Goto: Fail

Description: The week delivered so much amazing news, much of it requiring some detailed and careful discussion, that we have a pure news podcast. It's titled from the errant line of code that was responsible for this week's highest-profile fumble of the week: Apple's complete lack of SSL/TLS certificate checking in both iOS and Mac OS X (both since fixed).

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-444.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-444-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve is here. We would normally be doing Q&As; but, man, there's so much news - bitcoin news, goto fail news, flaws, exploits. The hackers are winning. Ta da, ta da. We're going to cover the news, and it's going to be a great show. You stay tuned. Security Now! is next.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 444, recorded February 25th, 2014: Goto: Fail.

It's time for Security Now!, the show that covers your security, your privacy online. And this is a good week to do it, I tell you.

Steve Gibson: Ooh, baby.

Leo: Steve Gibson's here. He's the Explainer in Chief, the guy who knows all, tells all, pulls no punches. And if you want an independent, intelligent source for your security information, I think you need look no further. And I say that even as the guy who hosts the show on our TWiT network. It's just great. I'm so glad you're here again, Steve. Thank you.

Steve: Well, we had a busy week.

Leo: Oh, yeah.

Steve: We never really - we always try to predict, or we sort of have a presumed thing we're going to do. And when we wrap up the podcast the prior week - for example, we discussed that we would probably be doing a Q&A. But that's always given the fact that the world allows us to do that. Which for this episode, #444, we just had too much news this week. There was too much going on. So this week in Security Now! we're going to talk about Apple's difficult-to-rationalize screw-up with SSL/TLS. I mean, I've looked at the source code.

Leo: So have we all. It's amazing.

Steve: You really have to wonder how this happened. So I want to talk about that. Then we have instant messaging issues in the news as a consequence of Facebook's announcement that they're going to buy WhatsApp. And we covered a few weeks ago the fact that Facebook had changed their terms of service such that they would be allowed to read people's text messages. So it's like, okay. That's caused an exodus from WhatsApp to alternatives. And one that a lot of people have asked about is this one called Telegram, which is - we'll talk about that a lot. Then we've got the whole Netflix, Comcast, Cogent and Verizon...

Leo: I really want to get your take on this one.

Steve: ...bandwidth confusion and what happened there. Mt. Gox apparently is gone. And then I found some very interesting stats analyzing all of the security Second Tuesday patch updates that Microsoft issued during all of 2013, and what the effect of not being an admin would have had versus having administrative privileges. This of course is increasingly important as we're counting down. We have 41 days to go now until XP's no longer getting a Patch Tuesday at all. But it also applies, of course, to Vista and Windows 7 and Windows 8, that is, this whole idea of the relative - the small barrier that is put up if you run as a standard user rather than as an admin user. And because of the problem, Microsoft has continued to try things, like the UAC, screens that come up that ask your permissions and so forth. Anyway, the stats are fascinating. So we'll talk about that. And I've got a little miscellany stuff. I ran across Cringely's, Robert X. Cringely's lost interview of Steve Jobs. Oh, my god, it's the best thing I've seen.

Leo: Wow. Great.

Steve: Yeah. So we have a great podcast.

Leo: Sounds good. So the WeMo users are safe, according to TechHive. By the way, TechHive, I love TechHive, was created by Jason Snell. It's an IDG publication. Jason oversees it. And they do a really, really good job, I think.

Steve: Yup. I wanted to update from last week when we talked about the potential problems, to note something that I found out about just after we did the podcast, and I tweeted it immediately for anyone who follows me on Twitter, that Belkin had, with very little fanfare, fixed all of the problems that we were enumerating last week and just sort of silently put them out there. So although they weren't communicating, which I think is

their mistake, they need to communicate with the security community and let people know who have been reporting problems that, oh, yeah, thanks for letting us know. We've got that fixed now. I mean, because that would prevent us from having any sort of wild goose chase and keep from getting people upset and concerned when they shouldn't be.

So the good news is they responded to all of the findings, which were valid, of original insecurities, and did get everything fixed. So really rookie mistakes, like the private key used to sign the firmware no longer being included in the firmware, which is handy. Those have been fixed. And the other problems, too. So I just wanted to let people know. I mean, there was a great concern.

Leo: That's great news. That's great news.

Steve: Yes. There was a great concern among WeMo users, yourself included, that, for example, there could be substantial exploits resulting from this. And there certainly could have been. So as far as we know, those are fixed, and I wanted to...

Leo: If you got the January 24th firmware, you have the fix. And the WeMo app for iOS, January 24th for Android, February 10th will fix those. We'll give you the update. That's good.

Steve: Right. Okay. So we've titled the podcast "Goto: Fail" because this has now become the famous and infamous line of source code that was discovered when Apple fixed a problem, which we first knew about from the patch into iOS. What happened was on Friday of last week, so five days ago - well, no, not, what, four days ago - they just sort of said, oh, here's 7.0.6, without really - and there was a little mention that it, like, fixed some SSL problem. But nothing more was really said. Then what quickly came to light is that they had made a change to the source. And this chunk of source is open. It's standard open source software that we talk about a lot. And a comparison of the source revealed that a line had been removed. The line was "goto fail."

And this particular chunk of code, yup, right there on the screen now for those who are seeing the video, this chunk of code, the job of it is to verify the certificate that essentially is being used to protect an SSL or TLS conversation, Transport Layer Security, SSL, that we use for authenticating end points and for creating privacy. And this code is written in C. And it uses an interesting property of the C language. Essentially what the code shows is the standard way you would hash a bunch of different information. The way the hash algorithms work from the outside is you just - you give it a big blob, and out comes the digest. We've talked about hashing often on the podcast. Internally, the way the hashing algorithms work is they're initialized and sort of set up at the beginning, opened, and then a series of updates are issued, each one taking a piece of this blob which is being given to it. Or, depending upon the algorithm, you may be hashing a number of different things together.

So each of these update calls would take one of these things and sort of add it into the hash, updating the state of the hash. And so you'd have a series of these updates until you were done with everything you wanted to hash. And then you do what's called "finalizing" the hash, which is where the very complex internal state of the hash, which you really don't care about from the outside, that's what all the stuff you're putting in is mixing and stirring. When you finalize it, that's when you get the final output from the

hash.

So in this code that Apple had, we should say both in iOS and in Mac OS X, we can see the very clear successive calls to updating the state of this SHA-1 hash, which its job is to verify the certificate. And the idea is each of these update calls can succeed, or might have a problem. Each of these calls has the ability to return an error.

Leo: And you want to do them successively. You want to do the first one, then the second one, and then the third one in a row; right?

Steve: Yes. You have to. And so, for example, there was some discussion that I had on Twitter...

Leo: This is kind of inelegant C, by the way. It's kind of a weirdly written way to do this. But...

Steve: Well, actually I think it's correct.

Leo: Oh, it's correct, absolutely.

Steve: No, I mean, I think it's the correct way to do it.

Leo: The right way to do it, okay.

Steve: Yeah, because what you could do is you could concatenate all of these functions in a big AND, a series of functions ANDed together. But then the order, I mean, and C defines this very clearly. If you concatenate functions with ANDs, they have to be done in the order they occur, and none will be executed after the first one fails. I mean, so the C specification is clear about that. But it actually - that's less clear, I think, than the semantic construction that was done here in the source code. To me, this is very clear because we understand that the first one of these that fails, that is, the first one of these calls that doesn't succeed will cause a jumpto fail.

Now, and so the point is that, once upon a time, sort of at the dawn of structured programming, the goto was frowned on. And there was, like, all these formalizations that showed you absolutely never needed a goto. That is, you could always arrange to stay with the structured programming paradigm, so you never technically needed one. And this actually was a response to - we were, like, coming from the world of BASIC.

Leo: Spaghetti code.

Steve: Spaghetti code, yes, where you just had, like, goto 326, and goto 749. And it's like, what? So, like, it created impossible-to-read code. And so the reaction was, oh, gotos are evil. You should never use one because you don't need to, after all. Well, so it's true you absolutely don't need one, and that's been shown programmatically by

analyzing the language. You can always avoid it. But in my opinion, there are very good instances where it's clearer what you really want. I mean, in programming you're talking to two different entities. You're talking to the computer, and you're talking to a human who is someday going to read this source code.

Leo: And this is really readable. I mean, it's very obvious what you're trying to do here.

Steve: Yes, yes.

Leo: So I agree with you. This is the kind of thing a pro will write, knowing that gotos are ugly, et cetera. But the goto is right there, inside the function. It's pretty obvious what's going on.

Steve: Yes. And in fact that NOT=0 is also not necessary. I mean, so...

Leo: That's a good point. If err is all you have to say. If it's pro it's going to fail, or not fail, right.

Steve: Exactly. Exactly. The way the if statement is defined is it's actually testing for zero or not zero. And so the extra NOT=0 at the end, that's, again...

Leo: Superfluous, yeah. It's clearer, yeah.

Steve: It's superfluous, but it's clearer. Exactly. It generates no additional code. Any compiler will optimize that out so there's no cost to the result in saying NOT=0, but it makes it very clear that that's what we are intending to have happen. And so the idea is, the other part of the if statement, which is a little confusing, is that, when you say "if something," well, there isn't, in C, there's no need for then. In BASIC, it's if something, then this. Well, the then is implied. And in the definition of the language, the single statement that follows if, the if clause, will be executed or not depending upon the, well, we'll use a Colbertism, the truthiness of the clause. And so...

Leo: That's where this second goto fail becomes a problem.

Steve: Yes.

Leo: Because it's always executed.

Steve: And so the point is that the language says we will either execute the following statement or not. So that means, if you actually want multiple statements to be executed or not, you need to create a block, and you do that with curly braces. So you would, if you really wanted multiple things to happen, you'd open a curly brace immediately after

the if. And of course there's lots of arguments about whether it goes on the same line, does it go down below...

Leo: It's kind of amazing how much...

Steve: Does it go all the way to the left.

Leo: ...people debate this crap.

Steve: Do we indent it? I mean, it has no effect at all on the semantics. But it's like, well, but it looks pretty or not, and its readability, anyway, lots of religious arguments about that. But the point is you would open a curly brace. Then you'd have one or more statements and then close the curly brace. And the compiler understands...

Leo: The truth is, had they used curly braces, the error might have been more obvious, frankly.

Steve: Yes, yes. And so here's the question. So what we have...

Leo: Although it's pretty obvious.

Steve: Probably it's not possible to explain this verbally on the podcast. But essentially there's an alternating set. There's the if and then a goto fail, and then an if statement and a goto fail, and an if statement and go to fail, the idea being that we are successively doing these hash updates. And we keep doing it only if they all succeed. So we update the hash. We make sure that that update function does not return an error. If it returns zero, then that's like, okay, fine. Then we do the next one. But if it returns a non-zero, then the if statement is true, and so the statement following the if, which is goto fail, that jump, that goto is executed, taking us to - it frees some buffers, and it returns the failure code from that most recently executed update.

And so every little detail of this is important because what happened was somehow the goto fail was duplicated. Either there was another if statement in between them, and it was deleted, or the goto fail just appeared twice. So what happens now is, when the if statement before the doubled goto fail, when it succeeds, it does not execute the goto - remember, when it succeeds, it returns zero, so the if statement is false. It does not execute the statement that follows it, which is the first goto fail. But it does execute the second statement that follows it, which is unfortunately the extra goto fail, the doubled one.

Now, what's really interesting is that, when we goto fail, we do so with a non-failure error. Remember that that update succeeded, meaning that it returned zero, which means no error. So we goto fail, free up the buffers, and then return no error. And that's the mistake. So the idea would be that this function is called VerifySignedServerKeyExchange. That's the function's job. And just as a consequence of this doubled line, this function never fails. That is, it cannot...

Leo: Even though it says goto fail, and it always goes to fail, the variable error, e-r-r, is a success.

Steve: Is a success.

Leo: And that's what's returning.

Steve: Always returns success. So even if you have a bad certificate, unsigned, bogus, malicious, anything, this thing, when you call SSLVerifySignedServerKeyExchange, it says, yeah, we got no problems. So, now, here's the conspiracy theorists because, when you look at this, I mean, and that's the reason I wanted to explain carefully what it means to goto fail after a success, and how that always returns success. I mean, yes, this could just be lightning striking. Yes, this could just somehow have happened. But it also is incredibly clever. I mean, do a million monkeys at typewriters ever actually create Shakespeare? Well, this did.

Leo: It's the kind of thing, I mean, it's really - it's 11 letters, including the space. Actually 10, including the space. Took three seconds to type. It's the kind of thing, if you really wanted to make the logic fail, you could just put it in there pretty quickly and easily.

Steve: And you have plausible deniability. That's just it.

Leo: Right, it looks like it was doubled by accident.

Steve: It looks like a mistake, smells like a mistake, walks and talks like a mistake. But at the same time it's like, it's diabolical.

Leo: It's elegantly done.

Steve: It's good. A monkey got lucky. And it's just like, it's amazing to me. So anyway, so it caused a huge controversy over the weekend. A site was created, GoToFail.com, which I used this morning, as did everybody who had Macs, to see whether the update we all received this morning to Mac OS X fixed the problem. And if it did, it did so silently because nowhere in the update does Apple say, oh, and we fixed this really embarrassing problem everyone's been screaming about all weekend. They talked about all kinds of other random stuff no one cares about that they fixed in, what is it, 10.9.2 was just released a few hours ago.

And so I used Safari and verified that my little mini was having the problem. Oh, and I also ran Chrome and Firefox. They never had a problem on Mac OS X because they use NSS, the Netscape Security Suite, to do all their crypto, rather than the native crypto in the Mac. So Safari was vulnerable. The alternative browsers, that is, Chrome and Firefox, weren't. So people who were browsing with those browsers would have been safe. But all the other infrastructure of the Mac and the Apple cloud world was all using this built-in

broken SSL. And we know that iOS 6 was broken, and 7 was until Friday. I didn't have time to do the research to look back at where this entered OS X.

But, for example, the DaringFireball.net blog, John Gruber, he said - his blog posting was "apple_prism." And noticing, I think it was a few days, or maybe it was the month after the PRISM revelation, or maybe it was a month before, I don't remember what John wrote. But it was, like, right in the area of them saying - oh, I know what. It was that the PRISM documents alleged when Apple had joined PRISM, and apparently this occurred...

Leo: Same day, October 2012.

Steve: ...right around the same time.

Leo: Yeah, yeah. So he's quoting Jeffrey Grossman on Twitter, who says the SSL vulnerability was introduced in iOS 6.0, which shipped on September 24, 2012. And the slide in the leaked PRISM PowerPoint deck said Apple was, quote, "added" in October 2012. Wow. I mean, that's a conspiracy theory. I'm not sure I buy it. But still...

Steve: No, no. And we will never know. This is almost too obvious. I mean...

Leo: It's in open source code; right? I mean, it's not - that's why we know.

Steve: Yeah, it's there for everyone to look at, absolutely. And so, okay. So we'll never know. What we want are really subtle things like random number generators that aren't really random but everyone thinks they are. That's elegant. That's insidious. This, where it just doesn't work at all, I mean, it's just completely broken.

Leo: It's a little bit of a broad brush.

Steve: It took an hour for someone to say, oh, let's test this and, you know, give it a bogus certificate. It's like, oh, it thinks its fine.

Leo: I think that's maybe more of an indictment is that Apple's own testing didn't discover this, frankly.

Steve: Well, yes. And that's what all of the people who understand how this should be done, how you do regression testing, the idea is that, with everything you do, you also create a...

Leo: A test, a test unit.

Steve: ...a test, exactly, a test unit to make sure that the things you are doing are

working. So somewhere, I mean, this is what - it's hard to understand how this escaped Apple's attention for so long. The idea that nowhere between 6.0 of iOS and now did Apple ever test their SSL connectivity by giving it something bad and making sure that it's unhappy. I mean, what you want is there should be a test of SSLVerifySignedServerKeyExchange.

Leo: Seems pretty obvious.

Steve: To make sure it says this is, no, this is bad, this is broken, this is not a good exchange. And it would have instantly raised a red flag, and somebody would have immediately caught this double blind.

Leo: And a supervisor doing code auditing would have seen this because it sticks out like a sore thumb.

Steve: Yeah.

Leo: I mean, any coder who looks at that code immediately sees the flaw. I mean, it's hard not to see.

Steve: Yup. Yup. There's absolutely no reason for two gotos in a row because the first one went. So the second one could have never come.

Leo: One of our chatters said that the IDE he uses, the programming editor that he uses would have flagged it as a double blind. I mean, it just seems - it's a little discouraging, frankly.

Steve: It is hard to understand, yeah. And Adam Langley's blog at ImperialViolet...

Leo: He's the guy who kind of revealed all this, yeah.

Steve: Yeah, really great coverage of it. And of course Ars Technica and iMore and everybody, just, like, wow. So it was fixed on Friday. Everyone wants to be at 7.0.6. And I haven't followed the controversy about 6, but there are earlier versions that cannot come up to 7, is that the deal? So they also fixed...

Leo: Yeah, earlier hardware that can't be fixed; right.

Steve: Okay, right.

Leo: And there's a new bug that somebody's discovered, but...

Steve: Oh, we've got that. We have that coming up next, as a matter of fact.

Leo: Oh, oh, oh, oh, oh. You know, there's actually an interesting story on Slashdot. The Linux - thank you to the chatroom for passing this along. The Linux backdoor attempt of 2003, a slight change in code, similar to what we're talking about, just a little change that looks like innocuous code, but it says =0 instead of ==0, which is how you do a Boolean test in C; right?

Steve: Right, equals as an assignment.

Leo: So this is assignment instead of a test and, in fact, is a backdoor. It offers a backdoor into Linux. The Linux team caught this. But this is the kind of error that you see all - this is a typo type of error that you see all the time in C.

Steve: Yup.

Leo: It's a very common error. So it might be their link caught it or something else. But even at the time, in 2003, Ed Felten said, "Could this have been an NSA attack? Maybe, but there were many others who had the skill and motivation to carry out this attack. Unless somebody confesses, we'll never know."

Steve: Yeah.

Leo: And that's Ed Felten, one of the great security researchers, one of the legends in the business.

Steve: And you could argue, too, that this is a flaw in the language, that it should not be so prone to something so erroneous, I mean, the idea that you would not detect an assignment where you are checking for equality, that it could completely change the meaning of what's happening. I mean, it's just bad that the language sort of...

Leo: But every C programmer knows this, and it's the first thing you check, and I'm sure most automated link tools and the like see that.

Steve: The other thing we didn't mention is the fact that the indenting of C - I did talk about how indenting doesn't matter to the meaning of the language. But of course Python famously doesn't have curly braces.

Leo: Tabs count.

Steve: Exactly. And it's really interesting then because then what you see is what you get. Whereas, where indenting is done for readability, but not for semantic meaning of the language, of what you're constructing, that's another place where you've got a

problem in C is that indenting can mislead us into assuming that that's what we - that the indentation means something to the compiler, when it absolutely doesn't.

Leo: White space is ignored.

Steve: Whereas in Python, exactly, where with Python this would not have been an error. This would not have worked in Python because it would have understood the indentation to mean the implied block underneath the if statement.

Leo: Right, that's a good point, yeah. That's a good point. There is no perfect language, but that's the way it is. It's programming.

Steve: Although Python is getting a lot of traction.

Leo: I love Python. I've always loved Python.

Steve: I've had some friends say that they're more productive in Python, I mean, some seriously seasoned veteran programmers who know their way around, saying that they produce more code, more quickly, that's better in Python.

Leo: Python and Ruby. But they're both scripting languages. They're not production languages like C, C++, or assembler. But they're fun to write in. They're the kind of languages that programmers like because they can be very productive. They're fun to write in, yeah.

Steve: So we do have a new - another exploit just came to light. The guys at FireEye, whom we've spoken of before, have - they got a proof of concept through the Apple app vetting process into the Apple Store. Benignly, I mean, this was a proof of concept. They're working with Apple on this. What they have demonstrated is that it is possible to do background keystroke monitoring across applications. And of course that's the definition of malicious keystroke monitoring. We know, for example, that we've got multitasking in iOS. And it was expanded in 7 so that more things could run in the background. And music apps, for example, have long been able to do that. You start playing music, and you switch away from it, and it continues to play music.

Well, with iOS 7, that was expanded. And it was also made visible. In the control panel there is now the opportunity to control the background execution of things. And, for example, one of the recommendations, if you're having, like, power drain, battery short life problems, is to go look and see if all the things that have asked for background operation privilege, you really do want to have running in the background. And if not, you can flip those things off, and iOS will allow them to run briefly to do any little housekeeping they may need to do and then suspend them.

So what we have is verification that a background app can - what it can do is it can pick up the press-and-release coordinates on the screen. It can also detect basically all of the UI actions, volume up and down buttons, the Touch ID event, the pressing of the home button, and the power button, so all of those events, including you tapping on the

screen. And of course the screen is where we enter passwords and our Apple ID verification to authenticate ourselves to iCloud and so forth. So that's not good. We have generally fixed keyboards, and I would argue that we've seen successful malicious use of the much less information than a background app is able to obtain. So this unfortunately happened after the release of 7.0.6.

Leo: It's nontrivial, by the way, to implement this because you have to get an app on the App Store. And because Apple knows about this, I can imagine they'll be looking specifically for this.

Steve: Exactly. Exactly. So even if we don't have - I was just about to say exactly that, Leo, is even if we don't have an update, it would be, I mean, hopefully Apple will go through and look at all the existing apps to make sure that nobody's doing it already, and it'll get fixed, because you absolutely don't want something monitoring your keystrokes when you're using a different application.

Leo: Bigger image, though, if you use a jailbroken app store, this is a reason why it's probably not a good idea, because nobody's looking at those apps.

Steve: Right, right. Okay. So Facebook announces to great fanfare last week that they're acquiring WhatsApp for an incredible amount of money. Was it $13 billion?

Leo: Try 16, with an additional 3 billion over four years, if everybody stays.

Steve: Wow. Okay, well, they already lost that because everybody's not staying. Or maybe they will gain more people.

Leo: Key players have to stay, yeah.

Steve: Yeah.

Leo: I'm not sure what the exact details are.

Steve: So, and what was that, one third of Facebook's cache, I think I remember reading. So, phenomenal.

Leo: 10% of their total value.

Steve: Wow. Wow.

Leo: It's a lot of money.

Steve: So among the security-conscious community there was, as I mentioned at the top of the show, an exodus, or is - we're in the process of there being an exodus. In fact...

Leo: Which is, by the way, somewhat ironic because I don't know why anyone would assume WhatsApp is in any way secure. It doesn't make any claim of encryption or security; does it?

Steve: No. Well, they all say, oh, we're secure messaging, although there have been, and we haven't covered it because I just sort of said, well, okay, there isn't time to talk about everything. I mean, there have been some WhatsApp failures.

Leo: Oh, yeah, it publishes your phone number in a way that malware can get it and use it. So there were four...

Steve: Yeah, I would call that a problem.

Leo: Yeah. There were four Android malware apps which have been removed that signed you up behind the scenes for paid messaging, SMS messaging. And it did so because it could get your number from WhatsApp. So they just queried WhatsApp, hey, what's the guy's phone number? Okay, we'll sign them up. So, I mean, it wasn't ever that secure, I don't think.

Steve: No. So the good news is the security-conscious people are now even more concerned with Facebook being the parent. And we just know in the long term that's not going to be a good thing. So, many things happened. Threema, the messaging app that I have looked at and have recommended in the past - and still recommend because what they're doing is so transparent and simple and clear that it makes sense. And it's very simple to do good security. I mean, we have all the tools. All the pieces are in place for solving these problems. These are not hard problems anymore. These are solved problems.

And so Threema, T-h-r-e-e-m-a, they saw their user base double after the announcement because a lot of people went to them. Telegram, which is Telegram.org, saw 8 million downloads of Telegram, their messaging app, after WhatsApp got acquired, and after this announcement. Again, so people were moving away. I started getting tweets from people, from our listeners who follow me on Twitter, saying, hey, what about Telegram? What do you think about it? I had never seen it, looked at it, not really even heard about it. Is their logo that paper airplane that kind of jumps around constantly? If so...

Leo: Yeah, the logo's a paper airplane, yeah.

Steve: Oh, yeah, there it is. So I had seen it visually, but I'd never dug into it and looked at it. But so many people were tweeting me that I got tired of saying go look at Threema. I thought, okay, let's go look at Telegram. So I go to the website, and I'm impressed by the openness of what I see. I mean, they appear to be good people.

Everything is open, open source. They're multiplatform. There on the screen you are showing their API. There's a bunch of different stuff going on. They're running competitions. They're like, who can create the best Telegram Android app and so forth. So all that seems good.

So then, of course, I want to go find out about the crypto. And I see this diagram that is the most bizarre-looking thing I have ever encountered. I mean, I don't think I'm exaggerating. Maybe it's the most bizarre thing I've ever encountered. And I look at it, and they're like, they're taking the plaintext of the message, and then they're running it through an SHA-1 and appending the result to the ciphertext, but then also using the output of the SHA-1 through what they describe as a KDF, a Key Derivation Function, several times to generate keys for an AES cipher that uses - there's the diagram, you're showing it, Leo - that uses a block mode I've never heard of. We've talked about how with the symmetric cipher you can't just use it over and over and over. Everyone will be familiar with that famous picture of the encrypted Linux penguin, where you can still see the Linux penguin, even though the message has been encrypted, because it was done with a block cipher that did not use an encryption mode. And we've talked about modes like Cipher Block Chaining (CBC) and so forth.

Well, turns out there's one I had never heard of. It's not on the Wikipedia page on symmetric cipher block modes. It's called IGE, Infinite Garble Extension.

Leo: Sounds good to me. That sounds real strong.

Steve: And it's like, whoa, okay. So I went digging into what the heck is Infinite Garble Extension. And now I know, and I don't know why, but it's just something someone came up with. It's got XORs crossing over, and it looks very confusing. And so I guess that they're figuring, well, the bits will be more confused by being garbled this way than if we didn't confuse the bits. But so suffice to say, what these guys have done is they've rolled their own. And they've used chunks of crypto. AES, that's good. And SHA-1 is, eh, well, that's old, and there's lots of reasons you wouldn't use SHA-1 today, if you were creating something, or even yesterday, if it wasn't too far long ago, because we know that MD5 and SHA-1 are subject to various kinds of attacks.

And so anyway, so this really piqued my curiosity, and I thought, okay, what the heck is going on here. So they have an FAQ where they ask themselves some questions and answer them. Because one of the things that I couldn't figure out was how this gave them end-to-end secrecy, how they got - what users of chat clients want, of chatting systems want, is they want no man in the middle. They want man-in-the-middle protection. They want to know that what they send is encrypted from before it leaves their phone till after it arrives at the other phone, and there it's decrypted and is presented. This doesn't do that. So that's first, is that this is not what Telegram does. Telegram, in order to, they say, create this cloud experience, where you can get your chats on multiple devices, they cannot do that, they say, and offer end-to-end encryption. Well, I could. But they say they couldn't. And other people are, but these guys don't. So everyone wondering about Telegram has to understand that, by default, the normal way it operates is not secure.

Leo: But they say that they do have an end-to-end secret chat style; right?

Steve: Correct. And I don't think that's secure, either.

Leo: Oh.

Steve: Well, because there are, I mean, what they've done is they've invented their own solutions. And people have continued to find problems with them and point them out. And then they add some more glue to make it stronger. It's like, oh, well, this was a little rickety, so we just put more glue in.

Leo: Yeah, that's kind of a bad approach. It's kind of, yeah.

Steve: Well, yes. So they say, asking themselves the question "How are secret chats different," their answer is "Secret chats are meant for people who really want secure messaging. All messages in secret chats use end-to-end encryption. This means only you and the recipient can read those messages. Nobody can decipher or intercept them, including us here at Telegram," which turns out actually not to be true. "Messages cannot be forwarded from secret chats. You can also order your messages to self-destruct in a set amount of time, after they've been read by the recipient." Well, see, now, this kind of thing really makes me uncomfortable because we know that's not true. I mean...

Leo: A screen capture would preserve it.

Steve: Yes, exactly. And they have open source software, and everybody could create clients. So any of those clients could be modified so that it doesn't obey the "destroy after reading" command. So one of the things that's really wrong is, in their secrecy assurances, to tell people that you can send this in a way that it will be destroyed at the other end, when that's not something they can do.

Leo: They can't guarantee that because somebody else might be writing the client.

Steve: Exactly. Precisely.

Leo: And logging the entire chat to a plaintext file on your phone.

Steve: Right. So they ought, if this is in their discussion of how secure everything is, they're telling you, oh, it's so secure, you can make it self-destruct. Well, okay, but you can't.

Leo: That's not true, right.

Steve: So then they say: "One last difference between secret and ordinary chats in Telegram is that secret chats are not stored in our cloud. This means you can only access messages in a secret chat on their device of origin." And I guess their device of destination, too. Otherwise you'd be chatting with yourself, and that wouldn't really get you anywhere. Then they asked themselves the question: "Why not just make all chats

secret?" And they answer: "The idea behind Telegram is to bring something more secure to the masses, who understand nothing about security and want none of it."

Leo: Apparently not. And that's what we're going to give them [laughing].

Steve: That's what we're going to deliver. Okay. "Being merely secure," they say, "is not enough to achieve this." So not being merely secure, being merely secure doesn't buy that for you. "...[I]s not enough to achieve this. You also need to be fast, powerful, and user friendly."

Leo: Well, it is true that in order to do a secret chat you have to exchange QR codes and stuff, like Threema. So it's a little more complicated.

Steve: Well, actually, no. To achieve three dots and absolutely provable authentication, you show your phone to each other, and they exchange keys. And then from then on you absolutely know. But, for example, you get security just by exchanging keys, as long as you're sure who you exchanged it with. So that's all you need. And you can arrange that out of band, or take a picture of your QR code and paper mail it to someone. I mean, there are ways to do this. Anyway, I don't want to take up too much time here because I have more I want to say.

First of all, a great analysis - and I've got links to all this in the show notes, by the way. So people who want more, there is more. I guess I'd pronounce this name Geoffroy Couprie. He blogged at UnhandledExpression.com - neat blog, by the way, that I just discovered by discovering his coverage of this. He came out early on and really took these guys to task. And I'm jumping all the way down to his fourth edit. And in the blog he shows the interaction of the Telegram guys and himself as they go back and forth talking about this. And so this is typical of when you just keep gluing stuff on.

He said: "In Edit 4, someone found a flaw in the end-to-end secret chat. The key generated from the Diffie-Hellman exchange was being combined with a server-provided nonce," which we know is a one-time random value. And then he shows the equation where it's done. "With that, the server can perform a man-in-the-middle attack on the connection and generate the same key for both peers by manipulating the nonce, thus defeating the key verification. Telegram has updated their protocol description and will fix the flaw. That nonce was introduced to fix random number generator issues..."

Leo: On mobile devices.

Steve: "...on mobile devices."

Leo: Yeah, right. Since they can't be trusted, we'll do it for you.

Steve: Exactly. So, and here's the problem. This is a perfect example of continuing to just add insult to injury. Geoffroy finishes, he says: "To sum it up, avoid at all costs. There are no new ideas, and they add their flawed homegrown mix of RSA, AES-IGE" - that's the garble block system - "plain SHA-1 integrity verification, MAC-then-encrypt,

and a custom key derivation function." I mean, it's just every cryptographer who has looked at this has just gotten the shakes. And our friend Moxie Marlinspike at ThoughtCrime.org, he also blogged. And so one of the things on their site that I want to make sure our listeners get is they have created a contest. They have produced a $200,000 award to anyone who cracks their encryption. And they're using that as, like, how much they believe in the integrity of their crypto.

And several cryptographers have taken them to task because it turns out this is actually not very useful. Moxie's is very good. His blog was titled "A Crypto Challenge for the Telegram Developers." And he said: "Earlier this week, a company called Telegram announced a 'secure' mobile messaging product. How secure? In the words of their FAQ, 'very secure.'" Oh, well, okay, good. "Curious to learn more, I went to look at the protocol," says Moxie, speaking, "and immediately had a number of questions and concerns." As would anyone. "However, when pressed on technical details by others, they responded with the academic credentials of their developers" - they have math PhDs - "instead of engaging in a more reasonable dialog. They also declined my suggestions for collaboration of any kind. Most recently, they've chosen to respond to the concerns of the security community with ... a crypto-cracking contest."

And then Moxie goes on to explain, and this is the point, the fallacy of the crypto contest: "As always, these things are a bad sign. By framing the contest the way they have, the Telegram developers are leveraging a rigged challenge to trick the public. They wasted no time in updating their FAQ to point to the challenge as solid proof of their absolute security, even when it's essentially meaningless. So, Telegram developers, by way of a response, I have my own crypto cracking contest for you. Below is a horrifically bad 'secure' protocol that wouldn't last a second in a real-world environment, but becomes 'unbreakable' when presented in the exact same framework as the Telegram challenge." And he goes on.

So I wanted to just make the point, I wanted to be sure that I didn't miss making the point that just creating a huge award for someone finding a flaw isn't proof of anything. And anyone who's curious can follow the links and spend some more time here because it turns out that the way they have established what they're trying to do doesn't reflect the lack of security.

And then finally, my favorite quote here about this came from Taylor Hornby, whom we've talked of before. Defuse Security is his site. His handle is "FireXware." He's a frequent and valued contributor to GRC's forums, and they've been studying and auditing crypto stuff for quite some time. He wrote, of Telegram, of course: "Some problems are immediately apparent. They use the broken SHA-1 hash function. They include a hash of the plaintext message in the ciphertext. Essentially, they are trying to do 'MAC and Encrypt,' which is not secure. They should be doing 'Encrypt then MAC' with HMAC SHA-512," for example.

He says: "They rely on an obscure cipher mode called Infinite Garble Extension." Then they have "some really weird stuff about factoring 64-bit integers as part of the protocol, they do not authenticate public keys." Taylor says - and here's my favorite quote. He says: "If their protocol" - and this is like a ways down, so anyone, again, can follow links if they're curious. "If their protocol is secure, it is so by accident, not because of good design." I love that. If it's secure, it's an accident, which is exactly right. I mean, the thing - you just look at it, and anyone who understands crypto just, like, what?

So Taylor says: "They claim the protocol was designed by 'six ACM champions' and 'PhDs in math.' Quite frankly, the protocol looks like it was made by an amateur. The tight coupling between primitives suggests the designer was not familiar with basic constructs,

like authenticated encryption, that you can find in any cryptography textbook." And he says: "What should Telegram do? Telegram's crypto is bad and needs to be scrapped. I know it's tough to throw away all that work; but, if they want to build a trustworthy product, it's what they need to do. Their protocol is already too complex to analyze, let alone prove secure, and adding Band-Aid fixes is only going to make it worse. They should switch to an existing well-studied protocol like the one used by TextSecure. They need to bring in a real cryptographer to audit their design and design process, and they need to make sure the programmers they've hired are qualified to write crypto code. Most programmers are not. If Telegram wants, they can email me, and I'll offer as much advice as I can. I think their hearts are in the right place. They just goofed on the crypto." So there's my answer to everyone who was tweeting about should we go to Telegram. I would say probably not.

Leo: And you like Threema in its place?

Steve: I like Threema very much. Taylor referred to TextSecure, and that is also Moxie's product, just released yesterday. So Moxie blogged about it yesterday. It's Android-only at the moment, but they do intend to do an iPhone version. And I would trust it implicitly because talk about guys who know crypto, Moxie at WhisperSystems certainly does. It's WhisperSystems.org is Moxie's company. And they've been working on TextSecure for quite a while, and it is just released in final form. Yup, there it is on the App Store.

Leo: And it's free.

Steve: Yes, and it is free.

Leo: And it looks like it does most of what, not all, but most of WhatsApp does. It does images. I think a lot of people are leaving WhatsApp more because they don't want to be a party to Facebook. But WhatsApp is group chat, images, audio, video, and they're going to add phone calls, as well. And I would guess the majority of people who use WhatsApp don't expect or think that it's particularly secure, any more so than SMS. It's a replacement for SMS.

Steve: Right.

Leo: But if you wanted secure, you think this is the one to get. I'm going to install it right now.

Steve: Yeah.

Leo: I have to say I stopped using Threema because, A, nobody uses it, which is, frankly, an issue. If you can't communicate with other people, doesn't matter. Because it is kind of complicated to establish three dots. You have to meet them in person and all that. We never did that, did we. You never - you were going to exchange your Threema with us, and we forgot.

Steve: Oh, you're right, when I was up there. I could just stick it in front of the camera right now.

Leo: I did it, yeah, I did that once, yeah.

Steve: I like Threema because it feels very clean, and we know it's very secure. So if your mode of operation is such that you have, well, for example, I've got a bunch of people I hang out with at Starbucks. And if I wanted, and if I had any need for secure texting among them, and I don't, but if I did, we'd just have a little Threema party, where we all aim our phones at each other. And then in that moment we have verified each other's identity and exchanged, through that mechanism, keys for each other that only exist. And here again, absolutely secure end-to-end encryption simply means that you give each other your public key. End of story. That's it. That's all it takes.

And then, because if you have somebody's public key, you simply - you generate - you need a good random number generator. We understand that. You generate a good random number. You use that as your symmetric key to encrypt the message. And you use another random number to authenticate the message if you want to do that, and that's a good idea. There are some block modes. The one that I'm using for SQRL is authenticated encryption. It's OCB, that's Phil Rogaway's mode, which is Offset Code Book, which does both. And, by the way, the Telegram people, if they'd used that, they wouldn't have had any of these problems. But the point is then you use the recipient's, the intended recipient's public key to encrypt the symmetric key which you arrived at randomly, and you send it to them. Only they have the matching private key. So they use that to decrypt the symmetric key, and then they use that to decrypt the message. I mean, it's like, it's done. This is solved. This is not hard.

Leo: It is very straightforward, yeah.

Steve: And if you want to do a group chat, then you just - you take the main message, and you successively encrypt it with different symmetric keys, which you successively encrypt with all the members of the group chat's public key. That you send off. And all they do is they use their private key to decrypt the one that they're able to. And then, so, I mean, group chat is not hard. It's like, it's a mystery to me why anyone would invent something so screwy in this day and age.

Leo: Where are these Telegram people from?

Steve: I think they're Russian.

Leo: Ah.

Steve: Because I did see something that said, would you like to do encryption in Russia or in Switzerland? Meaning that Threema is Swiss.

Leo: That might say something about it. They get money from Putin. Actually, they do have money from Pavel Durov, who's the investor in this. He's an interesting guy.

Steve: We absolutely know that, unless you explicitly ask for end-to-end encryption, it is not secure. You have a secure connection to their server. Then they decrypt it, so it's in plaintext, and they acknowledge that, if you dig around in there enough. They go, yeah, but, you know, we're just good guys.

Leo: To me it's a little ironic that people who are fleeing WhatsApp because they don't like Facebook are going to Telegram because it's funded by the guy that Rene Ritchie calls the Mark Zuckerberg - or not Rene Ritchie. Mark [indiscernible] calls him "Mark Zuckerberg of Russia," Pavel Durov, who is probably one of the richest guys in Russia, and a very interesting fellow. Kind of secretive. So, yeah. Anyway, that's who's behind Telegram.

Steve: So TextSecure, Android-only at the moment, iPhone coming; or Threema, if you just want something that's, like, done, solved, and you want really secure messaging between people you know who will also install Threema. As you said, Leo, it requires itself at the other end.

Leo: And we should mention that Moxie also has a voice app called RedPhone, which we've mentioned before.

Steve: Yes, yes, yes. Good stuff, all. Before I forget, I did want to mention I ran across a note just this morning, actually, the 25th of February. The subject line caught my eye. He said: "SpinRite violates the laws of physics." Of course SpinRite is what pays all of my bills and lets me be here for the podcast every week. This is Nathan Huebener in Marion, Iowa. And he said: "My brother's laptop was very slow when booting and while being used. The hard drive was suspected as being bad. Norton Ghost tried to clone it and failed. It was not pretty. I ran SpinRite at Level 2 and was then able to clone it to another drive. It cloned over perfectly, and no bits were lost. After the new drive was installed and running, I opened up the bad drive to look inside. Inside were tiny metal shavings."

Leo: That's not good. Oh ho, baby, that's not good.

Steve: That's never a good sign. "From what I understand, if a piece of dust goes between the head and the platter, it's game over for the drive. Somehow, SpinRite was able to fix the drive just long enough to clone it. SpinRite has also fixed my Intel X-25 Extreme 64GB SSD. It was showing some sector access times over 600ms. Now they're gone. SpinRite user" - oh, he's a SpinRite user and Security Now! listener since Episode 1, Nathan Huebener, Marion, Iowa. Thank you very much, Nathan, for sharing your success.

Leo: Yay. Comcast. Netflix. Tell me.

Steve: Yeah. So okay. So it's complicated.

Leo: Yeah.

Steve: Which is what makes it interesting.

Leo: Yeah.

Steve: Which is what makes, I think, is what makes it interesting. We've discussed in years gone by the nature of peering agreements with major top-tier bandwidth carriers - Comcast, Level 3, Cogent, AT&T, Sprint, these are all big bandwidth carriers. And the idea, of course, with the Internet is they all have customers, so the customers connect to their networks. But the customers may want material in their network, or they want something else, somewhere else on the Internet, on somebody else's network. That is, for example, Netflix is a customer of Cogent. Cogent provides very affordable bandwidth. And Netflix says, hey, we're going to - Cogent is the major provider that connects Netflix to the Internet. And for example, I'm also a Cogent customer. Cogent supplies my bandwidth. So if I were to use Netflix, then it stays in the family. The bandwidth stays within Cogent. It goes through a few Cogent routers from me on Cogent's network to Netflix on Cogent's network, and that's all there is to it.

But what about somebody who wants Netflix on Cogent who's a Comcast customer? Well, in that case the traffic needs to cross the boundary between those domains. So these top-level providers have these huge network domains of their own interconnected routers, and so that by itself is useful unless, as is mostly the case, you want to go somewhere else. And so they have to interconnect. And so this is the process known as "peering," where they peer with each other, that is, they regard each other as equals.

So these large domains created by these top-tier providers interconnect with each other at so-called "peering points" all over the world. And that's how the traffic moves. Sometimes it might cross through a provider on its way to another. So that can also happen. But the idea is that, between any two providers, they just agree that they're going to exchange traffic because each is receiving value from the other.

Leo: Well, and bits from the other. And that's one thing that I think is unusual in this situation because "peer" implies equals and that it's a two-way street. And a lot of peering agreements are mutual because I give you bandwidth, I give you data, and you give me data, and we agree to peer. But what Comcast is saying, well, Netflix is firehosing data into our network. It's a one-way peering agreement.

Steve: Well, okay. So the reason that's a problem is that - one way to think of it is that any given ISP has retail users that are, like, all the end-users.

Leo: Us.

Steve: Yes. And we pay retail prices for bandwidth. But they also host wholesale bandwidth users like Netflix or like Google or any major bandwidth user. They're buying

bandwidth at wholesale prices. So in a peering relationship, as long as you've got sort of an even amount of bandwidth - this is why the bandwidth equality thing comes in is, when you think about it, for example, if all of the retail customers for Netflix were over on Comcast's side, and the wholesale provider was over on Cogent's side, well, then, the disparity in who's making revenue from the traffic becomes a problem. And so what you want is you want it to be about equal.

And in fact, when I signed up, I'll never forget, when I signed up for Level 3, which is one of the top-tier providers, and my servers, GRC servers are in a Level 3 datacenter, one of the things they asked me was what's the ratio between inbound and outbound traffic? And I was like, oh. Because I knew that, like, they didn't want it to be too unbalanced. They're all thinking about keeping their peering relationships relatively balanced. But the fact is any web server is sending out much more traffic than it's taking in. And so I said, oh, about 10:1. And they said okay. They just didn't want to be...

Leo: They could live with that.

Steve: Yeah, they didn't want it to be 10:0 or 1:10,000 or something. They wanted to generally kind of keep it even. So here we have this dilemma with Netflix. Understand the scale of Netflix. In the evening, in North America now, Netflix accounts for 32% of downstream traffic on the Internet. One third of the traffic flowing on the Internet in the evening is Netflix. So it is huge. So what this does is this creates a massive imbalance in these peering agreements because all of these companies with end users, they're having to carry all of this traffic from Netflix. And Netflix is over here in the Cogent domain, so all of this has to cross peering points, which upsets sort of the long-term mutual agreement of balance, like balance of peering agreements.

Now, Netflix understands this. This is not news to anyone. And Netflix has been solving the problem with a program they call Open Connect. What Open Connect does is it moves Netflix servers into the domain, this networking domain of the major providers. And think about it. It solves the problem. So, for example, if Netflix didn't have servers in, for example, let's use someone where they do...

Leo: A good example would be Google Fiber. If you have Google Fiber, they're a member of the Open Connect CDN.

Steve: Perfect, exactly. So, but think about it. If they - okay. So right now, for example...

Leo: As are, by the way, Cablevision, Virgin, Bell Canada, British Telecom, Clearwire, GVT, Telus, in fact, almost everybody except Comcast.

Steve: Actually Comcast and Verizon are the two.

Leo: And Verizon, yup.

Steve: Yes. So if Verizon refuses to - by the way, this is a free program, Open Connect.

Verizon refuses. That means that, if there's a thousand people in Verizon all wanting to watch the same episode of House of Cards, then there's a thousand streams coming across that boundary between Cogent and Verizon of the same content. It's stupid. But that's the way it is now because Comcast and Verizon refuse to participate in Open Connect. What Open Connect does is Netflix puts servers over in Verizon, and it completely solves the problem.

Leo: Doesn't cross the Internet at all. It's sitting in Verizon's Network Operations Center.

Steve: Exactly. So now those thousand users, first of all, it's better for them because the source of the content is right next to them, networking-wise. They're going to get much better streaming performance than if it has to come from further away. And what has been happening is, due to this fight between Comcast and Verizon and Cogent, Cogent being the one they're fighting with because that's Netflix's provider, is routinely now in the evenings the connections, those peering points have gone past saturation. And we've talked about the way routers operate and how routers have buffers. And the problem is there is more traffic trying to go through the wire than the wire can contain.

So the buffers buffer a little bit in the hope that the bandwidth will free up, and they'll be able to still get their packets through. If not, the packets fall off the end of the buffer. There's no more buffer space for them, and they're just lost. And that causes stuttering and problems. And not as much, necessarily, for Netflix because the protocol buffers ahead, and it manages to deal with dropped packets. But many other services are still trying to use the Internet. Even though a third of the country is watching Netflix, everybody else would still like to do web surfing and other things. Those other services are having problems at these peak times if the traffic needs to cross this peering boundary. And Cogent has refused to pay for this imbalance. They've flatly refused. And so what brought this...

Leo: Why isn't Cogent the bad guy here?

Steve: I'm not sure that they're not.

Leo: Isn't it pretty - it's kind of traditional that you might pay for this; right?

Steve: Actually, what's traditional is that you don't.

Leo: It's mutual. It's a peering, yeah.

Steve: Yes, that it is regarded as mutually beneficial, the idea being that Cogent has servers that Verizon and Comcast customers want to get to, namely Netflix. And so remember when we had the fight, it was CBS was dropped by, who was it...

Leo: DirecTV or Dish or - it was a satellite network. Oh, you're talking about...

Steve: Yeah, it wasn't a satellite, it was not...

Leo: Yeah, yeah, it was Cablevision, I think, yeah.

Steve: Cablevision, yes, where it was the big fight that people were - in fact, Cablevision, as a consequence of that, lost a huge chunk...

Leo: It was Time Warner, I'm sorry, Time Warner.

Steve: Oh, yeah, you're right, Time Warner wouldn't agree to pay what CBS felt their content was worth. So I guess bottom line is we're going to have battles as we sort of sort all this out. I don't understand why Comcast and Verizon aren't willing to accept Netflix's solution, which is to put Netflix servers in their networks. I think it's because they're insisting on being paid by Netflix for - essentially they're saying Netflix is benefiting from their carriage of customers, their own customers, who want to watch their content, and Netflix should pay. The problem is that's the way the whole Internet works. The whole Internet is free services that are offered, or you pay for services. And I guess maybe that's the difference is that Netflix is making money. It's a subscription service. And it's resulting in huge amounts of bandwidth within those networks. And so it may be that Comcast isn't so concerned about the peering relationship. They just want money.

Leo: There's money changing hands here. We want our cut of it.

Steve: Yeah.

Leo: There's also the issue of SLAs, of Service Level Agreements. Apparently Netflix Open Connect doesn't provide that, whereas a deal with Cogent might. There's also the issue of peering and commercial interconnects are somewhat different. A peering is that kind of utopian hippie thing of we'll just connect. But there are commercial interconnect relationships that are paid relationships. We're going to - it's very complicated.

Steve: It is.

Leo: Dan Rayburn, who is an analyst at Frost & Sullivan, has a blog in a magazine called Streaming Media. And he says the media's getting this very wrong, and it has nothing to do with Net Neutrality. We're going to have him on This Week in Google tomorrow. And I am going to try to get somebody who understands all of this stuff, as well. I'm thinking Jay Adelson, who ran Revision3 for a long time, but was a principal at Equinix and is an expert on this because he provided these kinds of services.

Steve: That would be great.

Leo: Because I think this is complicated. And I think we need to understand. Do you think that what Comcast is doing, demanding money of Netflix and apparently of Cogent, and Verizon is doing the same...

Steve: Verizon and Cogent are fighting. And it's the Verizon-Cogent peering points which are overloading. And Cogent is upset because they say Verizon has equipment already installed, routers that are dark. They will not turn them on.

Leo: Of course not.

Steve: All they have to do is turn them on, and there won't be this problem. But Verizon wants there to be a problem in order to force...

Leo: And that's my question. Netflix themselves - and they've been spinning this that it's not a bad thing. But of course they want to keep Comcast happy. That's access to more than 30 million of their customers. Netflix - what was I going to say? I forgot now. It's so complicated, I can't even - oh, Netflix said that, since October, bandwidth to Comcast has dropped, throughput to Comcast has dropped 27%. Now, it could be that their viewership has gone up 27% amongst Comcast customers. But it seems more likely to me that Comcast is turning a knob.

Steve: Yeah.

Leo: Right? And we've seen evidence that Comcast is throttling Netflix.

Steve: Well, and remember the other problem here is that some of these major providers have their own streaming services.

Leo: Comcast has XFINITY, that's right.

Steve: Yup. And Verizon's got RedboX II or something like that. So here's a problem, is that they would rather that customers were upset with Netflix and saw, oh, look, Verizon's own service doesn't have stuttering problems.

Leo: Right.

Steve: Maybe I'll switch to that. And so this is where the Net Neutrality side comes in. It's like, wait a minute. So Verizon is essentially causing problems for the competing media supplier.

Leo: Well, that's what I'm assuming. But then I don't fully understand how all these

paid interconnects and peering arrangements typically work. And so it's obviously above my pay grade. But I thank you for helping understand it. Do you think that what's going on is, I mean, it's interesting this all happened after the Supreme Court said that the FCC cannot impose Net Neutrality.

Steve: I guess here's the question for tomorrow, if you can think to ask it: Why won't Comcast and Verizon allow Netflix's Open Connect solution?

Leo: Why aren't - yeah, that's exactly the question I want. Why are they not using Open Connect?

Steve: Everybody else is, and it has then been a non-problem. Because then you move one copy of the movie over into Comcast, and it serves as many copies as people want. I mean, that seems like - it seems like a done deal. So what's the problem? Why is that not okay for those two? And is it anything more than profit? And my god, we're in trouble if this merger goes through with Comcast and Time Warner. It's like, oh, ho, ho. They've got enough power now.

Leo: Well, we're going to try to cut through this a little bit, too, tomorrow.

Steve: Neat.

Leo: I fear that I, in my hatred, my red burning fury against Comcast, have perhaps not reported this accurately. So I want to make sure that we are fair in how we report this.

Steve: Yeah, yeah. I will definitely watch tomorrow's This Week in Google. Sounds great.

Leo: Good, good.

Steve: Okay. So I don't know - I don't have a lot to say about Mt. Gox because this is sort of - we're all sort of in a wait-and-see pattern. But apparently they're dead. What's happened is that there's something called the "Crisis Strategy Draft" document has been floating around within the bitcoin forums. And this is something from Mt. Gox that got loose. If you go to Mt. Gox, MtGox.com right now...

Leo: Nothing there, baby.

Steve: There's nothing. It says: "Dear MtGox Customers: In the event of recent news reports and the potential repercussions on MtGox's operations and the market, a decision was taken to close all transactions for the time being in order to protect the site and our users. We will be closely monitoring the situation and will react accordingly. Best regards, MtGox Team." Now, what we also hear is their claim that - and this is what's hard to

understand. They say that the company had lost 744,408 bitcoins in a theft that had gone unnoticed for years. Now, how is that possible? I mean, nearly three quarters of a million bitcoins, that's 6% of the entire current outstanding 12.4 million bitcoins that are currently minted and in circulation. So it's like, okay. I just - how can, I mean, it's a computer. I mean, it is itself the definition of an automated ledger. So how do you have 744,408 bitcoins that were stolen years ago that you didn't notice? I just - I don't understand at all.

Leo: And there are a great number of people who have bitcoins inside Mt. Gox, and they may never see those again. I mean, we don't...

Steve: Yes.

Leo: Mt. Gox may declare bankruptcy. We're not sure what's going to happen. It's a Japanese company. The Japanese regulators have declined to get involved at all.

Steve: Yes. In a statement, some of the CEOs of the alternative and competing exchanges called it a "tragic violation of the trust of users of Mt. Gox" resulting from "one company's abhorrent actions" and, they're saying, "does not reflect the resilience or value of bitcoin and the digital currency industry."

Leo: I beg to differ, my friend.

Steve: Yeah. Now, I will say, though, in poking around, I went to Blockchain.info. And my god, that's fascinating, Blockchain.info. And you look, and it's like the most recent transactions. And then they've got like a little breakdown over there on the right, in the lower right, where it's like the most recent large ones, or the most recent...

Leo: These are huge. People are exchanging thousands of bitcoins. Here's a 10,000 bitcoin transaction.

Steve: And Leo - yes. And Leo, as I'm browsing around in here through these transactions, I'm thinking, stuff is going on. I mean, something is happening. And you just have to think that governments are looking at this, scratching their head, going, okay, I mean, something is happening here. Something is going on. I mean, it really is alive and functioning, even though it's like a rollercoaster ride. It's like stuff's happening. It's just fascinating to me.

Leo: So I always used to go to Mt. Gox just to see what a bitcoin is worth. Where do I go now?

Steve: It got pushed down below $500 for the first time, I mean, just today, in the wake of this Mt. Gox collapse. Bitcoin finally dropped from its heights at 1,200 down into sub-500.

Leo: Why would anyone put bitcoins in Mt. Gox? It's not a bank. You can have your own bitcoin wallet.

Steve: Yes. And people have all been asking me, where are your 50 bitcoins, Steve? And I just say they're offline. They're not in any computer. They're in a wallet stored on a thumb drive that is sitting in a drawer. I mean, it's just like no one should, I mean, the only thing you would do is you would move them into Mt. Gox in order to liquidate them.

Leo: As part of a liquidation, right.

Steve: Yes, exactly. You just don't want to have them sitting there, stored there. I mean, convenience is the only reason you would do it. And of course with convenience, as we always know, comes a problem with security.

Leo: So it does seem to have stabilized a little bit. It was down to, like, a hundred bucks. But I think it's at now $509 per bitcoin.

Steve: Yeah. I mean, my sense is we're in early days. There's other news. There's something called SecondMarket, I think they are, that are - they're talking to major regulated U.S. banks because banks are beginning to evidence some interest in participating in bitcoins. There were some studies that came out saying, yeah, you know, there may be something to this. So I just think it's interesting as a strange phenomenon that we're all living through right now.

Leo: So what do you recommend? I guess do what Steve does, keep your bitcoin in an offline thumb drive somewhere. It's just bits; right?

Steve: Yup. I would not leave them sitting anywhere because you need to have control of them. And this is a currency that allows you to have control. You can have your money in a bank. It's the reason in the U.S. we have the FDIC that protects accounts up to $150,000, so that you're safe with your money being there. But there's nothing like that for bitcoins. So don't leave them sitting in an exchange. Pull them out back into your own wallet, and only expose as much of your bitcoinage as you feel secure doing.

Leo: So you still have faith in bitcoin. You're not giving up on bitcoin.

Steve: I guess I have no horse in the race. I've got 50 bitcoins.

Leo: You have a horse in the race, my friend. That was worth 50 grand a little while ago. Now it's worth half that.

Steve: Yeah. I don't care. I mean, I didn't do anything to earn them. They just spontaneously appeared in my computer. So it's like, I mean, I guess I'm interested

because I own some. But overall, my sense is virtual currency is real. The question is, what will its future be relative to governments. Governments are the big unknown. And it's just difficult to understand how governments are going to allow this to remain lawful for long. I mean, they've been outlawed in Russia now, bitcoin has.

Leo: Right. I also think it's - I would discourage people from speculating in bitcoin because it does seem like it's somewhat risky at this point.

Steve: Oh, it's not for the faint of heart, no. So, stats on admin or non-admin. Everybody knows that an administrative account is dangerous relative to a standard user account. I'm talking about Windows now. Since the dawn of time it's been understood that the root user in UNIX and then Linux is super powerful, and that you should not run as root. You should run as a non-root user so that, if something happens, the OS will prevent you from doing things, you or a surrogate running in your account behind your back, prevent you from doing dangerous things. Windows has that concept, too. But it's lax about enforcing it because it's inconvenient. And many people run as admin, and they just try to be careful. Many other people who have people that care about them have set them up as a standard user so that they are more safe. The question is, how much more?

A research company, Avecto, A-v-e-c-t-o, did an analysis of last year's Patch Tuesday vulnerabilities. They titled it "Mitigating Risk by Removing User Privileges." And the summary is: "Analysis of Microsoft Security Bulletins from 2013 highlights that 92% of critical vulnerabilities would be mitigated by removing admin rights."

Leo: Wow.

Steve: 92%.

Leo: Holy cow. See, I thought that user privilege escalation meant that, even if you were operating as admin, you would have to allow escalation to do admin-like things.

Steve: It turns out that's, I mean, that's a compromise Microsoft made. But bad stuff can still be done behind your back. What you want is you want to go to...

Leo: It could be poorly implemented.

Steve: You want to go to the - well, it's, again, nobody really wants the lack of convenience, of just being able to do what you want to do on your computer. The problem is the lesson we learn is that, unfortunately, stuff can be done on your behalf without your knowledge. So...

Leo: With your permissions.

Steve: Yes.

Leo: Using your permissions.

Steve: So last October 2013, it passed by us, and we didn't take note of it, so I want to. That was the 10th anniversary of Microsoft's Auto Update system.

Leo: 10 years, wow.

Steve: 10 years. And before the podcast began, by a year or two, that was in place. And of course very controversial at the time. People were, I don't want Microsoft automatically updating my system because it used to be we would have to go get those updates and install them ourselves, or they would be issued as service packs, not just this continuous dribbling dribble into our machines. But it's been 10 years. So here's the breakdown. During that year, 2013 of critical rating, so there were 147 vulnerabilities published during 2013 with critical rating. 92, as I said, were mitigated, blocked, by removing admin rights. I'm sorry, not 92, 92% were blocked by removing administrator rights. 96% of critical vulnerabilities affecting the Windows operating system, so nearly all, 96% of those vulnerabilities which affected the Windows OS were mitigated by removing admin rights. 100% of the vulnerabilities affecting IE were mitigated by removing admin rights.

Leo: Wow.

Steve: 100%. All you had to do is switch to a standard user. In the control panel, under Windows Users, you have a choice, be an admin user or a standard user. And unfortunately, by default, when you set Windows up, you're an admin user. That's what you get. So you need to create another user, set that up as a standard user, and that's the one you use. And then, when you need to do something that you're being blocked by, you need to enter the admin user's password. That's the way to be safe. Not even UAC gives you this level of safety. You need to be a standard user and then provide the admin password when you need to switch into the admin account, essentially. 91% of vulnerabilities affecting Microsoft Office would be blocked by removing admin rights and 100%, all of the critical remote code execution vulnerabilities, and 80% of critical information disclosure vulnerabilities mitigated by removing admin rights.

So the takeaway here is this is really important. If you simply stop being an admin, if history is any lesson, you're way safer. You are completely safe based on history from IE exploits, and those are the big way things get in is through Internet Explorer, through web browsing. And critical remote code execution is also how this stuff happens. 100% safe if you're not an admin. So we've got 41 days to go with XP. Certainly XP users ought to seriously consider no longer running as an administrator. Just run as a standard user, and use admin account only when you really know you need to.

Leo: You think that would make a difference?

Steve: I think it would really make a difference.

Leo: Yeah. So maybe - you still want to give up XP, but...

Steve: I really think, I mean, I'm going to be very reluctant because of the huge investment I have. I mean, just the idea of rebuilding this just makes me shudder. On the other hand, it does create an opportunity to get rid of - it's like moving to a new home and looking at all the things you think, well, do I really need this? Eh, I don't think so. We'll do a little spring cleaning in the process.

I did want to mention that just yesterday there was news of YouTube distributing ad-based malware. Google is on the job now, figuring out how this has happened. But YouTube is so popular that it receives in a month's time a billion unique visitors, spending about six billion hours every month on YouTube. It turns out that their instream ads were redirecting users to malicious websites and hosting the Styx exploit kit, which was leveraging a more-than-year-old known Java - not JavaScript, Java - vulnerability to infect users' computers with the Caphaw banking trojan.

And so the researchers who discovered this said: "We don't yet know the exact bypass the attackers used to evade Google's internal advertisement security checks. Google has informed us that they're conducting a full investigation of this abuse and will take appropriate measures." So that's good.

Now, I tweeted this immediately after watching it last week, and I wanted to bring it to the attention of our listeners who aren't following me in Twitter or may have missed it. This was posted late last year. Or, no, I'm sorry, longer ago than that. I've got October 23, 2012. But this was an interview of Steve Jobs which was excerpted from - and the excerpt was used, I think it was in Robert Cringely's "Heroes of Silicon Valley." I don't remember now. But he posted - essentially the entire interview was believed to be lost. And believe it or not, a VHS tape copy was, seven years later, found in someone's garage. And so I got a copy when I saw it. Maybe it was that it just came out on disk. Anyway, I'm not sure why it just crossed my radar recently.

But so it's called "Steve Jobs: The Lost Interview." It is available from YouTube for a few dollars, I think maybe $4. You can buy the disk. It's on iTunes also. I really recommend it. This was Steve Jobs in 2005, relatively recently chastened from having been booted out of Apple by John Sculley. What is significant to me is this is not the typical Steve Jobs ego that we all know and have seen interviewed and talking. And I was really surprised by the wisdom in here, I mean, by what Steve understood and was able to articulate about the way groups operate and the way you produce excellent products. I mean, frankly, I found a lot of my own philosophy of the way things are done right in what he said. Anyway, it was a fabulous piece of work, and I recommend it without reservation.

Leo: The reason - I've seen it, too. The reason for the data confusion is it was released theatrically for a brief period of time in 2012.

Steve: Ah, okay.

Leo: Yeah. But it's nice that you can get this now on YouTube and watch it yourself.

Steve: Anyway, I really, really, really enjoyed it. And Leo, I don't know if I picked up on

this from listening to one of your mentions somewhere. But there is an app called Rails for iOS? For the tablet? And oh, my god. It is...

Leo: I think it was a MacBreak Weekly pick, perhaps.

Steve: Oh. It is just too fun. I'm finally - my addiction has waned a little bit. I think it's 2.99. I created a bit.ly shortcut because sometimes I'm very unimpressed with the App Store's search engine.

Leo: Oh, it can't find anything.

Steve: It's awful. So this is bit.ly/sgrails, r-a-i-l-s. And what I did was I put out a warning: Do not get this. Because it will just take you over. You will not be able to stop playing with it. It is just too fun. So whatever you do, do not get Rails.

Leo: iPhone or iPad?

Steve: Not iPhone because it's too small. iPad only. SGrails is the extension on bit.ly, bit.ly/sgrails. And oh, my goodness, it's just, for me, it's a perfect puzzle because you...

Leo: Oh, it's railroad tracks. That's nice.

Steve: Yes. And the stations are emitting trains periodically, and you have to quickly design a track system which is capable of routing the trains among the stations without them colliding. And it's just wonderful. So it's well regarded, well rated. I gave it five stars.

Leo: Three bucks.

Steve: Three dollars. And, boy, it's the best three dollars you could spend, if you just want something - I'm not a - I don't waste time playing with my tablet. I've got too much going on. But I just - I stopped reading. I've stopped reading in order to spend some time with this thing. It's just really fun.

Leo: This is how Steve's mind works, is like these rails.

Steve: And speaking of which, just late last night I finished the user interface walkthrough of SQRL's Create New Identity. That's what I've been working on. I created the operation page. It's GRC.com/sqrl/operation, or as I mentioned, the bit.ly URL I created was bit.ly/sqrlui, SQRL UI, sqrlui. And if anyone is curious, it is a nice way of looking at how SQRL operates from the user's angle. It's funny, too, because this has been three weeks. And when I looked back, when I created the page was January 30th. And I first put it public on February 2nd. And so here we are a little more than 21 days

later, and I thought, three weeks? How did I spend three weeks on this? But then I remembered that it's because, as I was being forced to turn the technology into a user experience, I realized, oh, crap, this is too complicated. And I went back, making substantial revisions to the technology, in order to make the user interface work the way it should, losing nothing, and in fact improving the security overall in the meantime, and sort of making them agree. And so anyway, it forced some changes.

That's done now. The entire process of creating a new identity in SQRL is laid out. And I'm not sure what part of it I'm going to do next. But I'm going to continue, I'm going to basically finish now with the other bits of the UI, and then start writing code. So if anyone's curious, GRC.com - actually you can just find it. It's in the SQRL pages. It's page no. 6 from the big block of links at the bottom. And we're making progress, working as fast as I can, getting SQRL done for everybody.

Leo: Stop playing Rails, Steve. Now you've got me hooked.

Steve: Oh, believe me, if you start, you can't stop. This is worse than potato chips.

Leo: Somebody said $3 is not the true cost of the program.

Steve: That's true.

Leo: Yes, indeed. How fun. Oh, well. Oh, well. It's kind of cool, too. I mean, it looks like a pretty sweet little program.

Steve: Oh, it's - it is, so, yeah. So there's a tutorial that explains it. And you have to go to New Game and then tap Tutorial because that's the first one you'll get. And the idea is you draw the completion of the rail, but there's actually a huge network of rails that you can - it's of course tracks and switches. So you're having to, like, flip the switches. But you get to design how the stations are interconnected. And as you were drawing it right there, you could see lit up behind you was all the possibilities of where you could draw track.

Leo: Oh, my.

Steve: Oh, it's just - and a couple times I've just done some inspired layouts.

Leo: I can't wait.

Steve: It really is fun. It really is fun.

Leo: Oh, you got me. You got me, Steve. We do this show, well, maybe we won't be back next week. No. We do this show every Tuesday, 11:00 a.m. Pacific, 2:00 p.m.

Eastern, 19:00 UTC. Join us Tuesdays. I'm sorry, 1:00 p.m. Not 11:00 a.m. 1:00 p.m. Pacific, 4:00 p.m. Eastern, and that would be 21:00 UTC, right after MacBreak Weekly, right before Before You Buy. And if you want to watch live, do. But if you don't, if you can't, we've got on-demand versions. Now, Steve does some 16Kb audio versions for people with - what does your shirt say? Science? What does that say?

Steve: Science.

Leo: "Science: It's Gotten Us This Far." [Laughing] I love it. You leaned back, it's the first time I've seen the text. You had to show us that "Science: It's Gotten Us This Far."

Steve: I have one that I like, it's my cranky guy shirt. It just says "No."

Leo: Yes. And then you have the other one that says, "No, I will not fix your computer." So you really...

Steve: Right.

Leo: You're covered with those three shirts.

Steve: This is more general just because it's like, your faucet's dripping, you know, so forth.

Leo: No, no, no, not gonna do it. Steve has 16Kb versions of the audio. He has transcripts there. It's all at GRC.com. That's his site, Gibson Research Corporation. While you're there you should check out everything else Steve's got going on. Oh, I'm sorry, I've got to change the railroad signal. Waiting for the train to enter the station. Okay, go ahead, enter the - oh, now it's going back the other way. Aw.

Steve: That's okay. It'll come back out.

Leo: It'll come back? Okay.

Steve: If it hits a spot where the switch is wrong. There's also some little red and green signals you're able to turn on and off.

Leo: Uh-huh. Yeah, that's what happened is I accidentally turned that signal, and then the train - oh, no. Oh, oh, oh goodness, I've made all sorts of - forget it. Forget I - this is kind of...

Steve: You're going to have too much fun.

Leo: We have audio and video on demand after the fact, high-quality audio and video on demand after the fact, if you visit TWiT.tv/sn for Security Now!. And, by the way, do visit Steve's site because he's got all sorts of great freebies and, you know, stuff about passwords and diet and everything: GRC.com. And then of course while you're there you might as well...

Steve: Yeah, actually I forgot to mention I'm in the process of doing a cholesterol sequestration experiment.

Leo: What the hell is that?

Steve: I'll talk about it next week. I didn't have my - I did blood tests on Friday after two weeks of experimenting on a new idea of using pistachios to lower cholesterol.

Leo: [Laughing]

Steve: I kid you not.

Leo: His body is a temple AND a test tube. We also encourage you to buy SpinRite, world's finest hard drive maintenance and recovery utility, because that's Steve's bread and butter. That's how he puts two pieces of bread together, and he puts a piece...

Steve: And it's good for you.

Leo: And it's good for you.

Steve: Just like pistachios.

Leo: Yes, indeed. Thanks, Steve. We'll see you next week on Security Now!.

Steve: Thanks, Leo.