Certificate Transparency

Description: This week we discuss yesterday's further good privacy news from Apple, the continuation of VPNFilter, an extremely clever web browser cross-site information leakage side-channel attack, and Microsoft Research's fork of OpenVPN for security in a post-quantum world. Microsoft drops the ball on a zero-day remote code execution vulnerability in JScript, Valve finally patches a longstanding and very potent RCE vulnerability, Redis caching servers continue to be in serious trouble, a previously patched IE zero-day continues to find victims, and Google's latest Chrome browser has removed support for HTTP public key pinning (HPKP). And, finally, what is "Certificate Transparency," and why do we need it?

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-666.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-666-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. We've got a lot to talk about including the changes Apple's making to Safari and why they're really good for your privacy, and a little insider action on the certificate thing. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 666, recorded Tuesday, June 5th, 2018: Certificate Transparency.

It's time for Security Now!, the show where we protect you online, thanks to the good auspices of this fellow right here, Mr. Steven "Tiberius" Gibson of GRC.com.

Steve Gibson: Yo, Leo.

Leo: Yo, yo, yo, what's happening, dude?

Steve: Well, we have made it to Episode 666.

Leo: Oh, no.

Steve: So as I tease our listeners, we are two thirds of the way through the lifetime of

this podcast because all of my systems only support three digits.

Leo: We'll talk about this in 333 podcasts.

Steve: That's right. But until then, an event that happened a couple days ago, Chrome 67, caused me to think, you know, we should do an introduction to something that we have never yet introduced, and that is certificate transparency. This is an initiative that Google rather quietly initiated, like, four years ago. My favorite certificate authority, of course we all know, DigiCert, was the first CA onboard. And through the podcasts, through the 666 previous weeks that our listeners have endured, we have often talked about, because it's like a big issue for security, the problems with the CA system, with the Certificate Authority system, certificates being misissued, them being maliciously issued and so forth.

So certificate transparency is a publicly posted and publicly auditable log of certificates. And it very elegantly - oh, and the other thing, of course, that I annoyed Google with was the revocation problem, the fact that, as we know, certificate revocation really never has worked. So this is an interesting and way more present solution than anyone has really been aware of. And I've touched on it from time to time.

The reason it was brought to mind for me this week is that Chrome 67 just dropped and removed a previous sort of attempt at solving this problem with something called HPKP - HTTP Public Key Pinning. Well, that just got removed from Chrome, and that doesn't happen very often. So we'll talk about that. Lots of other stuff, too. We've got some nice privacy-advancing news that, as you know, because you were present during yesterday's WWDC presentation where Craig Federini - Feder what?

Leo: Federighi.

Steve: Federighi. I want to make sure I don't say Ferengi.

Leo: No, he's got better ears than that, yeah.

Steve: That's right.

Leo: Well, his hair, I don't know.

Steve: Well, we want to talk a little bit about Apple's movement in the privacy direction. Also a little bit of follow-up on last week's VPNFilter takedown by the FBI and them asking everybody to reboot their routers and what's been going on since.

The Picture of the Week involves an extremely clever web browser cross-site information leakage side-channel attack. In other words, we've talked often about the same-origin policy and how rigorous browsers are to separate what can be run, what code or images can be displayed from what site and so forth. This is a way of breaking the iframe sandbox that is just so cool that of course we have to talk about it. And the browsers have scrambled in order to fix this.

We're going to talk a little bit about quantum crypto today which is - it's also been one of those things kind of lurking in the wings. Microsoft Research, the research branch of Microsoft, has forked the OpenVPN project and developed an OpenVPN version which is quantum-safe, so they believe. Also, at the same time, the front office of Microsoft dropped the ball on a significantly serious zero-day remote execution vulnerability. Hopefully next Tuesday they're going to - because they're scrambling right now. They were given 120 days. It's like, whoops, sorry, the clock expired. So we'll talk about that.

We've got Valve finally patching a longstanding and, geez, it had been horrifyingly potent remote code execution vulnerability that has existed in all 50 million instances of the Steam client for, like, 10 years. We're going to revisit the Redis caching servers, which continue to be in serious trouble. We have a little more information about that. A previously patched zero-day in IE continuing to find victims. And then, as I mentioned, we'll talk about Chrome 67, their removal of HPKP, why, what that means; do a little bit of housekeeping miscellany; and then talk about certificate transparency, which is I think without question where we're heading in the future. So yes, another information-packed week.

Leo: Yes, yes, yes.

Steve: So our Picture of the Week relates to a story that we'll be getting to a few stories from now, but it's just so pretty that I wanted to put it up there. For those who are listening, it just shows a stack of layers, and the bottom one says "cross-origin iframe." And then there's a target pixel sort of focused on that iframe; and then a bunch of layers on top of it labeled "difference," "lighten," "difference," "color-dodge," "multiply"; and then some transparency layers labeled "saturation."

And the point is that this is a multilayer, many layer stack which web browsers, contemporary web browsers that support an advanced feature of CSS3, which both Mozilla and Chrome do because they share a common graphics library in their foundations which does this. This turns out to have been leveraged into what is essentially a side-channel leak in browsers that, for example, allows an unwitting person who visits a web page that does this to them to have, for example, their Facebook profile photo and their name divulged, even though it's explicitly against all of the browser rules for that to happen. So the way this works is very cool, and we'll be talking about it in a minute. But I wanted to put the picture up at the top because it's a pretty picture.

So since it just happened, I just thought I would mention that Apple is continuing, as they announced at the Worldwide Developers Conference yesterday morning, to move in the direction they've been going for some time, stating that one of the things that they're selling, one of the things their customers want, is increased privacy. And so one thing that they've done in Safari, they announced macOS Mojave as the next major MacBook OS, and iOS 12. I did note somewhere just digging around in the last day that apparently an iPhone 6s, which is now an old iPhone, would do something a lot faster under iOS 12. I don't remember what it was, but it was like, oh.

Leo: Everything. Oh, no, it's a big, big change.

Steve: Oh, so it's a big performance improvement?

Leo: Overall improvement, yeah, 70% improvement in opening the camera times, a double improvement in apps, I want to say app switching, 50%, big improvement.

Steve: Yay.

Leo: And people already reported with the beta that they're seeing this, yeah, yeah, yeah.

Steve: Oh, yay, yay, yay.

Leo: They really focused on performance.

Steve: Nice. So that's good. In Safari for both platforms, both macOS and iOS, we've talked a lot about fingerprinting, and that there's that Panopticlick site which makes it - I think the IETF sponsors it. And that's that site which has experimented with and shows a user when you go to Panopticlick how unique the fingerprint is that results from hashing a bunch of the browser headers. And we've talked about this through the years that, for example, the user-agent header will tell you not only what browser you're using, but what library of ActiveX, and their version numbers, you know, 3.29.724.0326. And somebody else will have .0327, and so their browser headers hash to a different fingerprint than yours.

And when you combine that, like all the version numbers of the plugins, in some cases the browsers will enumerate which fonts they have, which font libraries they have installed, so that a web server could know what fonts are available to be rendered in the browser. The browser headers have exploded over the years with all this extra information which with surprising accuracy can, without the user doing anything, allow sites that want to track them to lock onto them with this fingerprint.

So one of the things that we learned yesterday is that Safari is going to deliberately take action to thwart that kind of fingerprinting. So that's good. And it's funny, too, because in thinking about this, I realized how easy it is to confuse tracking with advertising. I think that the two get mixed up a lot together because, when you start talking about blocking tracking, there's this, oh, but wait a minute, how are sites going to survive? And it's like, wait a minute. Advertising and tracking are separate things. You can still have ads if you're not being tracked around the Internet.

So tracking blocking is different than adblocking. And it seems that what we're seeing is a pushback, well, obviously, against both. But there is an understanding, I think, and some websites are taking action on this as we've talked about, where if they see you running an adblocker, they can say - and I'm encountering that from time to time because I run uBlock Origin. And I'll often say, oh, yeah, okay, fine, and allow ads on a site that I want to support.

Now, that sort of follows into the other feature that I'm going to be interested to experience. And Leo, I don't know if you've talked to anybody who's using this next Safari. But what they showed us yesterday during the opening "what's new" at the WWDC, was a pop-up which you would get, asking for permission for another site to track you on the site you're visiting.

Leo: That's for like the Facebook Like button. So that means on the page that you're on there's JavaScript that then sends data to a third page. It's like third-party cookies, except it's not. It's using, I presume, JavaScript to do that; right?

Steve: Well, actually it could be using first-party cookies.

Leo: Right.

Steve: Because when you have an object like a Like button, then...

Leo: It looks like it's coming from inside the house.

Steve: Exactly. It actually is sourced by Facebook, so when you're logged into Facebook, as you move around the 'Net...

Leo: But the browser knows where - so the thing is the browser knows what page you're ostensibly on, and then sees a link back to Facebook. And that's when it says, uh-oh.

Steve: And so that's what I'm confused about. They're calling this "Intelligent Tracking Prevention 2.0."

Leo: Well, there's two things they're doing. So there's that. And then it looks like they're blocking supercookies.

Steve: Okay. So that's the browser fingerprint I was just talking about, right.

Leo: Yeah. So I think those are separate.

Steve: They are. And so what Craig said, for our listeners who didn't see this, he showed a Safari page with a popup notification reading, "Do you want to allow Facebook.com to use cookies and website data while browsing..." and then wherever it was you were. And their example was Blabbermouth.net. And so here's the problem, Leo. There's nowhere you go today that doesn't have, like, all this third-party crap loaded onto the page. So like I said, I'm going to be really interested to see what they're doing. I mean, are they going to - I can't imagine they're going to single out Facebook or Google.

Leo: No, although this was intended to single out Facebook and Google in the announcement.

Steve: Yeah.

Leo: I mean, that was very clear that it was a shot to the Facebook and Google. But, yeah, I mean, I guess they could treat any call to a third-party site like a third-party cookie; right?

Steve: But you'd be buried in permission dialogues.

Leo: Yeah.

Steve: It's freaky. And but on the second page of the show notes is my own chart from my own site that has always stood out in my mind. Years ago browsers were not handling cookies correctly. So I created this cookie forensics system.

Leo: I remember this, yeah.

Steve: But what stands out among all the browsers is Safari. Look at the Safari bar because Apple has always blocked third-party cookies by default. And they're the only browser to do so. And it's just - it's always impressed me that here among all the browsers - and obviously this chart is dated now. We've still got Firefox v2, v3, and v4. I haven't messed with this for a long time. But there is like this little tiny bar of Safari.

Now, this protection fails, as I was just mentioning, when you start hosting content, like active content, from other sources because then you've got JavaScript and other technologies present. But still Apple has, in fact, a long history of blocking this kind of action. So anyway, I just wanted to share with our listeners that Apple is continuing to move forward.

Leo: But you do raise an interesting point.

Steve: Oh, yeah.

Leo: It's not beyond Apple to kind of trumpet something like this, but do something less aggressive under the hood.

Steve: I just - I don't know how it could possibly work.

Leo: They can't do this, yeah, no, you're right.

Steve: It's just going to bury you.

Leo: Be like NoScript.

Steve: Yeah, nothing would work.

Leo: Nothing would work, yeah.

Steve: And maybe there's like a "don't ask me again" sort of thing. I mean, that's the only way I could imagine it is, yeah, you immediately start using the new version. So like it's not a per-site question, unless maybe you want it to be. But it's like, no, and don't ever ask me again sort of option. I just don't know. It'll be really interesting to see how this plays out because, boy, I mean, these days there's nowhere you go that doesn't have third-party stuff. So either they're explicitly looking for particular domains - of course then Google and Facebook would cry foul, saying, hey, you're singling us out for abuse compared to everybody else. And it's going to be really interesting to see how this happens. I don't know.

Leo: Yeah.

Steve: So as we know, last week's podcast was VPNFilter, talking about this collection of routers that had known vulnerabilities being exploited by, we believe, actually there's strong forensic evidence to suggest that it is the APT28 group, aka Fancy Bear, which is operating out of Russia, largely targeting the Ukraine. Since the takedown of this large botnet composed of all these routers - remember that the ToKnowAll.com domain was taken away from VeriSign's control over to the FBI, that were then using it in order to monitor the botnet and to get command and control away from those who had it.

What's happened is since that time the Ukrainian IP space has been actively scanned. And a number of security firms, JASK and GreyNoise Intelligence, have both been watching the 'Net in the wake of this takedown and have seen, let's see, one, two, three, four, five, six, seven, eight IPs scanning port 2000 of the Ukrainian IP space. Port 2000 is the one that the MikroTik routers unfortunately have opened that allows them to be taken over. And so this strengthens the sense that existed before that this was from Russia and aimed at Ukraine networking devices, and that this is still an active campaign, even post commandeering of the command-and-control server.

And what's odd is that one of the IPs, 9.110.0.5, well, nine-dot anything, the whole nine-dot network is owned by IBM and always has been. So I presume some obscure machine somewhere, I mean, it's not at IBM central headquarters, it's just some random machine somewhere on the IBM network at that IP, probably, I mean, unless it's a security scan, which is certainly possible, has probably got some malware in it and is scanning for Ukraine-based routers. Also there's two IPs closely related to each other in Russia, two in Brazil, one somewhere in the U.S. Oh, no, actually three in Russia, one in the U.S., and one in Switzerland.

So it looks like there are a few systems, without looking to see what they're all about, it's hard to know if they're malicious or not. But I think what we're going to see, and we'll be talking about another class of attacks here in half an hour or so, these Internet-exposed devices are going to be a constant source of trouble. And I don't know how they get fixed over time.

Okay. So Leo, this incredibly clever hack. Obscure as this is, it was independently discovered by security researchers who didn't know about each other - and the word I've used in the past for something like this is "sublime" - in CSS3, the latest standard of

Cascading Style Sheets for our browsers. As we know, HTML5 and CSS3 have just piled features onto browsers. I mean, they are rapidly becoming full capable citizens on the desktop and in fact are beginning to host full-blown applications as they become faster and the script that they're able to run is having every feature that you would want from a desktop app.

Now, unfortunately, some of the press didn't get this right. Ars Technica ran with the headline "EXPOSED [in all caps] - Chrome and Firefox leaks let sites steal visitors' Facebook names, profile pics," et cetera. Okay, well, yes. But it's really not the fault of Chrome and Firefox. As we've talked about, one of the core security requirements for browsers is the same-origin policy, the enforcement of the same-origin policy, the idea that if we're going to have web pages with content from multiple sources, as I was mentioning just a minute ago, virtually all - except GRC's web pages - virtually all web pages have stuff coming from everywhere else, all kinds of other places.

So sandboxing those is crucial because, for example, any web page that has ads is now using an ad network to fill in the rectangular space on the web page where the ad network will put up an ad. So that is inherently a third-party source for security since the server hosting the web page is giving the user's browser an ad network URL saying fill this frame in with whatever you want. It is crucial that that be contained, that that frame be sandboxed and has no access to other frames on the page or the page itself. Otherwise, we'd have no security.

Well, for example, that page has a persistent cookie which is how the user is logged in. So you absolutely don't want third-party content hosted on that page to be able to access, for example, the cookie of the hosting page or you'd have a problem like we did once upon a time with Firesheep, before HTTPS was in place virtually everywhere that it mattered, where session cookies were not being secured. So containment is crucial. Therefore, the idea that some researchers broke containment is significant, and that's what happened. And they did so in just the coolest way.

What they realized was that the CSS3 feature called "mixed-blend mode" is not constant time. So we've talked, for example, about mistakes in ciphers in the past. For example, AES did not make the mistake because we knew about the mistakes by the time AES was being designed. But in explaining this in the past I've explained that you don't want secrets to cause, for example, the bits of a key of a cipher to cause the decryption to take different amounts of time, depending upon those bits.

So what you want is you want a cipher that is constant time relative to the bits of its key because, if you don't have that, and you're able to time how long decryption takes, you can start inferring what the key's bits are from that. And so that's a classic example of side-channel leakage. Another instance is constant power. You don't want the power consumed to be a function of any secrets.

So what several independent researchers simultaneously realized - and there had been some related work involving SVG, the Scalable Vector Graphics system. So there had been sniffing around this before, and some other work in this area. They realized that this mixed-blend mode might not be constant time. And in fact it had not been. So the idea with mixed-blend mode is it's very much like applying filters to graphics. And in fact, Leo, there's a CSS-tricks.com site - I have a link here in the show notes - on the basic CSS blend modes that shows you examples, where for example you have a black-and-white photo of a building, and then a red rectangle, and you merge those two together, and it just sort of gives you a red tint on the black-and-white photo.

So the idea is that in our browser we are now doing compositing, image compositing of

addition, subtraction, multiplication, saturation, whitening, lightening, darkening, and so forth, all kinds of different features. And that requires math on the pixels. Individual pixels that line up through these layers have math operations performed on them. And if it's just addition, subtraction, and multiplication, then those are probably constant time. But if you have something like saturation, where you might add two pixels, and you need to check to see if the sum overflows, then you need to clamp the pixel at maximum luminosity rather than allowing it to overflow. So suddenly you have a code path that depends upon the brightness of the pixels.

And so what these guys did, and they've done a proof of concept, they arranged to create a code where an unwitting web browsing person would go to a website and in 20 seconds could have their Facebook name obtained by that site that they're visiting. And that's done by the site having an iframe to Facebook.com. That will, since the user has a presence at Facebook, if you go to Facebook.com it brings up the user's page in the iframe. And they hide this under another image, and then they're able to scan the iframe graphics with their own scanning pixel, measuring the length of time required to perform this compositing math as this pixel scans, and thus infer from the length of time it takes the color of the pixel in the iframe of a foreign site. And so it's just an amazing hack.

It turns out that this was disclosed responsibly some time ago, and that there is a graphics library that Facebook and Google share. It's called S-K-I-A. And it's an open source graphics library. Google purchased the company, Skia, Inc., some years ago and so brought them into the Google fold and then made the whole thing open, open sourced it under the new OpenBSD license, I think, yeah, the BSD free software license. When they fixed the problem, in order to basically focus on this layer compositing problem, they also in the process not only made it constant in time, thereby eliminating the ability to acquire any information based on how long things take, which you have to do if you're going to have security, but across the board it is way faster than it was before.

So basically they focused on it, made a constant time and constant power graphics processing pipeline, and at the same time, I've got the numbers in the show notes, it looks like in some cases better than a factor of three faster. So like in less than a third the time it took before they're able to do that. But anyway, just another example of how, despite our best efforts to keep things secure, when we add all kinds of new features to existing technology, because there's an exponential effect of every feature interacting with every other feature, when you add a couple new features, suddenly the possible number of interactions among them just explodes.

And so we have a whole bunch of new HTML5 features, a whole bunch of new CSS3 features, and here's an example of where you would never have thought that what you're seeing in an iframe could somehow be leaked into code running on the page that hosts the iframe. Yet that has just been pulled off. So I salute the cleverness of these attackers, or, well, these security researchers who thwarted what, I mean, it's not like it's a huge problem. But you can imagine some situations where you're logged onto a site and there is some sensitive information that the site displays when that's the condition you're in, and this would have allowed that displayed information to leak onto a page that you're visiting.

So, I mean, it does break the security guarantee which is so important for web browsing as we go forward. It's not a fast attack. It takes time to essentially measure the rendering time of individual pixels. But they demonstrated the recovery of the person's name in Facebook given about 20 seconds of just sitting there, staring at a page, doing other things, and this thing's happening in the background.

Leo: Would you have to be able to watch their browser? I mean, how would you get the information?

Steve: So you as a user, an innocent user, would just go to some site. And you would have no idea that this was going on in the background.

Leo: The site would have to be malicious.

Steve: Correct.

Leo: And it could measure it because of course it's loading it. Okay.

Steve: Correct, yup. Amazing. So OpenVPN is to this day my favorite virtual private network platform. It's what I use for connecting to GRC. It's what I use between sites. I mean, the OpenVPN guys just nailed this problem. It supports UDP and TCP protocols. It's always hosting the latest OpenSSL library, so it's up to date on all of its crypto. It's the one. And so if you were a user wanting a VPN, what you would look for would be an OpenVPN host, but OpenVPN is what you want to be using. You don't want Tom and Jerry's VPN. This problem has been solved with OpenVPN.

Except what about quantum crypto? As we know, cryptography is potentially endangered by the specter of quantum computing because it has been known now for some time that the hard problems upon which some classes of our cryptography depend may fall apart. They may no longer be hard problems, if we finally have quantum computers that are more than just at the toy scale. Right now quantum computing is still a toy, but that's the way all this stuff starts. Computers were once toys. Now they're certainly not.

So way back in 1994, a mathematician named Peter Shor, whose algorithm bears his name, Shor's algorithm, showed that, if a theoretical quantum computer exists with the set of properties we expect them to have, that integer factorization was no longer hard. So as we know, for example, classic RSA public key crypto, where we take two very large primes, and we multiply them, and that creates the public key, and that the reason we're able to, like, give it away and make it public is that, even though it's there, out for everyone to see, it is composed of the multiplication of two primes, and nobody has figured out how to unmultiply them, how to factor their product back into the individual.

So one of those primes is the private key, and it's also sitting there out in public view, except since it's been multiplied by another prime, you don't know where it is. You can't find it in the product of the two. Yet the cleverness of crypto allows you to use the properties of the private key without knowing what it is. So that's RSA-style public key crypto. It utterly, all the public keys floating around out there contain the private key. But it's only the fact that we don't know how to crack them back apart that makes it safe.

So Peter Shor showed us that, oh, give me a quantum computer, and I'll crack all those public keys back into their two components, and it's game over for RSA-style public key crypto. So that's a little worrisome for cryptographers. Elliptic curve crypto in its current form is similarly vulnerable. So it's not like ECC crypto isn't in similar trouble. Shor's algorithm can also slice right through the so-called "discrete log" problem, which is what

protects elliptic curve's point multiplication, which is the hard problem that elliptic curve crypto offers.

Now, what's interesting is that it's that public key crypto, whether it's RSA or elliptic curve, where the vulnerability exists. Our symmetric cipher, like AES, it is believed to be fine. Everyone wants to double the key from like 128 bits to 256. But AES with a 256-bit key is currently believed to be plenty strong, even in the face of quantum computing. And the same is true for our hash functions. SHA-2 with a 256, so SHA-256, or the SHA-3 functions, which have now been settled on, and they're just sort of like on the back burner, waiting for us to need them, they don't have to change post-quantum. Yet the public key components are based on difficulty. So that's why we talk about an RSA key length of 1024 bits, or 2048, or 4096.

The problem is that's a lot of bits because the thing that they're trying to make hard, which is in the case of RSA prime factorization, is not that difficult. So the Microsoft Research Group have forked OpenVPN and have a running, operating, you can download it and install it today if you want, post-quantum OpenVPN implementation. They have three different algorithms, one called "Frodo," which is a key exchange protocol based on what they call the "learning with errors" problem. Then there's one called "SIKE," S-I-K-E, which is a key exchange protocol based on what's known as Supersingular Isogeny Diffie-Hellman. And I imagine, probably before we get to Episode 999, I will be talking about what that is.

Leo: Ooh, that'd be interesting.

Steve: Yes. And then "Picnic," which is a signature algorithm using symmetric-key primitives and non-interactive zero-knowledge proofs. And again, I imagine we will be talking about that in the future. I won't get into it in any more detail except to note that those things exist, and that we don't need them yet. The reason cryptographers are thinking about post-quantum crypto is not that we're in any obvious danger, but that we now understand these things take time to develop and prove, and we would like to have them ready, much like we have SHA-3. That's a whole 'nother next generation of secure hashes, just sort of sitting there on the backburner. We'd like to have well-researched and tested algorithms which are resistant to the things that we think quantum computers will get good at doing when quantum computers get good at doing anything. And right now they're just toys. But everybody's excited about them, and so we know that, famously, these things are always going to be getting better. Quantum computing is going to be no exception to that, I think.

So anyway, it's called PQ Crypto, Post-Quantum Crypto. I have a link in the show notes for anyone who's interested. It's up on GitHub. And, oh, by the way, I'm sure, I don't know if you've talked about it yet, Leo, but Microsoft bought GitHub for $7.5 billion.

Leo: Yes, they did.

Steve: So they are GitHub's new parent.

Leo: This is upsetting to a lot of open source folks.

Steve: And GitLab has seen a huge spike of incoming projects, moving from GitHub to GitLab. There are other people who have no problem with Microsoft being a benefactor for GitHub. And in fact the new, or I guess the incoming Microsoft CEO of GitHub has said - well, of course he's Microsoft, so he would - that Microsoft will not be changing anything.

Leo: Oh, yeah, yeah, yeah.

Steve: Nothing happens.

Leo: I've heard that before.

Steve: I know, I know, yeah.

Leo: GitLab is not much different, frankly, than GitHub. It's going to have the same issues and could easily be acquired by somebody. The problem is, I mean, I use a private Git. But if you want to have, like this project you were mentioning, if you want to have your code be public for people to download, look at, and contribute to, you need a public-facing server to do that. And I think GitLab might in fact just - SourceForge has also said, hey, we're not as bad as we used to be. Come on over to SourceForge.

Steve: Oh, god, it's a sewer, Leo. Have you seen, I mean, like, it's...

Leo: Well, I think they're maybe not as bad. I don't know. It was sewer, you're right.

Steve: Whenever I'm looking for something, and I have to go get it from SourceForge, it's like, oh. Really?

Leo: SourceForge used to be the place, and actually GitHub kind of replaced SourceForge. And I'll be curious. I mean, Microsoft loves open source.

Steve: They are loving it more every day.

Leo: More than they did.

Steve: Yes, yeah.

Leo: I don't know. You can run your own GitLab instance. So that may be what people what people end up doing is just hosting it themselves.

Steve: Yeah. And just say...

Leo: And GitLab's looking at federating, which would really be cool. Then you could have multiple GitLab sources - including I could run my own GitLab server, which I do - and then federate it so that you could go to GitLab.com and see all of them. That would...

Steve: And that solves the problem of fragmentation, where...

Leo: Exactly.

Steve: Yes. And that's what's been so nice about GitHub is it was one-stop shopping for all these different projects.

Leo: Right. There were issues with GitHub, and frankly they were losing money, and that was what really happened. They were struggling.

Steve: So when four months is not enough time to fix a bad bug. Back in, oh, a little after the middle of January, a researcher, Dmitri Kaslov of Telspace Systems, discovered a significant problem in Microsoft's Windows JavaScript engine. He responsibly disclosed it to Trend Micro's Zero-Day Initiative to start the clock. The ZDI group, the Zero-Day Initiative group responsibly notified Microsoft on January 23rd and received a same-day acknowledgement from Microsoft.

Then a week ago, last Tuesday, May 29th, more than four months later, the ZDI Group decided to hold Microsoft accountable for not patching this still-unpatched serious remote code execution bug which exists in their JavaScript engine. They posted last Tuesday. Their post was titled "(0Day) Microsoft Windows JScript Error Object Use-After-Free Remote Code Execution Vulnerability." And we've talked about use-after-free problems, the idea being that in languages which incorporate automatic garbage collection, so that the programmer is not required to indicate when they're no longer using an object, the system releases it.

But in some cases, when mistakes are made, you can still have a pointer to the object whose memory has been released. And using that pointer, there are various ways to exploit that in order to get your own code to run, which is what happened here. Their posting says: "This vulnerability allows remote attackers to execute arbitrary code on vulnerable installations of Microsoft Windows." And at the moment, that's all of us. "User interaction is required," they write, "to exploit this vulnerability."

And before you go "Whew," it turns out that the target must visit a malicious page or open a malicious file. I mean, so it's not like somebody can reach out from across the world into any Windows system. But all it takes is going to a website which has leveraged this. Which means, as soon as the information about this gets loose, it will be actively exploited because that's what happens with these things. So in other words, just allowing Microsoft's JavaScript engine to interpret JavaScript code received from a visit to any website.

Leo: So that's on Edge or Internet Explorer. That's, like, Microsoft's browsers. Wow.

Steve: Yes, yes. "The flaw exists within the handling of error objects in JavaScript," they write. "By performing actions in script, an attacker can cause a pointer to be reused after it has been freed. An attacker can leverage this vulnerability to execute code under the context of the current process."

So the timeline of this is interesting. They posted it on their page. On January 23rd they say ZDI sent the vulnerability report to the vendor, in this case Microsoft. On the same day, 01/23/2018, the vendor acknowledged and provided a case number. 04/23, so 90 days later, like to the day, exactly 90 days later Microsoft apparently suddenly woke up and said, so ZDI says here: "The vendor replied that they were having difficulty reproducing the issue report without a proof of concept."

The next day, on April 24th, ZDI confirmed the proof of concept was sent with the original, and sent it again. A week goes by. Now we're at May 1st. The vendor acknowledged receipt of the proof of concept. Seven days later, a week later, on the 8th, Microsoft requested an extension. On the 18th, ZDI replied: "We have verified that we sent the proof of concept with the original. The report will zero-day on May 29," which was last Tuesday.

The good news is there's no additional - there has not been a full disclosure. They have not published anything. So it's not like they're jumping up and down and saying, sorry, here's all the details. This is a bad bug, apparently. And so, I mean, nobody wants to be malicious about this. Hopefully, since today, which is June 5th, is the first Tuesday of June, second Tuesday the 12th is next Tuesday, and let's hope that this thing gets fixed.

So for mitigation, because these guys do know exactly all the details, having had a proof of concept, basically demonstrating remote execution of code on a JavaScript recipient's computer, for mitigation they said: "Given the nature of the vulnerability, the only salient mitigation strategy is to restrict interaction with the application to trusted files." Essentially, and somewhere else I saw them saying, you know, don't use IE, Edge, or anything that could run JavaScript. Unfortunately, that's even Microsoft Office. So anyway, we will hopefully next week have this thing fixed.

The problem is that, in our modern ecosystem we know that patching or that the manufacturer patching a problem doesn't fix it universally. There is another instance of a zero-day VBScript, so not JScript, which is the last one, but a Visual Basic, a VBScript problem in IE and Office that was fixed previously, and has now been added to the popular RIG exploit kit, which is exploiting what was a zero-day, even though it's been fixed.

So it's been added to this active exploit kit. A target receives a Microsoft Word document, opens the document, which launches the second stage of the exploit, which is enabled by an HTML page in the document which has VBScript, which also has a use-after-free vulnerability and executes shell code of the attacker's choice. So even though that has been patched, it was a couple months ago, they're still actively exploiting these. And unfortunately I imagine that this JScript exploit, even if it gets patched, as we hope it will next week, it'll get added to the toolkit. And people who aren't keeping themselves current risk being vulnerable.

So for us, listeners of the podcast, the takeaway is do not wait long to update your systems. I know that, with systems taking in some cases a long time to update - as we

know, Windows 10 just got better about that. I heard you and Paul talking about this a couple weeks ago, Leo, the fact that - and we did on this podcast - the fact that Microsoft had moved a lot of the housekeeping needed for these major monthly rollups into the background so the system was much more ready to go when it finally said, okay, now you need to do a reboot so we can do the final work and bring the new version of the OS online, making it a lot easier. And it is possible now, as we know, to schedule these updates for a time when the person's not using the machine.

Leo: Sort of.

Steve: Yes. Ah, yes. I was going to say many people unfortunately turn their machines off when they're not using them.

Leo: Well, and also there's a problem of detecting when you're using the machine. Some people just sit there and look at it for a while, and I've seen machines restart just on their own while you're sitting there. It's very frustrating.

Steve: Yeah. In fact, I think I told a story when I was attending that DigiCert conference in Utah, like in November of last year, one of the people there opened their laptop, and it just sat there for like an hour. It was unbelievable how long it took. And he was completely offline, couldn't do anything.

Leo: It's stuck. I'm updating my Mac right now. I'm stuck. I can't do anything.

Steve: Yup, the little spinning rollercoaster dots.

Leo: Yup, no fun.

Steve: Okay. So the good news is Valve immediately patched a worrisome security bug in their Steam client, which had apparently been present for the last decade, always there.

Leo: Wow.

Steve: Yeah. And bad. A single packet remote code execution vulnerability. Valve designed their own UDP packet-based, connection-oriented protocol, which is fine. But it meant that their own code was receiving UDP packets. And that's also fine.

One of the things that can happen to packets on the Internet is they can be chopped up into smaller pieces. Years ago on this podcast, when we were talking about how the Internet works, I extensively covered this incredible, at the time, conceptual breakthrough with packets and routing where you would have this loose federation of routers that sort of know what direction packets should be sent to, and that's kind of it. So that packets come in, and they say, oh. They look at the destination IP, and they say, oh, that came in on interface four, and interface two is the direction that packet wants to

go in.

And of course we were talking just last week, that BGP failure is an example of what happens when this really lovely system fails us. When routers advertise that they have shorter routes than they really do, the routers that they connect to go, oh, you're a better place to send this packet, even if it's not true. So anyway, the point is that not all router-to-router links are identical. And they're more identical now than they used to be.

But, for example, there are some links where you can use larger packets. And a large packet could go a few hops and then hit a link where the recipient doesn't support large packets. That forces the sender to fragment the packet into multiple smaller ones in order for them to cross this inter-router link that can't do big packets. And there are things called "jumbo packets." So at the receiving end, once a packet has been fragmented, it is not reassembled. The designers thought, well, if we reassemble it, and then it goes another couple hops, then it has to be fragmented again, then routers are just doing a lot of extra work. So once a packet's broken up into smaller pieces, they stay small until they get to their destination.

The mistake that the authors of the Steam UDP protocol made was surprising, but maybe understandable. The first packet, the first UDP packet, because fragmentation is possible, it contains both a fragment size and a total datagram length. Normally they're the same. But if the packet has been fragmented, its own size will be smaller than the datagram length. Well, what's missing from the code to check the first packet in a set of fragments is whether that's true. And that missing check to make sure that the packet's length is no greater than the datagram's length was missing from the code. And that allowed a large packet to overflow a buffer in a classic remotely supplied buffer overrun attack which could be leveraged into a remote code execution, basically very trivially.

Now, the good news is last summer Steam independently, or Valve rather, independently, separate from this, added to the Steam system ASLR, Address Space Layout Randomization, which from that point on would have made exploiting this much more difficult. Not impossible, because we've seen various ASLR bypasses which can be done, but it's much more difficult. Anyway, just another example of how longstanding protocols can have errors. They're kind of a little suspicious sometimes. I mean, like the test for successive UDP fragments is there, but not the test for the first one. It's like, okay.

Well, there is a different code path for the first fragment in a series of them, so you could explain it that way. But of course what you always wonder for a problem this simple to implement, I mean, like this is way easier than Heartbleed, that you didn't know maybe you were going to get data from a server or not. I mean, that was a very low probability key extraction, and the world had a meltdown over it. Here you just send a malformed UDP packet to any of 15 million Steam clients, and you're running code under the rights of the Steam client, which is, you know...

Leo: That's terrible. That's terrible.

Steve: It is. And so it's been there for a decade. And so you have to wonder, did anybody else ever know about this? And was it used? The good news is it won't be any longer. And to Valve's credit, they fixed it immediately, like within hours of being informed. So credit to them. And, yes, everybody can make a mistake.

Leo: The scary thing about this is that Steam typically on a Windows PC is running all the time. Once you install it, it's just sitting there in the system tray.

Steve: Yup. Woohoo.

Leo: It's not one of those things you run it and then close it, yeah.

Steve: Yeah. So it's like having your own little open vulnerability on the Internet, unfortunately.

Leo: Yeah.

Steve: Speaking of open vulnerabilities on the Internet, it turns out we talked a couple weeks ago about these Redis servers, R-E-D-I-S.

Leo: Redis. We use Redis, actually.

Steve: Redis, okay, yeah. They are RAM-based indexed data stores which are, you know, they're very cool. They are used normally...

Leo: Caching.

Steve: Yes, exactly. It is sort of a big index meant to be used on an internal Intranet, never meant to be exposed to the Internet. That is, it's not a web server where it makes any sense at all for it to be publicly exposed. It's meant as an internal cache. As a consequence there is, believe it or not, the default installation. Well, I mean, yeah, it is believable. The default installation assumes that position, that is, a non-publicly exposed position, so there is no network or authentication security enabled by default. They are wide open. So naturally, in the world that we have, rather than the world that we want, although it was never their designer's intent, many thousands of these servers, and actually somewhere around 72,000 of them, to give it a number, are sitting happily out on the public Internet for anyone to horse around with, even though not one of them should be.

Okay. So Imperva Security, whom we've spoken of often, they're an active Internet security, network-based security firm, they've been running a network of Redis honeypot servers in order to observe and characterize the behavior of the attacks which are coming against them. As a consequence of their honeypots, they discovered that these servers have the SSH, the Secure Shell authentication keys installed so that persistent future access could be obtained if the servers were ever to be secured in the future. That is, if they're not taken off the public Internet, but if someone goes oh, my goodness, we've got malware in our Redis server, let's add authentication so that no one can make an unauthenticated access.

Well, so what's happened is the bad guys have already taken advantage of their access

to install persistent SSH keys so that presumably they'll be able to telnet in with authentication that they have provided for themselves. So hopefully anybody, if the security of these servers ever comes on the radar of people for whom it already should be, they'll just wipe and reinstall. They're RAM-based servers, so there's really no reason not to just clean them up and bring them back up on the Intranet without any public exposure. They should not be exposed publicly.

Anyway, so it's funny, as I was thinking about this, I'm just struck by how bizarre today's Internet, I mean, how bizarre today's climate feels. Leo, like just 10 years ago during this podcast, when the podcast was almost three years old, because we're coming up now on the end of our 12th year, this was the stuff of science fiction. Like this kind of problem didn't exist. But here we are now, Episode 666 of Security Now!, and it's just like, oh, well, just another episode. Unbelievable.

Leo: Oh, I can't imagine what it's going to be like in 999.

Steve: Oh, boy. Okay. So Chrome 67 they did, as Google always does, moving things forward. One of the things that they moved backward, just like a day or two ago, I think it was late last week, was the removal of one of the experiments - because not all of their experiments fly, as we know. They do stuff. They see how they go. Sometimes they stick, sometimes not. This one did not stick: HPKP, HTTP-based Public Key Pinning. So the idea was kind of cool, but it never happened. In fact, the number I saw somewhere was 0.04% of websites ever tried this.

So the idea was that a web server with HTTPS security, so a standard SSL, now TLS, security certificate, having a public key, could publish the public key in its response headers to browsers. And it would be saying, this is my public key, sort of formally. The browser would store it, and thus any future visits where a connection was established with a server, if the browser had previously received a declaration of the server's public key, it would reverify the public key declaration with the public key now just received from the server. And we call that "key pinning." The key would be pinned in the browser or by the browser so that, if the public key did not match, the connection would be denied.

So the point is, this only makes sense if you rigorously enforce it, that is, if a site has said proudly, here is my public key. And the only reason it's saying that is it wants the security of HPKP, HTTP-based Public Key Pinning, of anyone who might spoof it to be denied because there's no way - as we know, if certificates are misissued, like WoSign or Comodo or one of these sort of sketchy CAs were to issue a certificate for the domain that they should not issue it for, the certificate would be valid, but the public key would be different. And there's no way for a third party to copy the public key because that's based on the private key that hopefully never does leave the server. So you can create a duplicate certificate that is valid for the same domain. You can never duplicate the public key.

Now, of course, the problem is there are many ways this could go wrong. If a bad guy somehow intercepted your connections, like with a DNS spoofing attack, or somehow arranged to get an invalid public key pinned in browsers, then that browser would never accept the valid connection from the website. I mean, the idea is this public key pinning only works if the valid public key is what gets pinned. If there's any way for a malicious one to get pinned, you're in trouble.

And we also know that sometimes websites are surprised by the expiration of their SSL

certificates, or their TLS certificates. From time to time people will say, I mean, you'll go to a website and get the notice that the certificate is expired. Well, not only are the website's visitors surprised, but the website's administrators are scrambling around like crazy, trying to get a new certificate issued. The problem is that new certificate will have a different public key because it's a new certificate. And if they had previously pinned the expired certificate, now they're in trouble because they can't use the expired certificate's public key in order to update the PIN that the browsers have previously obtained.

So the point is the only way to use this safely is to use it very responsibly. And people who understood the risk just weighed that versus the benefits and said, eh, no thanks. So it didn't happen. Consequently, no adoption rate, and it's been removed from v67 of Chrome. And I don't even - I meant to look to see whether Firefox ever even supported it. It may not have ever gotten off the ground beyond Chrome and Chromium-based browsers. So anyway, now it never will.

All of this is my introduction to the topic that we will end the podcast with, which is Certificate Transparency. In Google's own coverage of this, when they talked about this, they announced it in their developers.google.com. They said: "Deprecate HTTP-Based Public Key Pinning." They said: "HTTP-Based Public Key Pinning was intended to allow websites to send an HTTP header that pins one or more of the public keys present in the site's certificate chain." And then they said: "It has very low adoption; and, although it provides security against certificate mis-issuance, it also creates risks of denial of service and hostile pinning."

So anyway, then they said: "To defend against certificate misissuance, web developers should use the Expect-CT header." And we now know that CT stands for Certificate Transparency. So there is a new header, Expect-CT; and, they said, "including its reporting function." They wrote: "Expect-CT is safer than HPKP due to the flexibility it gives site operators to recover from configuration errors, and due to the built-in support offered by a number of certificate authorities." And we'll be talking about that in a moment.

I did want to just note the final availability, and I know you've been talking about this, Leo, of Intel's previously known as XPoint, but officially trademarked as "Optane," memory.

Leo: We were all excited, remember, when we heard about XPoint?

Steve: Yeah, yeah, yeah. It's exciting because, just to remind our listeners or to inform those who don't know, it's an incredibly dense, very high-speed memory, non-volatile, like flash. But if you can imagine a grid of conductors on the bottom layer, and then another grid at a 90-degree angle to them so that you form a bunch of intersections between the grid points. And there you put a little blob of goo, to be unscientific. This blob of goo has memory. So unlike flash memory, where it depends upon stranding some electrons to create an electrostatic charge that can influence a metal oxide semiconductor gate, this actually is a state change technology where a previous zap of current permanently alters the state so that the resistivity of that intersection changes.

The reason it's exciting is that it is nonvolatile and four times the density of DRAM, Dynamic RAM, which is what we have now, is the most, has previously been the most dense storage technology we have. But of course it is famously volatile, which is why we have hard drives, and we've got caches that are faster. DRAM is just persistently slow. So it's going to be interesting to see how this changes the architecture of computing over

time. I mean, it's a game-changing technology. You can buy Optane SSDs on Amazon right now. They're not that expensive. And there are some things that look like DIMMs, which are like 512GB, so half a terabyte of Optane memory for particular applications.

I think what this is going to do is it's going to change the way computer systems are structured because it has such a different set of speed and performance characteristics, and our existing hard drive to DRAM to three layers of caching to processor core to registers in the processor, all this kind of changes. So it's going to be interesting to see how it evolves over time. It's not clear.

Leo: Yeah, it's pretty exciting. I mean, we'll see what happens.

Steve: Yeah. We don't often see this kind of a, like, major potential changing technology hit our industry.

Leo: Yeah, I've been waiting for a laptop with Optane and an i9. But they're coming.

Steve: Yeah, yeah.

Leo: I was hoping for a Mac.

Steve: I got an interesting note from a Gary Foard, F-O-A-R-D, I guess that's how you pronounce his name, in England, on the first of June. He said: "Hi, Steve. I'm a big fan of your show and happy SpinRite owner. However, as your next episode is 666" - so he wrote this last week - "I think it is time to ask my awkward question." He said: "I recently stumbled upon an old screenshot of Norton's Disk Doctor, and I was surprised to see how similar one of the screen images look compared to the main graphic status display of SpinRite." He says: "Although I've followed your Security Now! show for many years now," he says, "I don't really know what your personal background, timeline, progression, education is. All I know is I see a sincere, experienced IT chap who does [and he has an acronym here] GAS." And he says, parens, "(Give A Sh*t, as you once said)." And he says: "I guess that is why we all keep listening."

So to his question, he says: "Did you work for or with Norton at some point? Did DOS-based graphics dictate the identical look? Or something else? Regards, Gary Foard."

And I don't know if I've ever talked about this. But unfortunately, I refused to sell SpinRite to Norton after a lunch that Peter and I had where he wanted to buy it. SpinRite was only about two years old at that point, and he said over lunch that it was the number one thing that the purchasers of the Norton Utilities wanted was SpinRite. And he said, "So we're going to buy it."

And I said, "Gee, Peter, I'm very flattered. But maybe when SpinRite is in its waning years that would make sense. But it's brand new, and it hasn't even - I'm not finished with it yet. And it's my bread and butter, and I don't want to sell it to you." And so he said, "No, really, name your price." And I said, "There's no price, really. I mean, except an insane price that you would never consider. So thanks for lunch, but no."

And so we went back up to their offices in Santa Monica and met someone who was the

CEO of Norton at that time. Peter didn't want to manage the company. He'd sort of stepped aside from management. But there was someone named Ron Posner who...

Leo: Oh, I remember him.

Steve: Yeah, yeah. Not a good person.

Leo: No.

Steve: Unh-unh. And so Peter and I walk into evil Ron Posner's office. And he says: "So, we have a deal?" And really, I will never forget the look on Peter Norton's face because it was like - it looked sort of like the dog that had been caught doing something bad, like digging up the flowerbed or something. I mean, he really looked kind of sheepish. And he said "Um, no." And Posner said: "What do you mean, no?" And Peter said, well, you know, "Steve doesn't want to sell." And Posner, like, I don't remember the details now, I mean, I wasn't happy either. It was all kind of awkward. But I said, "I'm sorry, but no." So maybe, I'm not sure now exactly what the timing was. But it was around the era of SpinRite 3, or I was working on it or something. And there was another guy, John Goodman, who actually wrote a book about SpinRite. There's a SpinRite book.

Leo: Really. I didn't know that.

Steve: Yeah, it got written. And in the process he was covering other hard disk utilities. And he interviewed someone who explained why it was that what they created, which was called the Norton Disk Doctor, looked identical. What Ron Posner reportedly did was hand a copy of SpinRite to one of their developers and say, "Go home. Stay home until we have our own version of this." And what really upset some of my developers, some of my people, was that I'll never, I mean, it annoyed me, too, was there were some functions in the BIOS where support was not guaranteed. And in my code I was testing for the presence of the functions by putting some arbitrary data into the registers and calling the function to see if the function modified the data. We looked inside Norton's Disk Doctor, and they had copied the code verbatim.

Leo: Wow.

Steve: Yeah. So...

Leo: Did you go after them?

Steve: No, no, because it's just not my nature.

Leo: Where did they get the code? Did they reverse-engineer it?

Steve: Yeah. They disassembled SpinRite. And what was sad was that they didn't even, like, innovate the screens. I mean, it looked like SpinRite. I mean, I know exactly what Gary, who wrote this email, said because everyone in the industry at the time assumed I had sold SpinRite to Norton. And the good news was they ended up abandoning it because, since they hadn't actually written it, they couldn't support it. And we ended up, their own tech support ended up sending their Norton Utilities customers to us because they...

Leo: That's nervy.

Steve: They needed help. And so we would say, oh, well, you know, we understand you have a problem. Buy a copy of SpinRite because we're actually the people who wrote apparently what you've got from Norton. And then we'll be happy to support you, fix your drive, and answer your problems. So, yeah, that's the story of Norton Disk Doctor, as far as I know, from third-hand reports. But, I mean, from first-hand knowledge, there was absolutely my code in their product because they didn't know why I had put those random numbers, those particular numbers in the registers to call the BIOS. And so they didn't dare change them. So it was like, wow.

Leo: Wow.

Steve: Okay, yeah. And I had some employees who absolutely wanted me to sue them. But I was a little guy, and they were Norton, and you end up in court forever. And it's like, no, thanks. I'll just - well, and it had no future. SpinRite's still going strong. So it's like, it was the best decision I ever made not to sell it to Peter. I mean, he would have paid me a couple, what, hundred thousand dollars back then or something? It's like, eh, no, thank you. So the right decision made.

Leo: Fascinating story. Wow. All right, Steve. It's time to learn about our subject of the day.

Steve: Okay. So we have talked throughout the history of this podcast about the way the certificate authority system functions. We have a, well, once upon a time a few, a handful of certificate authorities who were responsible and trustworthy. And their public key is contained in our so-called "root store" in our computer. And a website asks them, proves the ownership over the domain, TWiT.tv, GRC.com, whatever. And so the trusted certificate authority uses their private key to sign the certificate, which it is like a certificate. It says, "We certify that the holder of this certificate is the owner of this domain."

So they use their private key to make that assertion, and then that certificate is what the website sends when you connect to it, saying you're really connecting to us. And so it has two functions. It authenticates the other endpoint; and, as part of the negotiation with TLS, it provides both authentication and privacy thanks to encryption. So keys are negotiated and so forth. But the key part of what the certificate's doing is it's a self-expiring assertion of ownership of a domain. And if everything worked like that, then we'd have no problems. But there are problems with it.

First of all, there are no longer a handful of certificate authorities. There's a gazillion of

them. That's the technical term. And among them are many that have unfortunately misbehaved, in some cases deliberately. Was it Trustwave, I think it was Trustwave that deliberately issued an intermediate certificate so that a customer of its could mint certificates on the fly for a middleware box. That was the story. I'm still suspicious that that was the case. But they got caught. And as a consequence that certificate was revoked.

Normally a middlebox like that, like a corporation has on their network, requires that all of the clients receive its own certificate so that they're trusting things signed by that box because they're inside the network, not like all browsers in the world trusting all certificates signed by everyone. This is the problem that we have is that, as the number of certificates has exploded, we have to contain - our root stores, which used to have a few public keys, have to have the public keys of all certificates that we might encounter.

So the system didn't really scale that well. So there's the misissuance problem. There's the fraudulent issuance problem. And then of course there's also the revocation problem that we've talked about, where no one could have made a malicious mistake. But due to a mistake made by the holder of the certificate or the private key that goes with that certificate, that could escape, and they would then need to revoke the credentials of the certificate that had been signed.

So through the years there have been different attempts to solve this problem. We were talking earlier about HPKP and how that was an attempt to do Public Key Pinning, and that never really worked. And Chrome has had that CRLSet where it's kind of revocation, but it prioritizes only the certificates that they feel are important enough. Maybe those are the EA certificates, but not the EV certificates; and not OV or DV, not the organization or the domain validation certificates, and so forth. And then there's the OCSP, the Online Certificate Status Protocol, and so forth. So this has been an ongoing problem.

About four years ago, Google came up with another solution. And it's gaining traction, or I should say it's gained traction, rather quietly. And it's a nifty idea. It is going to still take some time to go from sort of advisory status to enforcement status because it does require all certificate authorities to participate for it to work, in the same way that all certificate authorities have to have their root certificate in the store, although in this case participation is probably a little more involved than that.

Okay. So what is it? Quite simply, it's a public log. It is a special, cryptographically secured, append-only log of certificates which are issued. And it's called Certificate Transparency because this public log intends to make the issuing of certificates be transparent. Right now, if random certificate authority XYZ issues a cert, nobody knows. I mean, it's a transaction between the domain holder and their CA, who they have chosen to pay for the signing of a certificate. And that signature is trusted by everybody's browser in the world. So it's a quiet transaction.

Notice that one of the things we've talked about is how the Chrome browser's ability to know if the Google certificates were valid gave it a lot of leverage. That is, over the years there have been a number of instances where somebody somehow got a non-authentic Google property, like Google.com, certificate. And a user of Chrome went to a site that was spoofing Google, and instantaneously the jig was up. Chrome does have pinning for the Google property certificates. And so if anybody tries to pull a Google.com certificate that isn't valid on a Chrome user, the Chrome browser instantly phones home to Google and says, whoa, we just received a certificate that should not exist in the wild.

And so as a consequence of Chrome's dominance and Google's special position in the

world, there has been this particular closed loop. So certificate transparency is sort of an effort to extend that model globally. There are three components to the CT system. There's a certificate log. Then there are certificate monitors for the logs. And there are certificate auditors. And I should mention that there's a site where all this is explained. It's www.certificate-transparency.org is where this stuff lives. Everything is open and open source. And this is another significantly useful effort which Google has created and is helping to manage and run. And it seems to me like it's the future.

Google writes, as they're explaining it, about the logs, the monitors, and the auditors. They say: "These functional components represent discrete software modules that provide supplemental monitoring and auditing services. They are not a replacement for, or an alternative to, the current SSL certificate system." In other words, all of that stays in place. This is a means of watching the certificate authority system operate and catching mistakes.

Google continues, saying: "Indeed, these components do not change the fundamental chain-of-trust model that lets clients validate a domain and establish a secure connection with a server. Instead, these components augment the chain-of-trust model by providing support for public oversight and scrutiny of the entire SSL certificate system." And that's why this is so cool.

So they say, under "Basic Log Features," they say: "At the center of the Certificate Transparency system are certificate logs. A certificate log is a network service that maintains a record of SSL certificates. Certificate logs have three important qualities. First, they are append-only. Certificates can only be added to a log. Certificates cannot be deleted, modified, or retroactively inserted into a log. They are cryptographically assured. Logs use a cryptographic mechanism known as Merkle Tree Hashes to prevent tampering and misbehavior. And, finally, they're publicly auditable. Anyone can query a log and verify that it's well behaved, or verify that an SSL certificate has been legitimately appended to the log."

They say: "The number of logs does not need to be large. There need to be enough logs so that log failures or temporary outages are not a problem, but not so many that they become difficult to monitor." And Google says, "Say more than 10, but much less than a thousand. Each log operates independently of the other logs," which is to say there's no automatic replication among the logs.

And they said: "The append-only nature of a log allows it to use a special type of cryptographic hash to prove that it's not corrupt and that the log operator has not deleted or modified any certificates in the log. This special hash," they write, "known as a Merkle Tree Hash, also makes it possible for auditors to detect whether someone has forked a log or inserted backdated certificates into a log." And it's funny, too, because at some point as I was reading into this, I don't remember now where it was, it was some time ago, but the author of what I was reading said, "And, yes, it's inevitable that this is going to be compared to the blockchain, or a blockchain, or some blockchain, and that's not what it is."

So, however, of course the reason it's sort of subject to comparison is that it's got some of those sorts of features, a publicly auditable, non-modifiable thing, although it doesn't work the way a blockchain does. And it is a standalone, append-only data construction that I imagine we will talk about at some point in the future. And what I got a kick out of and what was significant to me also, as I mentioned at the top of the show, is my favorite certificate authority, DigiCert, was the first CA to get themselves going with a certificate transparency log. Since then, and that was February 1st of 2015, so more than three years ago, other CAs have been establishing logs. There is a Let's Encrypt log. And

the idea is that there is a JSON, a computer-readable list of logs, which allows CAs to publish the announcements of certificates that they're issuing in the log.

And so as I said at the top, once upon a time there was no knowledge of the issuance of a certificate until it appeared being received in a browser by someone going to a site that was asserting that certificate. This provides a repository of certificate issuance transactions over time which is publicly available; cannot be altered after the fact; and, once the system sort of has moved from the reporting stage to the enforcement stage, sort of solves this problem of the system not having sort of a closed-loop feedback into the issuing process. This closes the loop, providing a declaration that it can be machine searched quickly in order to determine what the status of any domain certificate is.

So I don't want to go into any further detail at this point. We're out of time, for one thing. But I imagine, as this continues to gain traction, we'll be coming back and talking about some of the very cool underlying technologies of the certificate transparency system. I've touched on the topic, just used the term a few times, but never really established what it is. So I thought with the opportunity of Chrome 67 removing another attempt at solving this problem, talking about the right way, the way that is clearly gaining traction, and it looks like it's going to win, is worth discussing. So now we have.

Leo: You bet it is. Thank you, Steve. Always interesting. I learn so much. And I understand so little. Steve does this show, and I'm here for the ride, every Tuesday about 1:30 Pacific, 4:30 Eastern, 20:30 UTC if you want to tune in live and watch us live. You can be in the chatroom at irc.twit.tv. Nice bunch of people in there. And you can also be in the studio. We had a lovely family here from St. Louis. Alan, Whitney, and Bennett were visiting. One of the three's eyes weren't glazed over. One and a half. But they made it. They survived. They got through it. Actually, Alan does security at a bank, IT security, so he kind of, I think, understands what we're talking about. If you want to be in-studio, you can. Just email tickets@twit.tv. You can also watch live on the stream at TWiT.tv/live. On-demand versions of the show available from Steve at GRC.com. That's his website. That's where you find the software even Peter Norton couldn't buy.

Steve: The original, for real.

Leo: But you can, for a lot less. GRC.com is the home of SpinRite, world's finest hard drive maintenance and recovery utility, and proudly written in assembly since 1984. When did you first write it?

Steve: I think that's right. It was the late '80s, yeah.

Leo: Wow. And still going strong.

Steve: Because I bought my home in '84, because that's of course George Orwell's book. And SpinRite was a couple years later, so like '86.

Leo: Nice, nice. GRC.com. There's lots of other good stuff there, too - this podcast;

handwritten, human-composed transcriptions which actually are great because it helps you, if you want to search for a bit of the show, you can search the text and jump to that part of the audio. We have video as well as audio at TWiT.tv/sn. All our shows are here on demand. And you can always find any show we do on your favorite podcast application. Just subscribe. That way it'll be in your inbox the minute it's available, Tuesday evenings, thereabouts. Thank you, Steve. And we'll see you next week on Security Now!.

Steve: Thank you, my friend. See you then.