event that SpinRite fixed.

The point is that, in general, the SMART data is very good at showing if the drive's just having overall trouble. The problem with SMART by itself is that SMART doesn't know that there's trouble unless trouble is actually happening. So the beauty of SpinRite and SMART together is that SpinRite runs the drive through its paces so that the SMART data has an opportunity to pick up the fact that there's trouble happening, if there is. And then it will show you. You get some red bars sort of descending, or actually it's sort of, I guess, cyan bars pulling back to reveal red. And the more red that's there, the more trouble the drive is having. And you probably shouldn't have any.

So the answer is, if you kind of think you know why there was a problem, and SpinRite fixes it, you're probably still okay. But check out that SMART screen. It's one of the six or seven that you can rotate among when SpinRite's running. And after it's been running for a while, flip the screens over and see if you see any red. And that'll give you a good clue to whether, eh, this drive, once we're done, it's probably a good idea to image it and move it, move your data somewhere safer.

Leo: That's a good tip. I didn't know that. That's good. I'll have to remember that. Because a lot of times you can get this false sense of confidence. Well, I was able to SpinRite it, and everything's good.

Steve: Right, right. And people do. People just say, yeah, this thing's kept my dying drive alive for 20 years. It's like, okay, you know, they are so cheap now that it's like maybe a good time to, while you're ahead.

Leo: All right. Let's talk about Apple. Not such a success story.

Steve: So this is sad, unfortunately. I didn't want to be sad about my platform of choice. But, okay. So as I promised, I dug much deeper into this whole issue that I'd never looked at closely of vetting or verifying the behavior of apps. And, you know, we've covered, a number of times through the years, there's been news of apps somehow sneaking through or getting past or doing something undesirable, despite Apple's best efforts.

And of course just last week, when I promised to give this deeper coverage, we talked about that Chinese advertiser who produced an SDK, a software development kit, which thousands of other Chinese companies incorporated into their own code in order to get the access to that advertising network; and how, in an analysis of it, it was clear that the Chinese bad actor had sort of been creeping forward their boldness of access to banned private APIs, until they finally got caught.

So, okay. What is this whole idea? So what I came to understand, after I really got a grip on the whole structure, is why Apple has this problem; and, unfortunately, why it has no true solution. They have the problem because the original architecture of iOS assumed fully trusted applications. And it assumed fully trusted applications because the original concept of the iPhone was here's the apps from Apple that do messaging and mail and surfing and clock and calendar and so forth, and that's it. And, I mean, when we look back on it, it's like hard to imagine, like that home screen that wasn't even full. I mean, remember like the first three lines, and then maybe two more icons or something, you know.

Leo: Well, you were supposed to be able to add apps as web apps. So you'd go to the - you could create a web app. And then you'd go to it in Safari, and you'd bookmark it on your home screen. And Steve said that's all anybody would ever need to do because HTML5 apps are so powerful. We don't, we just don't need to do an App Store.

Steve: Right. And HTML5 apps are so weak, meaning that, like...

Leo: Oh, yes, right.

Steve: They're powerful enough...

Leo: They're harmless.

Steve: Right, exactly. They're harmless. So Safari does operate in a constrained environment. It's well sandboxed, and it's actually one exception to the apps. Normally, no iOS apps are able to do something called "dynamic code generation." That is, they're not able to execute data. But Safari is an exception. It has the dynamic code generation privilege because of its just-in-time compiler. So its JIT, J-I-T, the just-in-time JavaScript compiler is able, as it processes JavaScript, that compiler produces code, but it's data coming out of the compiler. So it must have this privilege to essentially have the data that it generated execute.

Well, that's super dangerous because, I mean, in an environment where you have the expectation of scanning someone's code for malicious intent, which is itself difficult, the idea that an app could rewrite itself from runtime, when it's running, it could dynamically rewrite itself. And that's the difference between static and dynamic code analysis. Static analysis means that you're running a computer over the object code, you know, over the code, but you're not stepping like a pseudo program counter through it to, like, follow its nooks and crannies and jumps and things. You're just standing back and looking at it. So there's static analysis and dynamic.

Okay. So the point is that Apple originally conceived of this architecture under the assumption that the app, the native apps running on the phone were from them. Meaning they're not going to misbehave. And so those apps had unrestricted access to everything. They could do anything they wanted because essentially they were iconized pieces of the operating system. You know, they did run separately. They looked separate. But there was no clear delineation. Then, with this incredible success of the phone, anybody who is our age, Leo, or maybe any age, because it's not been that long, will remember the huge pressure for third-party apps. You know, it's like, hey, yeah, it's nice, but it's closed. When are we going to get third-party apps? That's all, I mean, we love the phone, but everybody wanted third-party apps.

So now Apple was in trouble. They had an architecture that had not foreseen - and that's the key, it had not foreseen the need for third-party apps at all. So, and as we'll remember, it took them a while, as a consequence of the fact that they weren't ready to do this. So what they did, unfortunately, was on one hand the best job they could, but they couldn't do a good job. The way the system was built prevented them from doing a good job. We talked a little bit about this last week, but now I have all the details.

And there are different development platforms for the iPhone. For example, you can write apps in standard C. An app in standard C can be enforced because a static analysis can see the functions that that C program calls. So you can do a static analysis of a straight C program. That's enforceable.

The problem is that the popular means of developing apps is Objective-C. And Objective-C, as we mentioned briefly last week, instantiates or calls methods in object libraries via sending a message, essentially a text message. It actually is. You send a message to the function in the library where a string is the first function parameter, telling it what you want to do, and then there are exceeding parameters for that function. That's the way methods are invoked, the way you get things done in Objective-C. In the object that you call is a dictionary, literally a list of the names of all of the functions, and then pointers to the code that implements the function.

So your string is looked up in the dictionary for the starting location, the starting address of the function, and then the system jumps there to execute it. So it is unfortunately that system which is still the majority, you know, you can write apps in C. Most of them are in Objective-C because that was sort of the officially supported development platform. They're moving toward Swift, but Objective-C is still where we largely are today.

So the problem is Apple understood that, again, because of the legacy, the available functions were too powerful. There was, like, get hardware ID. Get user ID. Get list of installed apps. You know, get anything you want, basically. I mean, and like I said, there was no strong app/OS boundary. The apps had the run of the place. And so suddenly they had to restrict apps so that they couldn't have the run of the place.

So what they did was they went through the list of all of these strings which an app should not use, and they essentially removed them from the header files. They undocumented them. I'm not kidding. So the headers that the developers got didn't list the functions that they should not use. But they were still there. And the reason they had to be there is that oftentimes functions that they should use relied upon the functions that they should not directly use because the function they should use could be trusted to use the function they should not use in a safe way.

So the idea was there was sort of a - there was a barrier between the application and the off-limits functions of good functions, trustworthy functions, basically watered down sort of app-facing functions that didn't give any dangerous power to the application. And those were documented. But those functions that were running in that same user process had to have permission to access the unknown functions, the undocumented functions behind them that also had to run in the same user process space.

So, but my point of that is you couldn't use process boundary rights where, like, the special functions had different rights privileges than the safe functions because the safe functions themselves, although the apps weren't supposed to call the dangerous functions directly, the safe functions had to be able to. So they got themselves painted into a corner. Invariably, developers reverse-engineered the safe functions to find out how they worked. How was this safe function doing what it was doing because, in order to do what it did, it would have to have knowledge that Apple wasn't telling apps they could have. But the functions they were calling had to have that knowledge. So that means you reverse-engineer the function to find out how it works.

And lo and behold, developers discovered that there were strings that were being called that weren't in the headers. And so they added those strings to their own custom headers, and now they could call them, too. Ah, but Apple stopped them because what Apple can do is do a string search through the code.

Leo: That's pretty weak.

Steve: I know. It is horribly weak.

Leo: All you have to do is ROT13 or something, I mean...

Steve: Exactly. Just, exactly, change the case, you know, and then the string won't match. And then, before you use it, switch the case back to what it's supposed to be, and it runs. And so, and then so Apple started saying, oh, naughty, naughty, naughty, we caught you using a function that isn't in the headers. And you're not supposed to do that, so you're rejected. So developers then had a choice. Okay, we play by the rules; or off-the-books developers say, eh, you know, if we're calling a function by a string, we'll build it at runtime. Then a static analysis won't show it.

And in fact the people who wrote this paper, their whole trick is to push this sad state of affairs further towards a better state of affairs, but you can't do a perfect job. And in their own paper they comment that their system attempts to be a dynamic runtime analyzer. That is, it simulates user actions, pushing buttons. It follows the code paths. It essentially sort of is like running the app in sort of a fuzzing way, like doing all the possible things it can think of, and trying to get the app to do something naughty, which then it would catch.

And the problem is, and as they say in their paper, it's possible for there to be a complex interlock of sequences that only then gets code to do something in a tricky way. It might even be the app going out to a remote server and receiving a certain token. And then the token provides a value that is used to jump to somewhere that you just - that no dynamic analysis could catch. So my point is this is fundamentally broken. It's, I mean, Apple's doing what they can.

Leo: Isn't there some mechanism you could use, I mean, just not documenting a call doesn't obviously stop somebody from using it. Can't you somehow limit who can use those calls, some sort of mechanism?

Steve: Yeah. See, but the problem is that even the good code needs to use the privileged code to do what it needs to do.

Leo: Right.

Steve: And so if you...

Leo: Why doesn't Windows have this problem? Why, I mean, I guess there's no way to - just merely undocumenting it, not documenting it, doesn't mean you can hide it. So there are undocumented calls in Windows that people use; right?

Steve: Yes, I was going to say, yes. Windows has a long history of undocumented calls. They were things that Microsoft - they were never things that would pose a security risk, though. At no point was it like, oh, my god, I found this undocumented call that does something amazing.

Leo: Right, right.

Steve: It was like, okay, I can draw a line myself, rather than having it draw a rectangle.

Leo: Right, right.

Steve: Okay, big whoop. But here, these really are things, just due to the heritage, that originally were available, and still are, to privileged apps. But the problem is, the privileged libraries need to use them, too.

Leo: Right.

Steve: And there just isn't a way of drawing a boundary, or Apple would have. Instead they're just, you know, they're screwed until Objective-C is no longer supported.

Leo: Why would a different language make that better?

Steve: Because what they could then do is enforce static binding.

Leo: Ah.

Steve: "Binding" is the technical term. This is dynamic binding.

Leo: I get it.

Steve: Where you bind with a string, and you design the string at runtime. But a different language, like C, C doesn't have anything like this. It's statically bound. So there's a list of every API that that program can call. And the fact that it cannot generate dynamic code means it can't change that. So there's nothing it can call that isn't in that list. So Apple vets it and signs it and never has to worry about its behavior. But Objective-C, because of this dynamic binding with strings, there's just no way to be sure.

Leo: I'm thinking that's why they're pushing Swift.

Steve: Yes, exactly.

Leo: So Swift does not allow that. Is that right?

Steve: Correct.

Leo: Swift doesn't allow dynamic binding.

Steve: Correct, it doesn't have this problem. But basically calls being in a list, in a dictionary.

Leo: This is why we like functional languages.

Steve: Yeah, this is just - and I'm just, as an assembly language coder, the idea, even the idea of evoking a function by sending a message to somebody with a name that you want in the parameter, it's like, does that even run? I mean, I'm surprised it gets up in the morning. Unbelievable.

Leo: Well, you run something effectively by jumping to that area of code; right?

Steve: Exactly, yeah, exactly. You are doing just an immediate jump there. This thing, you're sending a message to a message dispatcher that then sends the message to the object, and in the first parameter is the name of what you wanted to do. And then it has to scan through a dictionary to find a string match, and then it jumps. It's like, I mean, and of course what that does is it gives you all kinds of flexibility. You can put hooks anywhere you want. You can over, I mean, this is the way superclassing works. You want to superclass a function or subclass a function. You're able to go in and, like, put your own function in for the...

Leo: You overload it, yeah.

Steve: Exactly. Exactly. And add functions, I mean, so there's that level of indirection creates tremendous opportunity and power.

Leo: Sure, sure.

Steve: But the tradeoff is this, is that it just does not handle well in a high-threat environment like a mobile platform where you're going to have, what is it, I read it's more than a billion apps now. Oh, no, no, more than a billion phones.

Leo: Yeah. More than a billion phones. More than a million apps, though.

Steve: Right.

Leo: And I guess, yeah, I mean, it's foolish to think - and when I say "undocumented," I mean, one way you document these things is by putting them in the headers. But there's other ways to, you know. But just not documenting it, by not exposing it, you have a secret document at Apple that says how to do it.

Steve: Right.

Leo: And there's no way they could look and say, oh well, this is being called by approved library. Oh, this is being called by a third-party app. They can't verify. Well, it would add a lot of handling overhead; right?

Steve: Yeah. So the problem is that the approved library, to do what it needs to do, has to call secret functions.

Leo: No, I understand that. But that could say, well, this is an approved library calling it, as opposed to my app.

Steve: Yeah. And the problem is that that would require, well, I guess it would require a level of per-call auditing...

Leo: Just a lot of overhead.

Steve: ...that they, I mean, it must be infeasible, or they would have done it.

Leo: Right.

Steve: Instead, what we know they're doing is just scanning the app for functions you're not supposed to call. And we've seen proof now that all you have to do is build those function names dynamically, and the vetting process doesn't see them. And now we know why.

Leo: Yeah. Wow.

Steve: I know.

Leo: It's interesting. You know, it seems like somebody should have observed that was a potential problem way back when.

Steve: Well, it's actually why I called it a scam. The whole, I mean, yes...

Leo: They were just hoping nobody would notice.

Steve: They were hoping this would be good enough; and that, like, removing things that were found to be misbehaving after the fact is then the best they can do. Somebody reports the behavior, or it's observed, and then they take it out. But it is a very weak filter.

Leo: Objective-C goes back to NeXTSTEP, which of course OS X is based on.

Steve: Right. In fact, there's a table called the NIB Table that's part of this. And that's the next interface builder or something.

Leo: And all the function names are NS something which stands for NeXSTEP.

Steve: Yup. Yup.

Leo: So this goes back to the NeXT computer. And of course in the mid-'90s nobody was thinking about this.

Steve: No.

Leo: In fact, everybody thought this was a great thing. Dynamic message, dynamic, you know, function calling is a great thing.

Steve: Yes, very powerful, yes. And as long as it was only your apps...

Leo: Right, who cares?

Steve: ...there's no problem.

Leo: Right.

Steve: And so that's what they did. And then the world made them allow third-party apps. And that's the problem.

Leo: So at that point, somebody in Apple, probably Avie Tevanian, stood up and said, you know, Steve, if you do this, you're going to have a problem. It's amazing they've gone eight years...

Steve: It is.

Leo: ...before anybody took advantage of it.

Steve: It is. You know, I mean, there have been anecdotal reports, and we've covered them. There have been apps that have been caught doing misbehaving things. And this is how they were doing it. But now it's, I mean, now everyone knows. And so, I don't know. Apple, I hope, I mean, either they retire Objective-C, they implement some better way of catching this dynamic behavior, or really spend some time increasing, you know, doing a dynamic analyzer like this 13-page research paper demonstrated. They were able to capture thousands of apps. They scanned thousands of apps and found hundreds that were using off-the-books function calls.

Leo: Wow. Now, Android is written in Java. Does Java have the same dynamic calling? I think not. I think Objective-C was very advanced for doing this. This is very sophisticated. Interesting. Are you looking for the...

Steve: Yeah, this is the paper. I don't remember where I said where they had their...

Leo: By the way, somebody - I apologize to Coco. Coco, when we were talking about FORTRAN, typed some FORTRAN code into the chatroom and got booted because, as it turns out, FORTRAN code's all in uppercase, and we forbid that.

Steve: No shouting.

Leo: No shouting. So we have to write an amendment to the bot - see, you don't, you can't plan for everything - that says, you know, no uppercase unless it's FORTRAN code. Then that's okay.

Steve: So these guys say: "To show the effectiveness of our approach, we have analyzed more than 2,000 iOS applications. Our research shows that a nontrivial number of iOS applications use security-critical private APIs to access and collect sensitive user information.

Leo: Wow.

Steve: So this is happening.

Leo: This is something people discovered and have been using.

Steve: Yes, exactly, and it went through the underground, and then it spread over time. And, I mean, as is invariably going to happen.

Leo: Well, as you mentioned, there was always a brisk market for undocumented Windows API calls because you could do stuff that it was - but the risk was, and by the way, same thing on OS X, that the risk always was, well, it might get deprecated, and then your application would break.

Steve: Yes. Microsoft would always say we will not be held to the future of anything not documented.

Leo: Right.

Steve: And what it was in the early days of Windows, because I was coding Windows back then, it was Microsoft seemed to be able to do things that you couldn't do through the documented API.

Leo: Made people mad, yeah.

Steve: Yes, we were like, wait a minute, how do they do that? I want to be able to do that. And it's like, there's no way to do it. You can't get there from here unless you, you know, and then people would reverse-engineer their code and say, hey, what's this call? You know, and then it turns out to be useful. And so I just thought...

Leo: I know how Apple can fix this. They have to change their tool chain. They have to modify Xcode. And what they'll have to do is they'll have to say we will not approve any apps that are not written by an authenticated Xcode compiler. And then in their compiler they have to make sure that - because can't compilers see what - well, I guess it can't...

Steve: No.

Leo: Can't it see when it binds time what the call would do? No?

Steve: No, because the code at runtime could assemble something from pieces that would not be obvious.

Leo: Right. And the compiler can't test the runtime?

Steve: Well, or say, for example, the app could literally go to a remote server to get the name of the function.

Leo: Yeah, right. You can't verify that.

Steve: It doesn't exist anywhere in the - exactly.

Leo: Right, right, yup. Okay. Well, fascinating.

Steve: Yeah.

Leo: Fascinating. Do you want...

Steve: Q&A next week.

Leo: Okay. Here's how you do it. You can tweet him, @SGgrc. Steve responds to twits - to tweets and to twits. He'll even respond to long DMs because he's got open DMs. Nothing I would do, but anyway, @SGgrc. It's worked for him. You can also go to GRC.com/feedback and do the feedback form there. We'll assemble questions, Steve will assemble questions from that pool of questions, and we'll answer those next week, security news allowing.

While you're at GRC.com, check out SpinRite, the world's best hard drive and recovery and maintenance utility, awesome stuff. And lots of freebies, too. Steve's bread and butter is SpinRite, but he gives back all the time with so many great things like Password Haystacks, the SQRL project, you can read all about it. Even Vitamin D, it's all there, as well as 64 and 16Kb versions of this show, audio as well as transcripts, GRC.com. We have the show, as well, audio and video, at TWiT.tv/sn. And of course you can always subscribe. Every podcatcher has it, including soon

Google Music, Stitcher, Slack, Podcast Apps, they're all - we're there. Just search for Security Now!. Steve, we'll see you next week.

Steve: Thank you, Leo.

Leo: Bye-bye.