Poodle Bites

Description: After catching up with a few interesting events from the past week, Steve and Leo take a deep dive into the details of the Internet's latest security "catastrophe" which has been named "Poodle." Steve first carefully explains the trouble, then debunks it completely, showing why the vulnerability should be fixed but will probably never be exploited.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-478.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-478-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. You've probably heard about the "frufarah," the folderol, the fracas around this new exploit called Poodle. Steve says it's just a load of mutton. The latest on Poodle and all the security news, next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 478, recorded October 21st, 2014: Poodle Bites.

It's time for Security Now!, the show where we cover your security, your privacy online. We talk about hacks, exploits, attackers, the new word Steve's going to use for bad guys on the 'Net.

Steve Gibson: Yup.

Leo: Steve Gibson's here. That's the Steve I'm talking about, of GRC.com, the creator of SpinRite and also a father of spyware. Not that he made it, he discovered it and created the first antispyware tools. It's good to see you once again, Steve. Ten years we've been doing this show. Almost.

Steve: Feels like just the other day you were suggesting this. And I thought, what? I don't think we have enough to talk about. In fact...

Leo: Second show we did on the TWiT network.

Steve: We have so much to talk about we don't even get to our regularly scheduled stuff anymore. It's like, this week was supposed to be a Q&A to follow up on last week's discussion of the tokenizing purchasing system. And one of my little notes here is just to mention that, of course, Apple Pay went live yesterday. And in fact my phone, I went over, and it was saying, hey, you can update to 8.1 at any time. And I thought, well, okay, I'll do it now. So it's over there doing that.

But what happened was we talked - I mentioned sort of tangentially last week that there were rumors of something that was going to be disclosed, I think I said noon Pacific time on Wednesday, that was, like, synchronized with some particular time in Europe, I don't remember what or why. But it didn't last that long. It was just too big. And so it leaked out later in the day on Tuesday. And that's the so-called and annoyingly named "Poodle" exploit. You know, 2014, looking back on it, I hope we're able to look back on it...

Leo: What a bad year, huh? God.

Steve: It's been a rough year. Yeah, boy. Heartbleed, and what was the one we just had? I'm blanking on it. Shellshock. Heartbleed and Shellshock, and now we have Poodle.

Leo: Oy oy oy.

Steve: I'm excited about this podcast because this is a really interesting problem which pulls a lot of the different things that we've discussed over the years together. People who have not managed to survive with us over the years or, that is, who haven't been here that long, will still be able to follow along. But what I realized when I was first researching it last week, was that it was nonsense. It's like, whoa, okay, wait a minute. It turns out that there is a problem, but nobody would ever attack you that way. So, and by the end of the podcast, everyone will understand what I mean by that because it's really not that complicated. But it's got lots of interesting moving parts and details which is the kind of stuff we like to do here. So I think everyone is going to enjoy the next hour and a half or so. And there really wasn't, other than that, that much news. But we'll get to that and then dig into Poodle Puddles.

Leo: All right, Steve.

Steve: Okay. So, like I said, not a lot of news this week. We did have, I just sort of wanted to mention because a lot of people tweeted it, unfortunately we've got the FBI guy, James Comey, who is much in the news lately, now grumbling at his first official speaking engagement at the Brookings Institution in Washington. And this is like his first major policy speech, even though he's been at the FBI for 13 months. I guess someone said, "Come talk to us." And so he's complaining, not surprisingly, because we've been hearing grumbles of this for the last few weeks, about the encryption, the enhanced encryption in Apple and Google products and now, as expected, beginning to make noises about maybe it's time for a legislative "fix," in quotes.

Leo: Oh, oh, a backdoor.

Steve: Exactly. And of course our listeners will know that I felt this happening a couple years ago, which is why I stopped working on CryptoLink, was because it just felt to me like the way the country was going there was going to be some legislation, or if nothing else we were going to go through a painful period. And we're not in it yet. We're approaching it. The Huffington Post covered this, and I liked their reporting of it. They said that Comey said he understood the, quote - and I love this, this is like poll tested - "the 'justifiable surprise' many Americans felt after former National Security Agency contractor Edward Snowden's disclosures about mass government surveillance." Yes, we were justifiably surprised, Leo.

But he said, the Huffington Post said he contends "that recent shifts by companies like Apple and Google to make data stored on cell phones inaccessible to law enforcement" have gone too far. Comey said, quote: "Perhaps it's time to suggest that the post-Snowden pendulum has swung too far in one direction" - okay, now, I would argue it's still on its upswing, away from center, where it had been. And he said: "...in a direction of fear and mistrust. Justice, he said, may be denied because of a locked phone or an encrypted hard drive." And when I saw that I got a little chill. It's like whoa, ho, okay, what, what? An encrypted hard drive, uh-huh.

So Comey said that the FBI was seeing, quote, "more and more cases," unquote, in which law enforcement officials believe there was significant evidence on a laptop or phone which they couldn't access - so we are talking about hard drive encryption, too - due to encryption. It's not clear, however, that any of the cases he specifically referenced, and apparently he talked about a murder in Louisiana and a hit-and-run homicide in California, that they could not have been solved with a traditional warrant to cellular service providers.

And then, happily, Matthew Green, our assistant crypto professor at Johns Hopkins who's been much active recently, he was approached by the Huffington Post for his reaction to this, and he said: "Law enforcement has access to more data than they've ever had. As a society we're just finally trying to get back to a point where it's a little more in line with what law enforcement would have been able to get back in the '80s," you know, meaning that we have to have some sense of balance here.

And then on background the Huffington Post finished by saying: "Snowden's revelations have provided a crisis abroad for major U.S. tech companies, which could lose billions as foreign customers leery of American software and devices compromised by the NSA turn to other providers. Comey said that he was, quote, 'not trying to jump on the companies,' like Apple and Google, that implemented encryption systems closed off to law enforcement and that he believed they were, quote, 'responding to a marketing imperative.'"

So anyway, it's just, okay, I mean, back when the whole Snowden revelations occurred, one of the things we said on this podcast was that this was going to happen, that math is fundamentally unbreakable. We have unbreakable math. And the fact that we've been maybe somewhat lackadaisical in deploying it or enforcing it doesn't mean that it's not available to us. And it really hasn't taken long at all. What, a year? Because I think it was just about a year ago. And look at, you know, the terrain from the protection of the consumer's privacy today looks very different than it did a year ago.

And all of us would argue, I mean, I heard you on MacBreak Weekly covering the China government hacking story, where they were basically using a weak browser to get man-in-the-middle interception of their citizens' access to iCloud and using it to capture their usernames and passwords in order to decrypt their data. And so it's not anti-law enforcement. I mean, I understand from the FBI position that's their bias and the way

they see it. But it's truly as much protection against foreign governments and attackers as it is against law enforcement. I mean, it is just privacy that the math makes possible.

So anyway, we were talking before we recorded...

Leo: Yes, [indiscernible].

Steve: Yes, you and I both received envelopes from Yubico, our friend Stina Ehrensvrd at Yubico. My Twitter feed went crazy starting around midnight last night. Google posted on their blog, and Yubico sent out coordinated news and also provided you and me both with two of their latest toys. Yeah, you've got the little blue one right there, and then there's a little tiny one called the Neo-n.

Okay. So Stina and the Yubico engineers have been working with - of course we know that they have a longtime past affiliation with Google because it's one of the reasons that Stina moved Yubico from Sweden over to the peninsula was so she could be here in Silicon Valley in the hotbed of all this. And of course Google is a major player here. So she's been working with them, and she's also - I think I read that she's on the board of directors now of the so-called FIDO Alliance. And the FIDO Alliance is some hundred-plus companies that are all gathered together to try to arrive at an open standard for sort of next-generation Internet authentication. All of their work is device-based, that is, like the Yubico YubiKey.

And I guess I would characterize it as like sort of heavyweight. I mean, it is a - there's two specifications in FIDO, and U2F is the lightweight, sort of possible to actually implement specification, as opposed to - I don't even remember what the acronym for the other one is. But that one is so complex that only one company I'm aware of has got it working, and they're the company that helped to write the spec. And, you know, they like to sell stuff. And so it's more in their interest to keep it complicated and license their software than it is to make it open and easy.

And by comparison, because people will say, well, how does FIDO compare to the work I've been doing for the last year on SQRL, SQRL is like super lightweight. You can explain it quickly and easily and implement it quickly and easily. It can use hardware, but it doesn't, it's not tied to hardware, where the FIDO stuff always will be. So there's certainly room for more than one solution in the industry, and we'll see how all this pans out.

But so what Yubico has now is the most inexpensive solution they've ever offered. That blue key - the blue pill - the blue key you are holding, Leo, is only $18 on Amazon. And if you're a Prime Member, shipping is free, so it's $18. And that will not do all of the things we've talked about before with Yubico. That is, it's not a one-time password, touch the thing and it emits a string of sequentially encrypted one-time password tokens.

Leo: Because it looks exactly the same as those.

Steve: It does.

Leo: It's going to be hard not to confuse them.

Steve: It's pretty much, well, and the blue color, that's how you know, is it's blue. But it's also much less expensive than their prior technology. But all it does is the FIDO U2F, which currently is supported by Google and Chrome, but being an open standard can spread if it's going to. So, and knowing Stina, it'll be spreading.

Leo: Yeah, see, because here's the old YubiKey.

Steve: Yup.

Leo: It's black. It does have a little key on it.

Steve: As long as you have a color monitor, or eyes, or I guess color eyes...

Leo: But they also put a little key on the button so that you know it's a little different.

Steve: That's true, yeah, so you can - if you can't tell the difference. So that one's inexpensive and does U2F. Their newest product is the - oh, and that's called the Security Key. Google uses the term. I didn't see Yubico mentioned anywhere on Google's page, which I thought was a little bit - I don't know.

Leo: Are they the only company doing it?

Steve: No, actually. I think I noticed there was one other...

Leo: Yubico, they say in the letter they're on the whatever, the panel, the board, the...

Steve: Yeah, yeah, exactly. Anyway, so there's the - I think it's the Neo and the Neo-n are the two other new technologies. The NEO is 50 bucks, and the Neo-n is 60. And the Neo-n, the "n" of the Neo-n stands for NFC. So that gives you near field communications technology. And both of those are these cute little almost square things that are just sort of like just the plug part of a USB, sort of like you took the YubiKey and snapped it off like where it inserts. Anyway, those are both available. And they do, not only the new FIDO U2F, but all the other traditional Yubico protocols - one-time password, something called JavaCard which is another standard, and a couple other standards.

So they're very much standards-based. And where necessary, Stina goes and creates new standards, like she has with basically the bifurcation of FIDO into U2F, which it's actually possible to use, and the other thing that's not off the ground yet because it's like the Spruce Goose of authentication. So anyway, if you just go to Yubico, Y-U-B-I-C-O, dotcom, which is what I did this morning, and click on Products, it takes you to a nice grid where you can see the lineup of the various hardware offerings, the suggested retail pricing, and then what protocols each one supports and so forth. So the good - yeah, then click on Products up there in the menu.

Leo: I did. These are - this is where you go.

Steve: Oh, but scroll down. Oh, I'm sorry, on Product, then Hardware.

Leo: See? Yeah, that's where I am.

Steve: There it is. Okay.

Leo: Ah, here we go.

Steve: Perfect. Very nice.

Leo: Somebody said Neo-n does not have NFC, just Neo.

Steve: Uh-oh, wait. Really?

Leo: Well, if you trust the chatroom.

Steve: Okay. Well...

Leo: I'm looking at this, and I don't see NFC on the Neo-n, just on the Neo.

Steve: Oh, it does - so it does show it on the - that's weird. So the Neo is less expensive than the Neo-n. I don't know, I guess I don't understand that. Anyway, that's why I recommended people go there, because the grid is comprehensive.

Leo: I think the only one that has NFC is the Neo. The Neo-n does not.

Steve: Okay. Maybe that's what the "n" stands for, "not." No NFC.

Leo: No NFC, not NFC, yeah.

Steve: Anyway, that's pretty much all of our news for the week. I did want to just sort of - I noted over the last couple weeks that I was getting many more sets of four yabba-dabba-dos. And I had noted over the summer that that seemed to have sort of disappeared. And of course the difference is an individual license for SpinRite generates a single yabba-dabba-do when a purchaser obtains a license.

And the way we operate is that corporations can get a license for all of the machines in a

single physical site, like whatever, regardless of how large the site is, by having four licenses. And that just seemed, when I was coming up with the policy, a much simpler way of organizing things because I thought someone might want to try it and see if it works, and then they would have bought a license for one. So then if they wanted a site license, you have to have some sort of a site license for people who already owned one, or refund their purchase and then issue them a site license, and it just seemed like a mess. And so I liked the idea of just having X number of regular licenses.

And then it also kind of was really cool with upgrades because if we then had a paid-for upgrade, then they could upgrade their site license by upgrading their individual licenses. Anyway, just so the idea is that when I hear four yabba-dabba-dos, somebody purchased a four-license site license. And for whatever reason, in the last couple weeks, there have been, like, they've come back.

So I just wanted to thank people because - actually one of my favorites is when I hear three because that means that someone got one, they checked it out, and then they said, okay, we want to - this thing works. We want to use it sitewide. And then they bought three more in order to have a total of four licenses and then have permission to run it on all of their machines within a facility. So anyway, thank you. I really appreciate that. That keeps the wheels turning here over at GRC and lets me do everything else I do.

Leo: Did you - I guess you explained why yabba-dabba-do and all of that. We don't hear it anymore, right, because you turn it off during the show. Every once in a while it will be on by accident.

Steve: Yeah. I mute it because it's a little distracting. But it was just - what I have is I have a system that sort of monitors the GRC servers. It's sort of like my custom version of the advertiser you just introduced us to.

Leo: PagerDuty.

Steve: PagerDuty.

Leo: PagerDuty.

Steve: So, and for example, there was a time when we were under denial-of-service attacks, more or less annoyingly frequently. And so I built a sort of a real-time bandwidth monitor and server monitor that watches all of the things, the processes and servers in our offsite facility at Level 3. And among other things, it - and it's very cool. It uses UDP. It's all custom stuff that I built. And so I'm behind multiple layers of NAT, and so nothing can get in here. But UDP, as we know, is able to return up the path that it exited.

So my system here sends out a UDP query every second or two, actually I think it's every two seconds. It sends out a UDP query just asking for an update. And that maps through all of the security that surrounds the Fortress of Solitude and Research so that the server, when it receives a query, or the system at GRC, when it receives a query, it assembles a current state and then returns a UDP reply, which UDP doesn't really expect one. It's just the idea is it might get one. But the NAT routers have like been opened by

the outgoing query. And so the servers send back a "here's where everything stands" reply.

One of the things in there is the total of SpinRite sales. And so at this end I look to see if there's been any change. And if there is, I divide that by the cost of SpinRite, which tells me how many licenses sold, and I emit that many yabba-dabba-do WAV files.

Leo: So it's modulus SpinRite cost.

Steve: It's modulus SpinRite licenses, yes.

Leo: Oh, I love it. I think we assume that everybody who listens over the years has learned this. But, you know, people still come in the chatroom and say, "What are those lights blinking over Steve's left shoulder?" So we have to assume that there are people here who are not...

Steve: What's Fred Flintstone doing in the background?

Leo: ...hip to the lore of the Fortress of Solitude.

Steve: Yeah, exactly.

Leo: Yeah.

Steve: We're going to go from not very technical and what's your Zodiac sign to "Fasten your technical seatbelts."

Leo: Uh-oh.

Steve: But I think everyone's going to enjoy this. Okay. So once again the industry suffered another shock. Much like Heartbleed and - why can't I remember the name? - Shellshock.

Leo: Shellshock. You're blanking it out.

Steve: I am. So this was - okay. And the headlines all were hyperventilating, and people were making sure on Twitter that I had seen this, and I knew what was going on. And the scary headline title is that there's a new problem with secure connections involving a means of making browsers and servers use SSL v3, and then leveraging a vulnerability in that in order to crack the security of SSL v3.

So immediately, sites popped up on the 'Net to allow people to check whether remote websites had responded to this Poodle problem. And SSL Labs, of course, quickly added a

test to allow anyone to check. I started getting people saying, [gasp], "Oh, Steve, GRC is vulnerable." And it's like, okay, everybody. First of all, GRC is not vulnerable. Never has been a problem. I'll explain that at the end of the podcast, why this is not, I mean, independent of this, why it's not a concern for the way I implemented things.

But so here's the story. Here is exactly what's going on and why, despite all of this, and the fact that none of that is wrong, it's actually not a problem. The fact, well, I don't want to step on myself, so I'll take us through this. So what's going on? As we know, SSL has had problems through time. It was originally created by well-meaning smart guys at Netscape in order to create a secure link between browsers and servers so that we could do things like have usernames and passwords and cookies that were not in the clear, because before that everything was in the clear. It was like email is pretty much today. Just there, there go the bytes. If you're sniffing the wire, you can see them go by.

So SSL of course started off at 1.0 and has been incrementing sort of slowly and in various amounts over time as problems have been found and fixed. And we finally got up to the point where we were ready to go to 3.0, and someone decided, let's change the name. And it's like, oh, really? Okay, fine. And that decision never comes off very well. And in fact that's been a problem because we have SSL v3.0, and then TLS v1.0. But TLS v1.0 is newer than SSL v3.0. So not only did we change the name, but we reset the version number. And so that confuses people.

But then we've moved with TLS. SSL was Secure Socket Layer, that's what the acronym SSL stands for, because in UNIX world, UNIX thinks in terms of communication sockets. That's the name of the abstraction for communicating between two endpoints on the Internet. You create a socket, and then you connect to another socket on a different machine, and then they talk to each other. So Secure Socket Layer is SSL. TLS is Transport Layer Security. So we have a new acronym for the same thing, just a newer version of the same thing.

Leo: This could all be part of our new show on the TWiT Network, Acronym Olympics.

Steve: Acronym Soup.

Leo: Yes. Really we should do that. It'd be a good thing to do, like just give people acronyms and say, "Can you define this?" Because it's crazy.

Steve: You know, I'll bet it would be possible to do an entire podcast where you simply bring...

Leo: No English?

Steve: ...bring acronyms together with small conjunctions where the RC4 and the CBC of the SSL and the TLS...

Leo: You could.

Steve: ...and so forth, yeah.

Leo: You could, definitely.

Steve: So, and actually having listened to Andy Ihnatko on MacBreak Weekly talk about how he wants to use NASA portable audio...

Leo: As his ringtone, as his ringtone.

Steve: It's like, have you ever heard of anything more geeky? I mean...

Leo: I think he's the king.

Steve: That's my point exactly, yes.

Leo: The king of the geeks.

Steve: Yes. Yow.

Leo: Yow.

Steve: So, okay. So TLS is where we are now. And we went, of course, we started at 1.0, 1.1. We're now at 1.2. So there's always a problem as we are moving standards forward with systems that aren't advancing. And it's, like, not good not to advance, but it's the reality. So it turned out, when we moved forward to TLS, and clients, that is, users with Windows, Linux, and Mac, and smartphones and so forth, had clients that were initiating state-of-the-art, modern, recently updated, refreshed connections. They would connect to a server and say, "Hi there. I know about TLS 1.1." Because 1.2 is really very much newer. So probably 1.1. Maybe, well, certainly 1.2 now. And there were some servers that said, "Huh?" And hung up.

Again, if everything worked smoothly, there's supposed to be like a version protocol handshake. And we've discussed how SSL negotiates. The idea is that both ends of a connection advertise the highest level of the protocol that they're aware of, and they agree on the highest level they both speak. So that happens on a monotonic scale in terms of versioning. But then the client may have a different set of ciphers that it knows about. So it sends a list of all the ones it knows about to the server. And then the server browses through those and chooses, in some order, hopefully from strongest to least strong, the ones that it knows about, that it has in common. And then they agree.

So, and that's a neat theory. But it has been subject in the past to so-called "protocol downgrade attacks," various ways, I mean, again, bad guys are clever. And as we know, they only get cleverer. So we need to protect ourselves against a bad guy coming in, well, I mean, a classic one in the early days, there was actually a null cipher that was in the set of ciphers because the original engineers of SSL said, hey, you know, what if

something - if you're trying to connect to a skate key or something, I mean, that has no crypto whatsoever? So maybe we should allow that. And so you could actually say, I would like to talk to you over SSL, but I don't have any ciphers. And the other end would say, oh, shoot. Well, okay. And so you'd have an SSL communication with no encryption, which really sort of defeated the whole purpose.

Leo: Just SL. There's "S" in the SL.

Steve: Exactly.

Leo: Secure, but not - insecure, ISL.

Steve: No, just the socket.

Leo: Just a socket layer, yeah.

Steve: Socket layer, but no secure socket layer. So, okay. So when we realized, we the industry, that there were lame servers that were confused if we even mentioned TLS - we'd say "TLS," and they'd just hang up. It's like, ooh, okay, I guess not. So browsers learned to, if they got hung up on when they offered TLS, to try SSL. And if they knew about TLS, they certainly knew about SSL3 because that was the end of the line of the SSL acronym. So they'd say, how about SSL3? And at least it was SSL. So then the server would go, oh, okay, yeah, fine. What have you got? And we'd go from there.

So the problem with that is that that opens us to a version downgrade. That is, if - and this is an "if" we'll be coming back to several times. If an attacker managed to get into the connection, the classic man in the middle - now, in this case, it's not just an eavesdropping connection, that is, not a passive man in the middle who can monitor, like we now know the NSA likes to do. This is an active man in the middle which is, again, it's another escalation in attack requirement where somehow the victim's client traffic is passing through the attacker, who is able to change it.

And in the initial packets, which are going back and forth during this negotiation, there have been weaknesses in the past which TLS further strengthens. But in this case all the attacker has to do is force an error, which is trivial. It's hard, actually, you've got to balance checksums and do all kinds of things, not to have an error. But all the attacker does is lead the browser to believe that they are trying to connect to one of these lame servers. And so the client will go, oh, I guess no TLS here. Fine, we'll use SSL3.

So the stage-setting portion of this is that we are, today, are subject to this protocol downgrade where a bad guy convinces the browser that the server can't do TLS. So now we do SSL3. And so the first part of this is that we're forced onto SSL3 from an active man-in-the-middle attack. Now we have SSL3 problems which we had deliberately gotten away from, moving to TLS. And SSL3 has a choice of basically two ciphers. It can use RC4 or CBC. And we've talked about both in the past. I'll give a quick review.

RC4 is a really lovely stream cipher from RSA. The problem is that, when we looked at it more closely, we discovered that the way it starts up is not secure. RC4, it's lovely because it's just elegant. I love it for its simplicity. You have two tables of 256 bytes. So,

like, basically two vectors. Or, no, I'm sorry, one vector. It's two pointers. One vector, one table of 256 one-byte entries, and two pointers into the table. And basically, when you give it the key, it scrambles the initial starting conditions into something which is based on the key. And then, as you run the cipher, it continues to basically swap bytes in the table and, at the same time, emit pseudorandom data. And you could almost imagine how, if the table wasn't really scrambled up well, then the bytes coming out wouldn't be that random.

And that's precisely the problem. If the designers had just warmed it up more, if they'd, like, run it for 256 extra bytes, then the table would have always been sufficiently scrambled that it wouldn't have been a problem. But the weakness turned out to be that the first data being emitted by the RC4 cipher, which is then XORed with the user's plaintext to create ciphertext, it wasn't as random as we needed. And in fact, even more recently, detailed additional analysis showed that it's worse than we thought. So RC4 is out. No one likes it anymore. We hope no one's using it anymore. Not only was it used, of course, famously in SSL, but even more famously in the original WiFi protocol, WEP, which is where we really saw it collapse.

Okay. So the better cipher, although not without its own problems, is CBC, which is an acronym for Cipher Block Chaining. CBC takes the data in blocks of bytes where the size of the block is the size of the cipher's block. So let me just say to remind people that RC4 is a stream cipher, meaning that it emits a stream of bytes which are unpredictable, and so you XOR those bytes with your data to get your data encrypted. And then on the other end you give it the same key. It generates the same stream of pseudorandom key-based bytes, which are unpredictable. You XOR those with the ciphertext, and out pops the original plaintext. So very neat and elegant.

There are problems with a simple XOR cipher, though, such as you absolutely have to make sure you never use the same stream twice because then the whole thing falls apart in interesting ways that we have talked about in prior podcasts. By comparison, cipher block chaining uses a block cipher like AES, the Rijndael cipher that we're using now, which is 128-bit block.

The Rijndael cipher, and if anyone isn't familiar with it we did a beautiful podcast on it [SN-033] on AES some years ago when Rijndael had been chosen as the Advanced Encryption Standard (AES) cipher for the industry, and that was an NIST-based competition that looked at all the various candidates and said, okay, we like this for - it meets all of our criteria better than any other cipher, so we're choosing it. That one takes 128 bits, or 16 eight-bit bytes at a time, and converts those, sort of maps them to a different 128 bits, under the influence of a key. So the key uniquely determines the mapping between the 2^128 possible input bit combinations into a uniquely different 2^128 output.

The problem with CBC is that it is a block cipher, which is to say it only operates in these 16-byte blocks; whereas RC4 you could say, eh, my data is 55 bytes long. Give me 55 random bytes, which I will XOR with my 55 bytes of plaintext, and out comes 55 bytes of ciphertext. The problem with block ciphers is they work only in multiples of the encryption block size, which is in this case 16 bytes. So if you have any data you're wanting to transmit or encrypt which is not an even multiple of 16, you have to round it up with a process called "padding." You pad the balance out to an even block size multiple so that you can then run it through this block-at-a-time process.

Okay. So another problem with encryption is that we not only need to encrypt, but we need to authenticate, something we've often talked about, is we need to make this tamperproof. And Moxie Marlinspike is famously quoted saying: "If you have to perform

any cryptographic operation before verifying the MAC" - the MAC is the so-called Message Authentication Code. "If you have to perform any cryptographic operation before verifying the MAC on a message you've received, it will somehow inevitably lead to doom." Which is to say that, when you receive a message, absolutely the only safe thing to do is check to see if it's been tampered with. Don't do anything else. First verify that the message is authentic using the Message Authentication Code, the MAC.

Unfortunately, the guys who designed SSL got it backwards. SSL authenticates, then encrypts when it's sending. Which means that we have to reverse the process when we're decrypting, meaning that when we get the message, we have to, because it was authenticated, then encrypted, we have to decrypt, then authenticate. So that breaks this rule of never doing anything other than check for tampering. We decrypt, then we do the tamper checking in SSL protocol. And that's a critical downfall in the original design, and it's the way the BEAST attack happened that we discussed a few years back.

And this is essentially a variation on BEAST. Once we've been able to downgrade the communication from TLS to SSL v3, due to the fact that we decrypt, then we authenticate, it is possible for someone in the middle, in a man-in-the-middle position, very much the way that it happens with BEAST, to probe the communications, to probe the decryption. The padding at the end of the message is checked separately at decryption because padding is a process of encrypting. You have to pad, then you encrypt; which means you decrypt, and then you check the padding. So, and the padding is always by definition from - it's the last block of the cipher is the pad content.

And the way the CBC cipher works, when you're decrypting something that was encrypted with CBC, the second to the last block of ciphertext, that is, the text to be decrypted, is XORed with the last block of plaintext. So, and it's sort of an unfortunate characteristic of cipher block chaining. But what that means is that, if an attacker can mess with the second to the last block of ciphertext, they can, that is, by flipping bits in the second to the last block of ciphertext, they can induce bit flips in the last block of plaintext.

What that does is it gives an attacker control over the tail end of the message and allows them to essentially probe the plaintext because what happens is, after the decryption occurs, the padding is checked. And if the padding is wrong, it'll generate a padding error before it generates a message authentication failure. And that allows the attacker to essentially probe the plaintext by looking at the errors being returned by the server.

Now, this takes a long time. This, for example, takes 256, on average - wait, is 128 or 256? In what I was reading, the analysis kept saying 256. It seemed to me they ought to be able to get lucky, and it ought to be an average of 128. So maybe it was a maximum of 256 and an average of 128, which is half the number of possibilities of eight bits. So they probe one byte at a time in order to obtain the information about the plaintext, then go to the next byte. So first of all, many of the reports said that it would require several hundred probes. In fact, several hundred probes gets you one byte of information. So it's actually several thousand probes in order to get you a chunk of bytes, like a cookie, for example.

Now, okay. So what are we trying to get here? Let's step back from this for a second. The bad guy, the attacker in the middle, would love to, for example, get the session cookie. Remember that once the user logs in, provides their credentials when they log in, they maintain a persistent session with the remote server because every time their browser, so long as they're logged in, makes additional queries to the remote server, it provides the cookie, saying - reminding the server for every single query, this is who I am, this is who I am, this is me coming back, and here I am again, and here I am. So

that's the session cookie. And the lengths vary. Typically they're huge, like long mothers.

So it's going to take many sets of multiple hundreds of probes stepping through in order to extract the cookie. Now, where do these probes come from? Well, the probes have to originate from the browser. That is, the man in the middle, this is all opaque to the attacker who has poised himself in the middle. They're seeing gibberish go by. And so because they have an intimate knowledge, because they've forced a downgrade to SSL v3, they know what protocol has been negotiated, but they don't have access to the encryption keys. That was securely negotiated by SSL3.

So they're being forced to futz with the data going from the client to the server, making changes, introducing errors. Which means part of this attack requires malicious code in the browser to initiate these thousands of queries. So one of the reasons the bar is so high on this is that, not only do you have to have an attacker who's positioned themselves in the middle, but the page which is generating these queries has to be running malicious code in the context of the site that you're attacking. Because, remember, we also have the whole same domain protection that prevents script from obtaining information about any other domain. It's only able to receive information to look at the domain information for the domain from which the script, its same origin, from which the script originated.

So somehow the attacker not only has to get themselves in the middle, but they have to inject malicious code into the user's browser. And that causes the browser to be initiating all of these hundreds of queries. And the attacker has to have some sort of a dialogue, establish a dialogue so that it's able to tell the clients, the malware running in the client, okay, got that byte. Now what has to happen is the client needs to pad the query by putting some additional stuff in the header to force the cookie to change its byte alignment because the attack requires - the attack operates on the boundary of successive blocks of decryption. So, first of all, this is very complicated. And it requires active participation by malicious code in the browser.

Okay. So let's assume that somehow all of this happens, that an attacker is able to arrange to make that happen. They have to get themselves in line. They have to be able to modify traffic. They have to be able to inject somehow into a session that they've - it's never clear how they can inject anything into the browser. I mean, this is a secure page. And all they're able to do with thousands of queries which they've induced the browser to generate, and they're breaking the queries as they zoom by in order to cause the server to generate - in order to send back padding errors that allow the attacker to eventually determine a single byte, then tell the browser, okay, got that one, shift everything down, now I'm going to work on getting the next one. So it's never been made clear how that malicious script gets running in the browser, but that's another requirement for this.

So the bar for pulling this off requires all of that to be in place. So I'm going to, after Leo tells us about our final sponsor of the podcast, explain what the official fix is, why GRC doesn't care, and why this entire thing is completely bogus, and not just because it's hard to do, because it's ridiculous.

Okay. So, first of all, the recommendations in the industry which flew around last week with great gasps and breathless instructions and websites popping up to warn people if somebody still supported SSL 3.0, was either to, well, disable CBC. Except, what, then you're going to fall back to RC4? That doesn't work. So it's disable SSL3 completely. That was the only remediation anyone had. I hope people are a little skeptical now that anyone could actually pull this off. It's not even clear to me how it could actually be done. But in a minute I'll tell you why no one ever will.

The official fix is very clever. And that is that, remember how I said, during the negotiation of the protocol, the client provides a list of the ciphers that it supports - that's called the "cipher suite," the suite of ciphers that it knows about - sends it up to the server. Server looks over them and picks hopefully a good one that they share, and then that's what they use.

Well, there have been other things in the past that have sort of "overloaded," as the term is in programming, overloaded that with additional meaning. And there is a pseudo cipher suite called TLS Fallback SCSV, stands for TLS Fallback Signaling Cipher Suite Value. And it's included in the list of ciphers which the client knows about. But it doesn't represent a cipher. It represents an assertion that it knows about TLS because only if it knows about TLS would it know to include it in the list.

And so this is very clever because, if an attacker tried to perform the downgrade attack by faulting that initial handshake, the client would, believing that it had no choice, drop back to SSL3. But it would still include the TLS Fallback SCSV value. The server that's also aware of TLS would have seen an, oh, would not have received that initial handshake because the man in the middle grabbed it, blocked it, and returned an error. So what the server sees is an SSL3 request that includes in the cipher suite negotiation the TLS Fallback SCSV value. Which is to say, specifically to detect this. And the server says no. That tells the server that the client is asking for an SSL3 connection while knowing about TLS. A client should ask for a TLS connection if it's including the TLS Fallback among its cipher suite values.

So it's a beautiful prevention for this kind of protocol downgrade. Unfortunately, it's new. And that means the right solution for this problem is for the endpoints to upgrade to support this. OpenSSL immediately added support. If you update to whatever your platform is, if you're a Linux or a UNIX, and you update your system to the latest OpenSSL, it now knows about this. Now, it's got to be supported at each end. So we need the servers to update, too. I imagine next month Microsoft will have a patch for all of their supported servers and platforms, because you can also receive connections on a non-server, to add TLS Fallback SCSV support. That's the right answer. That completely solves the problem.

Now, GRC is not vulnerable because I don't use cookies. In fact, I don't use any state in my ciphers. I've mentioned this before, but even my eCommerce system operates in a cookie-free fashion. After the user has provided some information, that's encrypted in a blob for which only GRC has the key and provided back to the user with their next page. And then when they submit that, the blob is returned. So at no point does GRC ever have any state information. No one has to log into GRC. There is no notion of logging into GRC. We do have cookies, but that's only for background R&D, to sort of look at statistics of how cookies are being handled by different browsers. We use it for nothing.

So for anyone who is worried that GRC is still supporting SSL3, yes, I am, and I intend to continue doing so. I will certainly update the server next month or whenever Microsoft produces the fallback support. But it's really not necessary because, at least for GRC, there's no danger in falling back to 3.0 because, in the first place, I don't think anybody is ever going to perpetrate this attack, just for reasons of it being so difficult.

But here's the final point. Not only is it incredibly difficult, it's completely bogus because the absolute requirement to pull this off is that the attacker somehow get malicious code in the client running in the context of the site that is under attack so that queries are being issued to that site, and that it then do these thousands of queries in order to provide the opportunity for the attacker to break the end of the queries in order to generate the padding failures. Okay? And that's ridiculous.

If anybody could get script running in the user's browser, even without any fault in the protocol, in SSL, even under TLS 1.2, the latest one, a client could easily use side channel leakage in order to communicate with somebody passively, not even an active attacker, a passive listener on the connection. The client knows what its cookies are, for example. That's something you can easily read in JavaScript is your HTTPS server-side authentication or your various cookies.

So the client could simply issue, essentially send out in binary, for example, binary encode the cookie in a sequence of short and long queries: short query, short query, long, long, short, long, long, short, short, short, long, short, long. Somebody monitoring just looks at the length of the outgoing query, which is the client, the malicious client they stuck in the user's browser, essentially like using Morse code to communicate the sensitive data out over the wire. And so a passive attacker can use what is essentially a side channel attack in order to obtain that information. And that works on any protocol. It doesn't require any vulnerability. And it's vastly simpler than this ridiculous thing that's going to get closed up here in a month or two.

So anyway, when I looked closely at what it took to actually pull this off, it looks like what they did was, I mean, they truly did find a problem. And, yes, there's a problem with the protocol. That should get fixed. We always want to fix our protocols. So any weaknesses should get fixed. I'm sure this will be. It's already fixed in OpenSSL. The other server platforms I'm sure will be pushing out support for TLS Fallback shortly. So this problem will go away.

But to me it looks like they took a theoretical vulnerability and then reverse-engineered an attack which is so difficult to pull off, if you could arrange, if you could set the situation up that makes that attack possible, then you're already able to do something far more easily and far more damaging against which there's no protection whatsoever, a side channel attack using query length in order for the browser to communicate out to a passive listener. So there you go. And Leo is now smoothly shaven.

Leo: Do I - yes, I am. And go ahead, kiss that there. Do you - if I run cookies on a browser, is it worth worrying? I mean, you're invulnerable because you don't use cookies. It's just too hard to do this, even if you do use cookies.

Steve: Yes. It's a theoretical attack. Nobody has ever been attacked by it. I don't think anybody ever will. I mean, I just - it's not at all clear how it could ever actually be set up. We know some pieces, but the requirement to run malicious script in the browser, that's the deal breaker because, if you could run malicious script in the browser...

Leo: You could do a lot of other stuff, too.

Steve: All bets, yes, all bets are off. You don't need a downgrade attack. You could do this over TLS 1.2. It's ridiculous.

Leo: Yeah. That's the key here.

Steve: So it's like, okay, thank you very much. Everybody, you know...

Leo: In the words of Frank Zappa, I think we have - oh, I don't have any audio. Darn it. I was going to play a little bit of a Frank Zappa song in which he says "the poodle bites."

Steve: Ah.

Leo: Steve Gibson is at GRC.com. That's where he gets his mojo and his yabba-dabba-dos. So be sure you go on over there.

Steve: And thank you to our listeners, yes.

Leo: Yes, buy some SpinRite and make his day. And Fred Flintstone's day. You can also, while you're there, find so many other great things, all free. The only one he charges for is SpinRite. You've got the Perfect Paper Passwords. You've got Password Haystacks. You've got information about SQRL and test implementations and a whole forum on that. And you have a place where you can ask questions. And there's always questions for Steve. You could tweet him because he's @SGgrc on the Twitter. But you can also go to GRC.com/feedback and leave your questions there. Do you think - I know this was scheduled to be a Q&A.

Steve: Well, let's try for one next week.

Leo: With any luck we'll have...

Steve: As long as the sky doesn't fall, yes, I will suck up our mailbag from this week and last week, and we'll have a great Q&A next week.

Leo: Although I'm having to think that maybe this YubiKey may end up, FIDO may end up being part of the show, as well. But, you know, we have room for that.

Steve: Yeah. And if people ask me a question, then that's a perfect segue.

Leo: There you go. Hint, hint.

Steve: Yeah.

Leo: Of course you can also go there to get 16Kb audio versions, the smallest version offered, as well as nicely written transcripts.

Steve: Handwritten.

Leo: Handwritten by a human being. Steve pays Elaine to do those.

Steve: No bots over here. We do not have Siri at this end, unh-unh.

Leo: We also have full-quality audio and video at our site, TWiT.tv/sn. And wherever finer podcasts are aggregated, you're sure to find it because it is one of the oldest shows on the 'Net nowadays.

Steve: We're surviving everybody else.

Leo: Yes, we are.

Steve: We're like the cockroach of the Internet.

Leo: Steve, what fun. Thank you so much for helping us don our propeller caps. We'll see you next week.

Steve: Thanks, Leo.