SECURITY NOW!

Transcript of Episode #691

ECCploit

Description: Hackers and attackers apparently enjoyed their Thanksgiving, since this week we have very little news to report. But what we do have to discuss should be entertaining and engaging: Yesterday the U.S. Supreme Court heard Apple's argument about why a class action lawsuit against their monopoly App Store should not be allowed to proceed; Google and Mozilla are looking to remove support for FTP from their browsers; and from our "What could possibly go wrong?" department we have browsers asking for explicit permission to leave their sandboxes. We also have some interesting post-Troy Hunt "Are Passwords Immortal?" listener feedback from last week's topic. Then we will discuss the next step in the evolution of RowHammer attacks, which do, as Bruce Schneier once opined, only get better - or in this case worse.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-691.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-691-lq.mp3

SHOW TEASE: Hey, it's time for Security Now!. Kind of a slow week in security. I guess the hackers took Thanksgiving off. But Steve does have a "What could possibly go wrong?" segment, a picture of something that really did go quite wrong, and a look at the next-generation Rowhammer. They're calling it "ECCploit." Stay tuned. Security Now! is next.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 691, recorded Tuesday, November 27th, 2018: ECCploit.

It's time for Security Now!, the show where we cover your security and privacy online with our friend, it's kind of like your neighbor, Steve Gibson, if your neighbor were like the king of computing. Hello, Steve.

Steve Gibson: Hey, Leo. It's great to be with you again for this post-Thanksgiving episode of Security Now!. In gathering together all of the news of the week, I came up kind of short. Luckily we had a bunch...

Leo: Oh, good. I don't mind.

Steve: Yeah, I think the hackers and the attackers all...

Leo: They took the week off.

Steve: They enjoyed their long Thanksgiving week. Of course that's not, I guess, globally celebrated. But anyway, it was slow. We do have some interesting things to talk about. I wanted to give my own take, I saw that you guys led off MacBreak Weekly with it, to talk about what was initially my concern over Apple's presentation yesterday in front of the Supreme Court because we know how important it is to have a curated App Store. And I was like, oh, goodness.

Leo: That's a good point. There's a security angle on this.

Steve: Oh, it's like all - it's all there is. And we also have Google and Mozilla looking to remove support for, believe it or not, I mean, you will believe it because who uses it anymore, FTP from web browsers.

Leo: Oh, I guess, yeah.

Steve: It's not gone yet, but it's always been there.

Leo: It's still in there? What a surprise.

Steve: Yeah, I know. Like whoever uses that? Which is not to say there's anything wrong with FTP. Calm down, you old-timers. It has its purposes.

Leo: I still use it all the time, yeah. I use SFTP.

Steve: Well, and there are, yes, there are the alternatives people want. Then we have, from our "What could possibly go wrong?" department, browsers considering looking for permission to leave their sandboxes. And it's like, oh, really? We have some interesting post-Troy Hunt "Are Passwords Immortal?" listener feedback from last week's topic. And then we're going to take a close look at the next step in the evolution of Rowhammer attacks.

Leo: Oh, boy.

Steve: Which do, as Bruce Schneier once opined, only get better. Or maybe worse. Anyway...

Leo: More successful.

Steve: Yes. As a consequence, and this comes from our friends Herbert Bos and his team, although a different team, at VU [Vrije Universiteit] in Amsterdam. Anyway, they named it ECCploit. And when I was kind of looking at that, I realized, okay, how would you pronounce that? Well, ECC is "eck," so it's "exploit."

Leo: Oh, "exploit." Oh, clever.

Steve: Oh, and Leo, we have a Picture of the Week that, again, the old-timers here, they'll be like, "Oh, I recognize that. That looks like what I have." Anyway, a lot of fun I think this week for our listeners, and a bunch of good stuff.

Leo: Awesome. I guess do you want to do the Picture of the Week?

Steve: I've been staring at it. And I was imagining you accept a job, a new job as the IT guy at some company, and your boss says, oh, and we have an intermittent connection somewhere.

Leo: Can you find it? This is somebody's wire closet. But you know what? I've been in that closet.

Steve: Oh, I think we all have. I don't know what it is about these closets that evolve this way. Almost without exception they start off neat and planned. In fact, you can see on the back wall it looks like there are some nice loops of a black something, like they're all tacked down. There was like, somebody had this dream of we're just going to have the nicest wire farm. And oh, my goodness. I think my favorite one is this tan one that's coming across the walk path. I mean, it's one thing for them just to be kind of going down on the sides. But, yeah, you have to duck under this one that's going kind of diagonally from left to right.

Leo: Oh, lord. This is not as bad - we weren't as bad. But it is a little bit what it looked like at the Brick House because we rushed over there, and we added stuff. But when we moved here - and really kudos to Alex Gumpel, Burke McQuinn, and John Slanina, our engineering team; and Russell, too. When we moved here, we already kind of knew all the connections, all the boxes. We already had everything. And they took the time. And I'm proud to say our server room does not look like this.

Steve: Ooh, god. I just noticed. Look at the top in the middle where there's other wires that are looping over and pulling down.

Leo: Oh, no, no. That's ridiculous.

Steve: So they had to cross over, but they didn't want to block the walk path.

Leo: Oh, geez, Louise. Oh, man.

Steve: So they hung them over the other wires.

Leo: Oh, man.

Steve: It's, like, pulling them down. It's like, oh, yikes. Yeah, I think, you know, remember the old saying, no battle plan survives first contact with the enemy. And there's an analogy there. It's like you initially know where everything should go, and you lay things out, and then something comes along, and you've got to change things, and you don't have time. So it's like, okay, well, I'll just do this. Just a kind of hack here, real quick. We'll just run these wires over here, and then I'll get back to this when I have the time. And then something else happens before you are able to fix that last one, and it piles up. And the result is this.

Leo: This is our old wire closet before we moved. And, well, it's not that bad. Although if you go around behind it, it is pretty...

Steve: That was the basement below?

Leo: Yeah, this is the basement in the Brick House.

Steve: Yeah, it was, well, and you guys really, you had an advantage of an entire floor that was essentially your wiring closet.

Leo: Right.

Steve: I mean, you were able to drill down wherever you were, and you'd be down in a completely available space to then run cables to somewhere else.

Leo: Do you have a picture anywhere of - by the way, there's the engineer, Scotty, who is the patron saint of our closet. Do we have a picture anywhere of our current setup?

STAFF: I can go take one.

Leo: Go take one and send it to me because I want - I'm very proud - John just came in. I'm very proud of what our team did because they made it look beautiful.

Steve: And it is a challenge.

Leo: And you know it's important because if you have, exactly as you said when you began, if somebody says, hey, there's some intermittent problem, you don't want to be the one that has to go in there.

Steve: It's like, I can't even see the lights. What are you talking about? They're just, like, totally obscured. So, okay. So while we're waiting for the photo of...

Leo: Much improved.

Steve: ...the pride of your current wiring closet, I wanted to bring up a topic that you guys mentioned on MacBreak Weekly at the top of that podcast, which I took from a Security Now! angle.

Leo: Yeah. And that actually - I'm thrilled you did because we were discussing the Supreme Court hearing yesterday over the - really it was more about the standing of the plaintiffs in a class action lawsuit against Apple, claiming that the App Store was a monopoly.

Steve: Right.

Leo: And that people should be able to buy apps from other sources on iPhones.

Steve: Exactly. And that's of course what set me off because we know how unable to make those kinds of decisions end users are. They're just not - they shouldn't have to be, and they're not, equipped to protect themselves. So at issue is whether Apple's App Store, through which as we know it alone can offer and sell applications for its mobile devices, constitutes a monopoly and gives it monopoly power over all sales of iOS platform apps. And as you mentioned before correctly, the lawsuit, which is known as Apple Inc. v. Robert Pepper et al., it's a class action. And I feel as we all do, and as you said on the podcast, these things are generally just for ambulance-chasing attorneys to line their pockets, and the people who are members of the class typically receive pennies.

So this suit has been around for almost a decade, and it argues that Apple's 30% cut, which they take of all App Store sales revenue, is excessive and anti-competitive and amounts to price gouging since consumers have no alternative other than to purchase apps for their devices at Apple's "inflated" prices. On the other hand, come on, like, really? But this issue is, as I said, of interest to us on this podcast because we've often noted that consumers can act unwisely and may be poorly informed about the true dangers of obtaining applications which have not been rigorously scrutinized, vetted, and curated - which is, I think, a vital service that Apple performs, and not easy. I mean, we've often talked about how difficult it is. And yes, they make mistakes.

But largely, for example, it is that and some of the other lockdown, one could argue, monopolistic things that Apple does are what makes the iOS a more secure platform today than Android. And we know that Google has been making efforts to sort of catch up in that regard. So as is often done in the law, and as you mentioned, Leo, Apple is not defending itself against that allegation. But their attorneys said, oh, here's how we'll spin this. What they've been arguing is whether the consumers who are bringing this complaint have grounds for that complaint in the first place. And this is known as "standing." Apple is arguing with the plaintiffs, essentially, of these litigants, whether or not they have standing to sue Apple over any of this. So, I mean, aside from even looking at the issue.

Apple's taken the position that only its developers who are being charged a 30% sales "commission," as Apple frames it, on the retail sales of their apps, would be damaged if Apple's conduct was found to be unlawful, and that it would be up to them to complain, that is, "them" the developers, not the end consumer. The precedent which the Supreme Court is revisiting was - and that's the reason it's gone to the Supreme Court after Apple has already won in the appellate court. I mean, so the U.S. Department of Justice has already come down on Apple's side. So it's like, okay, we're taking this to the Supreme Court.

So the precedent was set back in 1977 in a case known as Illinois Brick Co. v. Illinois, which was a dispute in which the court ruled in favor of concrete brick manufacturers. The state of Illinois had sued the brickmakers for allegedly inflating their prices, causing an increase in the cost of public building projects, which of course the State of Illinois was upset about. The court ruled then that, even though the increased brick costs might hurt Illinois indirectly, only the contractors who actually bought the bricks had standing to sue. And so that established what has become known as the Illinois Brick Doctrine, which gets hauled out every time an issue like this comes up. And that says that only the direct purchaser of a good can collect damages from a monopoly holder.

So as I said, the U.S. Department of Justice has already ruled that the Illinois Brick case was controlling here, whereas they're arguing that the Court of Appeals misapplied the Illinois Brick Doctrine, which of course is what they want the Supreme Court to agree to. So Apple argued yesterday that it is not directly selling apps to iPhone users, rather that they're acting as an agent for app developers who ultimately are selling their wares to consumers. And so they're saying, in exchange for the commission Apple takes on app sales, the company provides access to its vast user base and performs other services such as malware detection and so forth.

So anyway, we know where we are on this. One way or the other, it would be a disaster if, I mean, we're a long way away from this happening. But it would be a disaster if the government ever forced iOS apps to be available from anywhere that a user wanted them. The presumption is, oh great, we'll save 30%. First of all, no. It's not, you know, you're not going to get something useful for less than $0.99 or free with in-app purchases. That's just not going to happen. But mostly we lose the ability for Apple to control this very attack-prone surface that our mobile platforms represent and just stop being able to protect consumers from themselves because we just know people would just be downloading stuff from everywhere, and it would be a disaster. So, yikes. And, you know, that's pretty much what you guys said on MacBreak. I thought it was interesting though, that bricks came into it.

Leo: Yeah, I mean, the disaster you're describing is what happens on PCs today. You can download an app anywhere, from anywhere, on a PC. Microsoft has tried with Windows S, Windows 10 S, to make a version of Windows that is only available to the App Store. And the first thing anybody does who gets a PC with Windows 10 S on it is disable that so they can download Chrome and other things. I mean, when you get Chrome, you're not getting it from the Microsoft App Store, you're getting it from Google.

Steve: Yeah.

Leo: And I think the right to do that is great. But maybe a mobile platform is a little bit different and should have a restricted App Store. I honestly think, if that's what you want, you should get an Android phone that's more appropriate.

Steve: Right, right, right.

Leo: But it's an interesting case. I mean, a class action case is not going to change anything.

Steve: No.

Leo: Even if they win. Oh, let me show you our wire closet.

Steve: Oh, yeah, yeah, yeah.

Leo: This is the current - so let me just - just so you remember the wire closet...

Steve: Oh, come on, really?

Leo: Yeah, from earlier, this is the Picture of the Week. This is our - it's called a Video Hub, which has all the cameras are going into one side and then all of the outputs are going to another side. The whole server room looks like this.

Steve: That's a thing of beauty.

Leo: Isn't it nice?

Steve: That really is.

Leo: They're not even showing the best stuff, and there's some beautiful cable bundles and things going through conduit, those cable trays up above, everything is done beautifully.

Steve: Very nice.

Leo: And it's important to do it that way. We even custom cut these RCA cables to be custom lengths so you wouldn't have a lot of extra stuff.

Steve: Yup. Very nice.

Leo: They did a beautiful job. I just want to make sure they get - Alex Gumpel and Russell and Burke McQuinn get credit because they just did a fabulous job.

Steve: Sure, sure.

Leo: Yeah. It's clean; right?

STAFF: They're not RCA.

Leo: They're not, I'm sorry, did I say RCA? They are not RCA cables. They are, what are those things called? BNC, thank you.

STAFF: It's video, SDI.

Leo: SDI BNC cables. So those are the ones that lock. We don't use any of that cheap RCA crap. Isn't that nice? I'm proud of them. I wanted to give them kudos because they did a nice job.

Steve: Now, if we only knew who was running that closet that's the Picture of the Week, we could send your team over there and...

Leo: Oh, no. They can't have them. They can't have them.

Steve: You'd see them about five years later after having nervous breakdowns.

Leo: This is only because we moved; right? We were able to - because nobody was going to go in that old rat's nest until we moved.

Steve: That's right. That's what you do. Once it gets out of control, you just find a new location.

Leo: Just move.

Steve: Exactly. And you just do it, okay, now this time we know how to do it. So, very nice.

Leo: I mean, those labels, guys, didn't we give you a Brother label maker? Couldn't you do a better, I mean, come on. They're a little handwritten. But other than that.

Steve: So Bleeping Computer had an interesting posting about the endangered status of web browser support for the age-old FTP protocol. And I'm sure, again, the old-timers among us have put in ftp:// and the name of an FTP server, and today's browsers will understand that, will act as an FTP client, will bring up a connection to a remote FTP server - for those who don't know, that's File Transfer Protocol - and display the files that are in a directory. And you can click on a file and download it. So, I mean, it's not that different than a browsable directory on a web server, on an HTTP server.

The trouble is that FTP is barely and rarely used anymore by the mass of Chrome and Firefox users. I saw some stats that were something like 0.1% over some period of time. So it's not absolutely zero, but it's like - the problem is that there have been various attempts to secure FTP in the way that HTTP was secured. But they never took hold. And there are some technical challenges. I recall talking about the NAT traversal challenges of supporting FTP many years ago on this podcast.

The trouble is, unlike HTTP, where a single connection is made to a server, FTP by default, there is a variant that is known as "passive FTP," but its normal operation uses a pair of TCP connections, one for the so-called "control channel" and a second connection for the data channel. That second channel is allocated dynamically where the FTP server that answers the incoming data channel connection, in its response after they negotiate, tells the client which service port it should use to connect to for its data link. Who knows why that was the way it was designed in the old days, but that's the way it was designed.

So this of course creates a problem for firewalls and NAT routers, which adopted the practice, which is what we talked about years ago, and which remains in use today, of actively snooping on the FTP protocol control messages in order to determine which secondary connection they need to allow for inbound. So when they see the server that is behind them telling an external FTP client, oh, make your inbound data connection on port whatever, the NAT router sees that and goes, ooh, and opens up. It performs its own NAT mapping, opening up on the fly a port to receive the now-expected incoming FTP client data connection.

So, okay, now, that gets complicated when you add security. In fact, it becomes impossible if the FTP control connection is encrypted using something like TLS, since what is then a deliberate man-in-the-middle firewall or NAT router cannot determine the TCP port number for the secondary data connection which has been negotiated between the client and the FTP server because it's encrypted.

So there have been, as I mentioned, some attempts to secure FTP. But mostly, first of all, they have succeeded, like doing FTPS. Simply layering TLS on top of FTP in a way analogous to the way we added TLS to HTTP wouldn't work for that reason, but the fact that we're currently depending upon being able to snoop on that traffic in order to handle the FTP protocol in its standard mode. And of course there's a whole issue of the web browser's attack surface being kept open and expanded for this use for a protocol that virtually no one uses through their browser any longer. The point is that the browsers are currently supporting this and thus have an available surface for attack that they have to keep maintaining, even though it's not in use.

So it's not that FTP itself isn't a useful application protocol for cross-Internet file transfer, but first of all, standalone full-featured FTP clients are available for every platform. I don't know if our listeners know, but Windows has long had an FTP client command line. If you open a command line and enter FTP and hit Enter, you're in an FTP client. And if you type "help," you'll get a large list of available commands, and you can do FTP stuff. And then if you type "quit," you'll exit the client and come back to the Windows command prompt. And of course all operating systems that are in use today on some level support it.

And so what's happened is that the insecure and unsecurable FTP has become sort of the lowest common denominator for Internet file transfer. First of all, oftentimes just HTTP with S, HTTPS will allow you to download files. GRC has some directories where I explicitly make them available for developers to grab stuff that I'm working on. When I was working on SpinRite 6, I had a SpinRite 6 directory, and people could browse around in that. And in fact I had to deliberately put a text file in the directory starting with an underscore so it would be at the top of the list that says "This directory is deliberately publicly accessible." Because I would get helpful notes from people saying, "Steve, you realize you've got an exposed directory on your...." It's like, yes, yes, yes.

Anyway, so the point is you can do this in a secure fashion already with HTTPS if all you want to do is make files available for download, and even with directory browsing and the other things that FTP allows. And as you mentioned, when we were talking about this at the top of the show, there have been successful variations. There is SCP, which is the Secure Copy.

Leo: Actually, that's the one I use. I use that one all the time, yeah.

Steve: Yes, yes.

Leo: That's SSH; right?

Steve: Exactly. And then there's another one. There is an SFTP, which is another use of SSH for file transfer. There's even FISH, F-I-S-H, which is File transfer over Shell protocol. So there have been solutions to this. It doesn't affect anyone today. But just sort of as an early note to our listeners, if something you're doing, and I can't imagine why, but it seems like, okay, if something you're doing requires FTP through a browser, just an early heads-up that both Google and Mozilla are kind of looking at this thinking, you know, we're going to have to retire this somehow. So it may well be that at some point in the future we're discussing the announcement that version X of Chrome and Y of Firefox are planning to remove their native support for this venerable and increasingly obsolete protocol. So it may happen.

From our "What could possibly go wrong?" department, you're not going to believe this, Leo. I just like, okay. There is a discussion underway at Google and on public forums of the so-called "Writable Files API," intended to simplify local file access. Okay. So from our "What could possibly go wrong?" department...

Leo: What could possibly go wrong?

Steve: Okay. So I think this is actually going to happen at some point. So yikes. "The Writable Files API: Simplifying Local File Access" is the heading of this sort of proposal. And I'm just reading from Google's - I've got the link here under the developers.google.com. "What is the Writable Files API?" They say: "Today, if a user wants to edit a local file in a web app, the web app needs to ask the user to open the file. Then, after editing the file, the only way to save changes is by downloading the file to the Downloads folder, or having to replace the original file by navigating the directory structure to find the original folder and file." They say: "This user experience leaves a lot to be desired and makes it hard to build web apps that access user files. The writable" - oh. Oh, good. Okay.

"The Writable Files API is designed to increase interoperability of web applications with native applications, making it possible for users to choose files or directories that a web app can interact with on the native file system, and without having to use a native wrapper like Electron to ship your web app." In other words, Electron creates an environment that a web app can run in natively so it's not in the browser. Google of course is wanting to release the restraints on web browsers.

They say: "With the Writable Files API, you could create a simple single file editor that opens a file, allows a user to edit it, and save the changes back to the same file; or a multi-file editor like an IDE or CAD-style application where the user opens a project containing multiple files, usually together in the same directory. And there are plenty more."

Okay, now, as I was thinking about this, I thought, rather than filing this in the "What could possibly go wrong?" department, maybe this should really have been in the "We should have seen this coming" department. You know, our web browsers first began pulling crap from all over the Internet to fully monetize the pages that we visited, sometimes by mistake. Then they got us hooked on JavaScript so that nothing works anymore without it. But compared to native OS apps, JavaScript performance was bad.

So first we got NaCl - that was the native client - which was very clever, but never achieved critical mass. Then consensus was finally reached around Web Assembly, so that now code running from random sites on the Internet that we visit can much more efficiently run Monero mining on our machines while we're trying to figure out what that popup says that's asking for permission to store cookies on our browser after we leave the page that we visited by mistake.

So what we see is a history of desperation on the side of the web browser apps to become a first-class native-born citizen of our OSes. And we know that the whole sandboxing thing has to really be chafing the web browser guys. They want to be real apps on our machines. They don't want to be stuck behind any safe enclosures. They've got speed now. So next up is access.

Anyway, so there is a discussion. And the good news is they recognize they're paying lip service to the security issues. Under "Security Considerations" they said: "The primary entry point for this API is a file picker, which ensures that the user is always in full control over what files and directories a website has access to." Okay. Good. "Every access to a user-selected file, either reading or writing, is done through an asynchronous API, allowing the browser to potentially include additional prompting and/or permission checks.

"The Writable Files API provides web browsers with significant access to user data and has potential to be abused. There are both privacy risks, for example, websites getting access to private data they weren't supposed to have access to; as well as security risks, for example, websites able to modify executables, encrypt user data, and so forth. The Writable Files API must be designed in a way as to limit how much damage a website can do, and make sure that the user understands what they are giving the site access to." Oh, lord. So there is, on GitHub, there is a discussion of this and an explainer. And although the following is a little bit redundant, we have time, so I want to explain what they're thinking.

So under "Proposed Security Models" they say: "By far the hardest part of this API" - okay, now I would just say no would probably be a good idea. But they said: "By far the hardest part for this API is of course going to be the security model to use. The API provides a lot of scary power to websites that could be abused in many terrible ways." Okay. People, I'm saying to Google, are you hearing yourselves? It's like, yes. And it's going to happen. But okay. But, oh, we can write a multi - we could do a CAD app. We could do a multi-file editor. We could open a project that then has access to all the files in the project. What could possibly go wrong?

They said: "There are both major privacy risks (websites getting access to private data they weren't supposed to have access to), as well as security risks (websites modifying executables, installing viruses, encrypting the users' data and demanding ransoms, et cetera). So great care will have to be taken to limit how much damage" - how much damage. Yeah, let's limit how much damage a website can do. By all means. I'm for that.

And they say: "And make sure a user understands what they are giving a website access to." Oh, yeah, right. Our moms are going to understand this? Can you just see the website saying, in order to do what you want, we need to have - oh. Anyway. "Persistent access to a file could also be" - get this. "Persistent access to a file." So they're suggesting that websites have persistent access to files on our systems. "Persistent access to a file could also be used as some form of super-cookie; but of course all access to files should be revoked when cookies/storage are cleared, so this shouldn't be too bad." Oh, okay.

"The primary entry point for this API is a file picker, i.e., a chooser. As such, the user always is in full control over what files and directories a website has access to.

Furthermore, every access to the file, either reading or writing, after a website has somehow gotten a handle, is done through an asynchronous API, so browser could include more prompting and/or permission checking at those points. This last bit is particularly important when it comes to persisting handles in indexed databases. When a handle is retrieved later, a user agent might want to re-prompt to allow access to the file or directory." So this suggests that they're proposing that websites would, by default, have enduring access to specific files or projects or whatever on the user's machine.

They say: "Other parts that can contribute to making this API as safe as possible for users include limiting access to certain directories." Oh. Well, that would be good. That would be handy. "For example," they say, "it is probably a good idea for a user agent" - that is, the browser - "not to allow the user to select things like the root of a file system" - you think? - "certain system directories, the user's entire home directory, or even their entire downloads directory. Also, limiting write access to certain file types. Not allowing websites to write to certain file types such as executables will limit the possible attack surface." And then "other things user agents come up with."

Okay. So, you know, we've often observed here that just because it's possible to do something, doesn't automatically mean that it should be done. So it occurred to me as I was putting this together that a better name for this proposal might be the Security Now! Four-Digit Podcast Numbering Assurance Act of 2018, since this guarantees that we are never - we're going to get into four-digit Security Now! podcast numbers, if we give websites the opportunity to say to their users, "Okay, now, here's what we need you to do." Oh, lord.

So for our more technical listeners, there is a discussion of this with a lot of opinions echoing mine, like just back away from your computer, you developers. Find something else to do to add to our browsers. So I've got a link to the discussion at discourse.wicg.io, where anybody who's interested can throw their two or three cents in, if you want to spend some time and just look at what's going on over there. I just - oh, yeah. I mean, these show notes are produced in Google Docs. I download the PDF for them, and that's where I get the PDF that is then posted online, which Leo is looking at right now, and our listeners are looking at, those who bring it up while the podcast is underway. And I know that lots of people grab it.

So it's absolutely the case that web apps are useful. And we're seeing, now that we've got scripting, now that we've got WebAssembly that allows very close to native performance of an app running on code that we got from a website, I mean, I can see the next step is to allow this thing to step out of its boundary into our file system. That's, like, that's the last straw. But I'm afraid it might be true. I mean, I'd be tempted just to turn that off because, well, we'll see how it develops.

But the good news is they clearly recognize the danger. But the one thing we see that is the thing we have never been able to solve is social engineering attacks. Phishing attacks continue to be successful because they leverage people's inherent trust in - they just assume that there's authority coming from the other side. These computers, they don't understand the workings. They don't understand. They want something to work, or they want to do something, and the website says, oh, your version of Flash is out of date. Click here to, I mean, that's how all of these problems happen. And so you could just imagine the website that says, oh, in order to do this, click here and then go do the following. And in order to be useful, the app has to have access to some directories on your file system, and that's what it's going to try to get.

So I don't know. I get it. But it really does sound like it's not a matter of the technology being a problem. It's the users who will unwittingly follow instructions. Historically they've been following instructions to download updated Flash players to do what they want, or click here to get what they want. One of the core pieces of advice from the podcast that I really, I just love this one, is never download anything you are told to. Never download anything you didn't go looking for, meaning if something says you need something, just say no. Just no. It's not worth the risk. And this to me, I think it's the social engineering abuse potential which this raises. I mean, yeah, someone could say "Download this and run it," and we can already do that today. Maybe this isn't any more dangerous. But bad guys keep coming up with clever ways to trick users.

Last week's podcast asked the question "Are Passwords Immortal?" And we shared Troy Hunt's blog post. It generated a huge, really interesting set of responses from our listeners, both through Twitter and the mailbag, and I wanted to share four of them.

Dan Stevens wrote: "Hi, Steve. I listened to Security Now! Episode 690, where you discuss Troy Hunt's blog post about passwords, with great interest. I think good points were made on both sides. But I'm optimistic SQRL will eventually become widely used. When credit cards came out in the 1970s," he says, "before my time, I bet everyone was saying 'They'll never replace cash,' and they'd be right. Cash is still widely used even in developed nations. Yet over the last few years, perhaps since contactless payments became available, I've needed cash less and less, so much so that I now leave my wallet at home and just carry my payment card.

"I think this shows if people are able to perceive the benefits of a new system over the status quo, a large portion will eventually switch to it. It's just a matter of time. Yes, passwords will never die, but they don't need to. If SQRL becomes widely available enough for me to 'leave my password vault at home,' so to speak, that's good enough for me. I'm really looking forward to the official release of SQRL and excited to see what becomes of it."

And I really thought that was a great analogy. I'm a huge, for what it's worth, credit card user. And I do remember, he was right, when credit cards first happened, there was some skepticism about it. I mean, it was like, what? And it was weird, and it was like, not everyone took them. And you had to ask restaurants, do you take this, or do you take that? And it was like, you know, there was - definitely it was anything but friction-free. Yet they did happen. And they are now, I mean, that's the way we buy stuff on the web. And that's a huge, huge portion of commerce. Yet cash is still in place. And so really I thought that was a great analogy for the notion of the adoption of something new over time.

And on the issue of friction, one thing I forgot to mention when I was talking about this last week, but I just did want to note, is that, unlike any of the other multifactor systems, SQRL's setup is needed once. No question about it. You've got to get the app. You create an identity. It asks you to do some things to protect yourself, to write down what we call a "rescue code," which is your "Oh, crap, I forgot my password," your "get out of free" pass. You can back up or print out your master identity and fold it and put it away in a drawer so that you have it. Normally you'll spread that one identity across your devices. We have a means of importing and exporting identity among computers and smartphones and things. So you sort of get an automatic backup of your identity in a multi-device mode that many of us have today. But the point is there is some one-time stuff.

But I will argue, and everyone will soon see, that is literally the only friction you ever get. And so the point I really didn't make last time is that, unlike any other multifactor system, all subsequent use is literally zero friction. I mean, even multifactor, where you're having to snap a QR code or write down the key for your time-varying multifactor authentication. Each time you do that, every time you set that up with a new site, that's certainly not zero friction. So much so that I've discussed how to solve the problem of setting up a new device that now needs to know what all of your codes are. And my recommendation was, while you have that device or that site's QR code up, print it so you end up with a sheaf of papers that you can easily allow another device you're setting up to see.

But the point is, after this is just - after identity is created, the real joy of SQRL, what it really does, is from then on nothing can touch it. Nothing can compete with it in terms of ease. And I think that will probably be the main reason it succeeds is in the example that I gave you go to a blog site, and they say, oh, we need to know who you are. It's just like instantaneous to do that. And once people see it, it's like, okay. We need everybody to offer this.

Don Williams said: "Does SQRL have an option for the following scenario?" He says: "My employer contracts with a third party for services to be used by all employees. The third party permits access to only those users who can authenticate to the employer's security system. Therefore the third party is depending on the employer's authentication system."

And the answer is yes. The example I have given is, for example, imagine that in the future we had a U.S. identity system, that is, we wanted to try online voting; or, well, we do have things like the IRS and Medicare. We have government-provided services. The government could create a site, say identity.gov. And when you go to Medicare's site, or you go to the IRS, or you go to Veterans or whatever, those government sites could all use the SQRL domain of identity.gov. So you automatically get, within your control, centralized identity and authentication.

So it is entirely possible for one site to use the authentication of another site. You see that in the SQRL dialogue. It comes up and shows you the domain to which you are authenticating. So it's very clear what's going on. But this does allow for one site to use the identity of another. And in fact that's sort of the example that I gave last week in answering the question about what happens if Ford purchases some other car company and wants to move its users over. We're able to use that flexibility in order to make that happen. And the user who's doing it would see that they are authenticating to the other site in order to transport their identity across.

And, finally, James said: "Hi, Steve. I was delighted to hear my feedback being discussed regarding delegation of access on SQRL, and even more delighted to hear that it has been addressed already. Listening to SN-690, I was enjoying your discussion of Troy Hunt's blog post. You mentioned the importance of a frictionless process being key for the adoption of any potential successor to the current security model of username and password. In particular you articulated the often encountered situation where people will refrain from leaving a comment on a blog post or site simply because they do not have an account, and the process of creating one falls into the 'too hard' drawer." Or I would say "not worth it" drawer. "Consequently, people just move along, and their insightful comment or reply is lost to the world. Which got me thinking (again)."

He says: "The beauty of SQRL, as you have explained before, is that it allows a person to anonymously identify and assert said anonymous identity with a site. It allows them to do so quickly and easily. But," he says, "many, many, MANY sites base their entire business model on knowing something tangible about their visitors - their name, their email address, a human-friendly username, et cetera. In the case of ecommerce then, delivery addresses, billing addresses, and credit card information would also be required in due course. In short, most sites probably will not accept the creation of an account with just one anonymous handle. They will want more.

"My question," he says, "finally is, is there a provision in the SQRL spec to allow this to be handled in a 'frictionless' manner?" And then he proposes one. He says: "I envision or propose something along the following lines: On connecting with a site for the first time, the recognition is made that the user is unknown." Okay. SQRL has that.

He says: "The site passes its challenge to you, asking you to provide and assert your identity. Could at this moment the protocol also allow the site for the first time to request other information about you that the site would want to know? Your preferred handle, your email, your real name, your billing address, and even (cringe)," he says, "your CC details. The SQRL application could then display, one time only, a list of the details that are requested by the site and which of those are considered by the site to be 'mandatory conditions of account creation,' much the same way that Android requests app permissions at install time.

"The user could then choose what they wish to disclose to the site at that point, and those they wish to withhold. Or, if they feel that the site is being too nosey, they can withdraw, and the identity creation process is aborted. This would also likely assist in compliance with GDPR, as the user is being made explicitly aware of what information they are being asked to disclose and providing tacit consent in the process." And he goes on, but everyone gets the idea.

So I put that in because he raises a really good point, and it has been a source of a lot of discussion over the years in our SQRL group. And that is, what is it? The earlier specification made what I now consider to be a mistake of doing too much. There wasn't all of that in there, but there was more. And in one of the major revamps I made, it was in pulling way back from that. We decided that what it should be is, with very little exception, only authentication. That is, that's what it should - it should only be we should keep it clean and simple, and it just is authentication. Many people have observed, as James has, that sites want more. And that's absolutely the case.

It's also the case that, if it turns out people want more, it could certainly be added. But it's not how we're going to start at the beginning. I don't want this to be - I don't think it should be more than just a pseudonymous assertion of an anonymous identity, which solves and addresses all the cryptographic aspects. Because it's certainly possible then to do a form fill-out thing of some sort to add that or to have that be separate. I just didn't want to confuse the specification by bundling that in. And then it becomes - it's much trickier from a privacy standpoint, from a GDPR standpoint, from a multilingual standpoint.

And so the answer is no, we have decided to keep it with very little exception. There is one or two exceptions we'll eventually talk about. But I think it's better if it's just that. Because then the site can say, oh, hi. You've established your SQRL identity. If you would like to receive notification when someone responds to your blog posting, then we'll need an email address from you. Feel free to provide it and so forth. Different sites will have different requirements. I think it makes sense to let the site handle it at the site end, after SQRL is used to establish you as an identity, rather than mixing it all together.

And, finally, closing the loop. Joe Petrakovich said: "Shout out to @SGgrc Steve Gibson who I think gave me the idea to use Amazon S3 instead of Dropbox." He said: "Folder auto-sync job set up. Monthly bill reduced from $10 to $0.10." So I've mentioned before that I'm using S3 to store all kinds of stuff. And I have gigabytes of stuff. But because it's just archiving, my bill is like $2.43 a month, and it's just a great repository. So I just wanted to acknowledge, to thank Joe for the note and to note that that's an effective solution.

So I like the name "ECCploit."

Leo: ECCploit.

Steve: Rowhammer attacks against ECC protected memory. Okay. So to sort of remind ourselves and set the context for this, as we'll recall, it was Professor Herbert Bos, B-O-S, and a team of researchers at VU Amsterdam who were the first to demonstrate how potent and potentially practical Rowhammer attacks could be. And I'll explain again what Rowhammer attacks are. But at the time, everyone took some relief from the observation that parity-protected RAM would tend to catch and thwart any single bitflips, though not dual bitflips; but that at least the presence of full ECC error-correcting RAM, which is ECC, would provide much stronger protection.

Well, now Professor Bos and a different team of researchers are back with the results of their examination of just how safe we should feel about ECC's protection from Rowhammer attacks. Their paper is titled "Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks." And to give everyone a hint of what they found, the headlines in the technical press that covered this included "Rowhammer Data Hacks Are More Dangerous Than Anyone Feared," "Not Even ECC Memory Is Safe from Rowhammer Attacks," and "Potentially Disastrous Rowhammer Bitflips Can Bypass ECC."

Okay. So first of all, let's remind ourselves about Rowhammer. That was four years ago, in 2014, that this jumped onto our radar. And over the course of four years there has been a bunch of research produced that demonstrated that this problem is real. As we have said - in fact, we were just talking about it last week, the difference between SSD and DRAM. SSD is relatively permanent storage because electrons are stranded on a little chunk of conducting material, but stranded there by insulation, so that they have nowhere to go; and their presence, their electrostatic presence can be sensed, thus creating a permanent one or zero data bit.

DRAM, the reason DRAM is as dense as it is, where you can get just a phenomenal amount of data, billions of bytes of data on a strip of circuit board, is that the mechanism of storage cannot be further simplified. It is an itty-bitty little tiny capacitor which is just essentially two electrical plates separated by an insulator, technically called a "dielectric." And that capacitor is charged, meaning that electrons are pulled off of one side and put on the other to create this charge, which the plates are so small, and they are so close together, that there is leakage. And so this little capacitor, this imbalance in charge, tends to rebalance itself. This capacitor discharges rather quickly. Consequently, it's necessary to refresh the memory in the DRAM continuously.

The idea is that in order to store at such high density, we have sacrificed the ability to store data permanently in favor of storing much greater data. And that requires the cells to be small, that they cannot retain the data for long. So what happens is, before this grid has a chance to discharge, row by row the data is read out, and the capacitors are essentially recharged. The ones that have no charge in them have a zero bit, and so they're left discharged. The ones that have a one bit in them are read before they have a chance to discharge so that we can't tell that they have a one bit stored in them, and they're essentially recharged. So that's really what refreshing is. It's recharging the capacitors, essentially by dumping them all into an end-of-row buffer, and then reading what they had, and then recharging them all, and then going to the next row. Okay. So this is a large matrix of itty-bitty tiny cells that cannot be made any smaller, or they would have been. And they need to be constantly refreshed.

In the same way that hard drives are sort of on the edge of viability and rely on error correction to kind of bring them back out of the gray zone, thus the reason SpinRite is still useful today because it's not physical defects on the platters that SpinRite is now compensating for. Twenty-five years ago it was. I was dealing with physical storage problems on the platters. That's no longer the case. Now we're dealing with the fact that densities have gone up so high, and the drives have been engineered so tightly, that they are counting on error correction in order to keep the data readable. So, I mean, the analogy is exact here.

So what's happened is the noise margins are so tight in DRAM that what was discovered four years ago is, if somebody hammered on a row of memory, and by that I mean continuously read from the memory, that would create enough noise, electrical noise in the neighborhood that it could cause adjacent cells of DRAM to be misread. That's Rowhammer, hammering on a row, and the row next to yours could be misread. And the clever, again, remember attacks only ever get better, or worse, or stronger, or more powerful, whatever adjective you want to use, or adverb I guess. Wait. An attack? Yeah.

So anyway, what the designers realized was you could hammer on both sides of a target victim row and make an even stronger opportunity of inducing what are known as bitflips, actually flip the bit, cause a bit stored in memory to change. And then once they were able to do that, they figured out how to spray page tables through memory and cause the operating system to switch its page tables from the real ones to the ones they had created, and thus get write and read access to regions of memory that they weren't supposed to. So the point is, it turns out by starting at just, if we arrange to read from DRAM hard enough, over and over, they took that and went all the way to engineering truly effective attacks.

So memory can have a parity bit added to it. You can take, for example, a row of 64 bits of memory, which is generally the size that DRAM is read out in; and you can have, for example, a parity bit added to that 64 bits to make 65. And it's a very simple matter to have the hardware look at the bits that are being written, because the DRAM is also - it's read in a whole row and written in a whole row at a time. So it's easy to have hardware add a 65th bit, which will be set to one or zero depending upon the number of one bits, the even and oddness of the number of one bits set in the rest.

The point is that - and this is known as "parity." It's a well-known process. Back in the days of paper tape we often had seven channels of holes punched in tape, and the eighth channel was the parity bit. And the idea being that, if you punched the tape with, for example, even parity, then every row of eight had seven data bits to contain the ASCII characters, and the eighth bit guaranteed that the total number of punches would be an even number of holes. That way, when the paper tape was being read back, the paper tape reader would always verify even parity, and it would stop if it read something that was wrong. And in fact when we got to optical tape readers, you would see them stutter because they were able to read and reverse the tape direction. And so they'd back up and come forward again and might stutter a little bit in order to correct the misread of a row of tape. And so you dramatically improved your reliability.

So the idea with parity memory, which I want to talk about first, is that it would detect a problem. If you always know that any row of memory is supposed to have even parity, that is, the number you add up, you sum the number of one bits and include the extra one you added, it should always be even. And of course everyone should understand that you need the extra bit so that you don't - because you can't force the 64 bits to be even. If the 64 bits happen to have an odd number of ones, then you set the parity to one to increase the number, the total number of ones by one, thus making it even. So that's the simple logic.

And the point is, the parity memory allows for the detection of a single flipped bit in the row of 64 bits. Note that it does not detect two bits that are flipped because, if you change two bits, then you have not upset the evenness and oddness of the parity of the 64. But hopefully no bits are ever being flipped in the normal case. I mean, we need our system's memory to remember what we wrote there. And for the most part it does. The idea being, though, that if a cosmic ray, which is literally one of the causes of DRAM problems, you just get a random cosmic ray come shooting through your DRAM, and it'll blast the bits. You'd like in a system which is really caring about the work it's doing, it's better to have it stop and just say, whoops, we had a parity error, than to go blithely along and do stuff wrong.

So the next level up from parity is error correction. And error correction, the short version of this is it uses many more added-on bits, typically eight added to 64, so you get 72 bits in a row. And using some very complex math, which is also done in hardware, and interestingly is proprietary, it's possible to figure out which bit flipped and fix it. And it's tricky again because you've got 64 bits, and you have a relatively small number of ECC bits compared to the whole, and the math that's used is able to determine which one of any of the 64 bits was wrong, if one of those bits flipped. And whereas parity memory cannot detect a pair of bitflips, ECC memory will detect, but it's unable to correct, a pair of bitflips. So it just sort of ups the game on the whole DRAM error issue.

And so it was originally felt that parity memory would be protection, well, protection from Rowhammer to the extent that it would prevent abuse of single bitflips by stopping the system; by saying, whoops, we had an error here. Don't go any further. So bad guys couldn't get anything done. Then the Rowhammer attackers figured out how to do two bitflips and to make it practical. There was something I was reading when I was digging through this 17-page research paper that these guys wrote, where they talked about a number of instructions which were found in the pseudo command which is used in Linuxes and Unixes to elevate the privilege of a non-privileged user. There were, I think it was 13 locations where they had identified instructions where a pair of bits flipped in the instruction would, for example, convert the instruction from a conditional jump which was testing whether you had authenticated properly into a move, thus neutering the jump and essentially authenticating any user who then used that instruction to elevate their privilege and thus bypass authentication.

So, I mean, this stuff has been turned into practical attacks, amazing as it is. So the addition of error correction code was believed to really up the ante. It would detect single bitflips and correct them, and it would detect and not be fooled by dual bitflips.

So I'll share the abstract from their research to give you the context for this. They said, "Given the increasing impact of Rowhammer and the dearth of other adequate hardware defenses, many in the security community have pinned their hopes on error correcting code (ECC) memory as one of the few practical defenses against Rowhammer attacks." Meaning we haven't come up with a solution for this.

"Specifically, the expectation is that the ECC algorithm will correct or detect any bits they manage to flip" - meaning attackers - "manage to flip in memory in real-world settings. However, the extent to which ECC really protects against Rowhammer is an open research question due to two key challenges. First, the details of the ECC implementations in commodity systems are not known." Which came as a surprise to me. I just assumed we all knew about that. Turns out it's proprietary. "Second, existing Rowhammer exploitation techniques cannot yield reliable attacks in the presence of ECC memory.

"In this paper we address both challenges and provide concrete evidence of the susceptibility of ECC memory to Rowhammer attacks. To address the first challenge, we describe a novel approach that combines a custom-made hardware probe, Rowhammer bitflips, and a cold-boot attack to reverse engineer ECC functions on commodity AMD and Intel processors. To address the second challenge, we present ECCploit, a new Rowhammer attack based on composable, data-controlled bitflips and a novel side channel in the ECC memory controller. We show that, while ECC memory does reduce the attack surface for Rowhammer, ECCploit still allows an attacker to mount reliable Rowhammer attacks against vulnerable ECC memory on a variety of systems and configurations. In addition we show that, despite the non-trivial constraints imposed by

ECC, ECCploit can still be powerful in practice and mimic the behavior of prior Rowhammer exploits."

I won't go digging deeper into this except to note that these guys pulled this off. It turns out that ECC details are contained on chip and are undocumented by Intel or AMD. So it was first necessary for them, in order to figure out how to bypass error correction, to determine to reverse engineer the error correction algorithms, that is, to figure out exactly what math was used on the 64 data bits to create the ECC "tag," as it's called, to compute ECC in order to know how to flip bits deliberately that would bypass error correction. They did that through a brutal process of reverse engineering. They reverse engineered one AMD and three Intel chipsets to figure out what was going on. They recognized that normally the chipset that software is running on is made aware or is not obscured from software that is running on the hardware platform.

So the point being it's possible for software to know what hardware architecture it's on. That would then tell it what the respective ECC algorithm is, which would then allow it to choose the proper Rowhammer attack in order to flip bits on that particular platform. So while it is definitely the case that error correction code does make Rowhammer attacks more difficult, it does not solve the problem of Rowhammer. And one thing I learned from reading this paper, which just made me sort of close my eyes and shake my head, was it turns out that the refresh rate of DRAM, we have always known that it has a performance impact, that is, since refreshing is taking some of the RAM system's bandwidth for no good purpose, rather than just running through row by row by row, reading and rewriting that memory, that's going to consume some of the bandwidth.

Thus we had known that, whereas one of the mitigations for Rowhammer is increasing the refresh rate in order to have the cells' data be stored further from the noise margin, that is, having the one bits more firmly one, thus making it more difficult to have noise induce a flip, we've known that.

It turns out that there are instances where systems are very power critical. They absolutely have to run at low power. So it turns out that refresh is deliberately reduced because not only does it reduce performance to refresh at a high rate, it increases power consumption. So there are places where refresh rate has been deliberately compromised, and ECC is used as a prophylactic to - and I was of course again reminded of hard drives that have done the same thing. Hard drives are now relying on error correction to make up for the fact that they're no longer able to guarantee that they're able to read back the data without correction.

Similarly, it turns out, error-correcting DRAM is now being used as a performance prophylactic to solve the fact that, where power has been reduced, DRAM refresh rate has been reduced to compromise on power, and error correction is then assumed to be part of the system to bring back the reliability that was lost from having DRAM refresh rates reduced. Which, again, as an engineer, I just shake my head, and I go, well, okay, good luck to you.

So anyway, that's the news. The headlines were correct, although we're really out in the academic weeds at this point. Again, we're sort of where we were with Spectre and Meltdown in that the real danger is in shared hosting environments where bad software has an opportunity to run alongside good software and is able to, by being mischievous, get access that it shouldn't have across protection boundaries. It's true that we don't want bad software which we inadvertently run in our own workstations to have greater rein in our system than the OS wants to give it, and Rowhammer attacks allow that. That's why our browsers were immediately hardened against Rowhammer attacks because the browser is the way that you have untrusted stuff running in your system which again is why I'm not so sure I'm bullish on the idea of giving our browsers, like releasing our browsers from their sandbox and allowing them to have access to the native file system that they're running on. But we'll see how that goes.

Similarly, if you've got something that is already hammering away at your system's RAM, you've already got malware running in your system, and that's not a place you want to be. So anyway, I wanted to address this because the popular press, the technical press, has talked about how Rowhammer is now functional in ECC memory. And it's like, yes. If the chip is one whose ECC has been reverse engineered, if it knows where it's running, I would argue anyone who, like state-level actors, where a huge amount of resources could be mounted to attack a specific system, a specific thing, where it could be known what the hardware platform is, people could be put on the task of reverse engineering the ECC algorithms for that particular hardware platform.

Then the Rowhammer attack could be designed to flip the bits specifically necessary to achieve a specific goal on that specific platform. Look at all the times I had to use the word "specific." I mean, it's a really specific, really focused attack. Then we have a problem. But for us, for most of us, it's of academic interest to see how something from four years ago keeps coming back, attacks keep getting stronger against all the mitigation that is brought to bear. From a theoretical security standpoint, really interesting. But I don't think it really affects us directly very much.

Leo: Well, I hope those guys got their Ph.D. That's all I can say.

Steve: Well, four years ago, I think that whole team probably graduated. They've got their doctorates, and they're gone. And so Herb has a new team of Ph.D. students. Okay.

Leo: No, let's do it again.

Steve: You want to be doctors? Let's do it again.

Leo: No Nobel Prize for that. But, you know, a Ph.D. is a good start.

Steve: That's right.

Leo: Steve, we've done it again. We've completely dissected and hammered upon the rows of information, of security information from this week. And there's nothing left but pulp.

Steve: Among us and our listeners.

Leo: Yeah. Well done. Bravo. I hope you enjoyed watching the show. Now, some of you watch live. And if you'd like to do that, you're more than welcome. That's the best way, if you want to interact with the show, to get in the chatroom at irc.twit.tv, do it while the show's live. That's Tuesday afternoons, 1:30 Pacific, 4:30 Eastern, 21:30 UTC. You can watch at TWiT.tv/live.

You can also get the downloads. We offer a variety of ways to download it. Steve's got the 64Kb audio at his website, GRC.com, along with transcripts, which are handy if you like to read along as you listen. I know a lot of people do. And, you know, I bet you there are not a few people who have a complete collection of all the audio and a complete collection of all the transcripts so that they can search it, refer back and so forth. That's a very good idea. Put it in binders. Put it on your shelf. You'll impress your friends. GRC.com.

While you're there, pick up a copy of SpinRite, world's best hard drive recovery and maintenance utility. Find out more about SQRL. Everything else is free. Lots of freebies on there including ShieldsUP to test your router, Perfect Paper Passwords, Password Haystacks, everything. It's fun. And even health advice, and Vitamin D information, and it goes on and on. It's a treasure trove: GRC.com. If you want to get a hold of Steve, you can leave a feedback message there at GRC.com/feedback. But you also can DM him at Twitter. He's @SGgrc on the Twitter.

We have audio and video at our website, TWiT.tv/sn. You can download whatever you like there. For most people, the most convenient thing to do is get a podcast app on your phone or your tablet and subscribe. That way you'll get every episode automatically, the minute it's available. It's the fastest way to get the downloads without having to think about it. We should actually make a feed of the show notes, too. I bet you people would appreciate that at some point. But for now the show notes are at GRC.com.

I think that's everything. Thank you, Steve. Have a great weekend.

Steve: And we'll be back in December.

Leo: Holy cow. How did that happen? I will not, actually. I'm going to be in New York City next week for our annual trip to see shows on Broadway.

Steve: Must be Jason time. I do remember that you were going to be...

Leo: It will be Jason time. It'll be Jason time.

Steve: Cool.

Leo: And then I'll be back for the week after that. And we have planned a great Best Of. I don't think we've ever done a Best Of with Security Now!.

Steve: We haven't, but we're going to this year.

Leo: We're going to let you take a couple of weeks off because Security Now! falls on Tuesday, which is Christmas and New Year's.

Steve: Yup.

Leo: So we're just going to play that Best Of twice. And we'll be back doing live shows on the second Tuesday of 2019. That's like Tuesday, December 8th, I mean January 8th.

Steve: Cool.

Leo: Thank you, Steve.

Steve: Thank you, my friend. See you next week.

Leo: Have a great night. See you next time.

Steve: Bye.