Great Firewalls & Cannons

Description: Leo and I catch up with the most interesting and significant security and privacy news of the week. Then we take a close look at what's known of the mechanisms China has developed - both filtering and offensive weaponry - to provide for their censorship needs and to potentially attack external Internet targets.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-504.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-504-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson's here. Yes, as always, the news ripped from the headlines of today's paper including the Great Firewall of China, and now the Great Cannon. How do they work? Steve explains all, next.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 504, recorded Tuesday, April 21st, 2015: Great Firewalls and Cannons.

It's time for Security Now!, the show that protects you and your loved ones online. This guy is the Great Firewall of Gibson. He protects us all. And you know what I love about this show, you also listen and learn. He's a great teacher. Thank you for being here, Steve Gibson.

Steve Gibson: Hey, you know me, this is a pleasure. On the 500th episode I thanked everyone for their being here because they're always thanking me. So, and I know that this is a useful couple of hours that we spend every week.

Leo: Useful, fun, and enlightening.

Steve: Yeah.

Leo: And, by the way, thanks to Mike Elgan for filling in last week.

Steve: Yeah.

Leo: We were down at the, or up at the, or over at the NAB show.

Steve: He did a great job with the Q&A.

Leo: Good.

Steve: And it's funny because, when I don't have you, I'm always a little self-conscious of, like, that we'll be talking about things that you won't get so that - because this podcast, probably more than any other one...

Leo: You cannot miss an episode.

Steve: It builds, yeah.

Leo: Yeah, yeah.

Steve: You know, it really does build on the things we've covered in the past. How many times am I saying, oh, we talked about this three weeks ago, we talked about this four weeks ago, and now this is happening, blah blah blah. So I'm always thinking, oh, Leo won't know about this. Well, that's okay, you know. But you're rare - although, I was going to say, you're rarely gone. But you have, you are talking about trips coming up. So is that in the summertime sometime?

Leo: Yeah, just so you know, I'll be leaving June 27 through July 14th.

Steve: Okay.

Leo: Lisa and I are going on a river cruise down the Rhine...

Steve: Nice.

Leo: ...the Main and the Danube.

Steve: Well, and you need to refresh.

Leo: You've got to take a break. But I've got to say, because this is - you're the same way. It's our passion. I do nothing but read tech news day in, day out, wherever I am. So it's - few are the stories that I haven't been reading about and cognizant of. What frequently happens, though, and I know I'm not alone in this, is I

read a story, and I say, I can't wait to hear what Steve has to say about this.

Steve: Yeah. I'm seeing people tweeting that a lot, too. They'll say, oh, this looks bad, but we'll have to wait and see what Security Now! says. So, because a lot of this stuff does beg some interpretation. We've got a really interesting, sort of a very brief follow-up on TrueCrypt audit that we talked about two weeks ago. That was the topic of our show two weeks ago. Everyone's talking about a nasty IIS bug that was part of the Patch Tuesday, last Tuesday's Patch Tuesday, which is bad because it is just so easy to do. We'll talk about that.

I found an interesting, sort of from a privacy standpoint, new service that Google is offering that I want to talk about. Also some changes in Chrome. A popular library in iOS that has some people worried. News of Let's Encrypt, which is the automated server is able to issue its own SSL certs service, which is moving forward.

And then our big topic for the day, the title of the show, is China's Great Firewall and Great Cannon. There's been, in the wake of the cannon blasting that they were doing of GitHub, like mid-March, now we understand much more about the architecture of the system. We've never really addressed the Great Firewall at all. I understand exactly how it works now. And we did talk about the Great Cannon. But I want to tie that all together because this is a major actor on the Internet, not only choosing to censor the content that gets into China, but now, as we know, actively using other people's machines for cyberwarfare.

And it's just, I mean, I feel like it's science fiction, but it actually happened a few weeks ago. And it was embarrassingly public. China never took any responsibility for it. They sort of deflected the questions. We now have hard evidence that it was a state-sponsored attack, intimately tied to the Great Firewall, which we know is a state-sponsored Internet filter. So lots of new information that I think will be interesting to share with our audience this week.

Leo: Very, very exciting.

Steve: So the picture of the week on the front page of our show notes is a pop-up that I got when I began to experiment with a new feature that Google has made available, that we'll be talking about in a minute. But it was big, so I stuck it up there on the front page, and we'll come back to it.

I wanted to quickly do a little bit of a follow-up to the TrueCrypt audit coverage from two weeks ago because I spotted something on the 'Net from the - Tom, he would be upset if I called him the "lead cryptographer." He was the team lead or the managerial lead. And Tom Ritter is a cryptographer. He was one of the three who were part of the TrueCrypt audit. And I found a quote by Tom saying, "The audits of TrueCrypt get a lot of press because it's something flashy. But the development effort that went into TrueCrypt at the beginning are immense and incredible. And the developers don't get as much credit as they should for producing a disk and volume encryption project for multiple platforms and for maintaining it for a decade or more. There are successor projects, and they are improving it in their own ways. I'm excited to see those projects grow and thrive and last as long as TrueCrypt did. I still use TrueCrypt," says Tom Ritter, who audited it and is a cryptographer.

Leo: Now, that's high praise.

Steve: That was my point. "I still use TrueCrypt," he says, "and want to see it supported in the future." And I have looked at, what is it, VeraCrypt, and can't think of the other one right off the top of my head. They're making a point of being forward compatible. That is, being able to take an existing TrueCrypt volume, presumably, and still be able to interpret it. Although, for example, they're improving some ciphers. No doubt they're keeping the older legacy ones around in order to be able to still function with an existing TrueCrypt container.

But I just thought it was interesting that Tom Ritter, who would know better, says, "I still use TrueCrypt and want to see it supported in the future." So, and for what it's worth, I'm using it because it still solves the problem. And I will be forced away from it when at some point it turns out that it can't handle GPT partition types or something like that, where some compatibility gotcha occurs, just because of TrueCrypt's age.

Leo: And that's fairly soon; right? I mean, in some cases, I think, yeah.

Steve: Yes, yes. Yeah. And so I'm glad that it is being moved forward. And as I said, I'm so glad for the audit because it spotted a few areas, they weren't critical, but might as well fix them in the code that is now alive and fixable. And I guess my point was that I'm always going to feel more comfortable with a third-party solution. I just, you know, using BitLocker from Microsoft is like, you know, they're not going to show us everything that it's doing. And it's hard to trust that there isn't some way that they could honor a demand for a drive to be decrypted. And I only, I mean, the reason for using it is that it really, really is safe. So the industry...

Leo: And where can we get it, by the way? I think it's worthy of a note there. Because you're preserving the - because you can't just go to TrueCrypt.org anymore.

Steve: No. I grabbed the 7.1a, all the archives and resources at that time. So it is at GRC.com. I think if you just put, like, TrueCrypt into Google, I was, for a while at least, GRC was one of the links that came up pretty quickly in a Google search. So, and a lot of people are visiting that page and grabbing the content from me. And I think it might have been, I don't remember if it was - somebody was also maintaining the hashes on a separate site. So, oh, yeah, there it is, GRC, TrueCrypt, the final release archive.

Leo: Now, if you get it from - if you get 7.2 from the TrueCrypt site, don't.

Steve: Ah, see, that's the one that they deliberately neutered. This is why we knew it wasn't something done overnight because 7.2 only removes TrueCrypt. You can't use it to install TrueCrypt. And that's a nontrivial change. They had to go through all of the code and, like, remove all of the aspects of it that were about creating a new volume. Turns out there was a lot there that they had to do. So 7.2, it will interpret a TrueCrypt volume, but it will not create one, and it will allow you to remove it. It'll remove, it'll decrypt a drive for you. So that's the only thing that they're now making available.

Leo: I mean, obviously Tom - is that his name?

Steve: Yeah, Tom Ritter.

Leo: Is not using VeraCrypt. But do you recommend that people go to your site, download the final release of TrueCrypt? Or would you recommend they use VeraCrypt?

Steve: I guess my sense is, I mean, I'm sure VeraCrypt is fine. But TrueCrypt is bulletproof. And my recommendation would be, if it works - but you know me. I'm still sitting - I'm sitting in front of XP, Leo, so I'm really - I'm a bit of a skewed sample. But my sense is, if it works, use that one. It's been really just pounded to death by hundreds of thousands of people, that is, the original TrueCrypt 7.1a. But at some point it won't work for you because your system will no longer be compatible, in which case you'll want to go explore the forks of TrueCrypt. That would be my strategy, I think. There's nothing that they have done that makes theirs superior to what TrueCrypt was because there was just nothing wrong with it the way it was.

Leo: Yeah, yeah. And it has been vetted. And it has been audited.

Steve: Yes,

Leo: Which we can't say for anything else. So there's that, too.

Steve: Yes. And it is so easy to make a mistake. In fact, that is our next story because this bug was not in Windows Server 2003. It apparently got introduced in Windows 7 and the same codebase, which is Windows Server 2008, and 8.8 and 8.1 and Windows Server 2012 and the core installation. Anyway, this is the bug that the security industry has been abuzz about, mainly because it is horribly easy to do.

Microsoft gets in trouble always when they move stuff from the user space into the kernel. How many times have we run across JPGs or image files that are able to take over your computer? It's crazy to have an image file able to execute code on your machine. But they famously moved GDI, the Graphics Device Interface, from user space down into the kernel, at a time when they were desperate for more performance. And the so-called "ring transition," when an application needs to call an operating system service, the application has limited privileges, and it needs to connect to code in the kernel that has infinite privileges. I mean, god. I mean, literally, it can read any address it wants to. It can do anything. The kernel is maximally privileged.

It turns out that the architecture to make that transition in a carefully controlled, secure fashion takes some time. There's lots of caches and buffers that need to get flushed and switched. And so every single time the code in the user space has a request for anything from the operating system, there's some overhead. So when you're desperate for performance, as Microsoft has been at various stages during the history of Windows, one of the things you look at is, ooh, you know, if instead of having code in the user space, which is making lots of calls into the kernel, it's so tempting to move that code itself into

the kernel because then there's no more transitions. You'd still have to talk to that from the user space.

But, for example, the user might say, draw a rectangle at these coordinates. And so that command goes one transition into the kernel. Then the graphics device interface code is able to just go bzzz, you know, and do it much faster. All of the individual colors and pens and corners and roundings and shapings and mergings, all the complicated variations can be done far faster than if it was living with the application in the application space and had to keep begging the kernel to do all these little things for it. So Microsoft did that when they moved GDI in the kernel.

They faced another temptation, which they succumbed to, when they moved a chunk of IIS, the server, into the kernel. And they did it in a module, a kernel driver called http.sys. So it is, basically, they looked at a way that they could cut the server in two so that, with a minimal amount of relocation, that is, moving the smallest amount that made sense into the kernel, they could get the most bang for the buck. And so what they did was they created in Windows 7, which corresponds to Windows Server 2008 and all the subsequent OSes, because this has been there, they allow the kernel to parse the HTTP query enough to see whether the item being requested is in the cache. And they also put the cache in the kernel.

So the performance gain is potentially dramatic because we know that so many websites are fulfilling requests from cached resources, like all the images, the same images that they're serving, not just over and over to the same browser, but to all the different people who come whose browser needs to get at least one copy of the image. Well, if it's already been found in the file system and pulled down into memory, and it's sitting in the kernel, there's all kinds of optimizations you can do. Windows can just sort of point some pointers at it, and it just shoots that asset back to the requester.

Well, it turns out they made a mistake, which is pretty simple, but of course all of these mistakes are simple in retrospect. And on the show notes here I show the piece of code which is incorrect. Basically, one of the request headers that a browser or an HTTP client can include is called "range." If you don't specify a range, then the server assumes you just want the whole thing. But you can also say, for whatever reason, maybe you're actually running four connections in parallel, and each connection is getting a different piece of something large, and you're doing that, you know, that's what the download accelerators did for years. For whatever reason, it's in the HTTP spec. And that allows the browser to say, okay, I want this, but start at byte 702 up through 5,027. You can just arbitrarily say this is the first byte of the object I want; this is the last byte.

Well, if you have a first and last byte, one of the things you need to do is determine how many bytes that is. The way you do that is you subtract the first, the beginning offset of the range, from the ending offset. That gives you the difference. But this is one of the tricks in programming. The difference is not the number because say that you wanted - you said I want to start with byte two, and I want to end with byte three. Well, if you subtract two from three, you get one. Except that the range is inclusive, so you actually need two. So you subtract the smaller, the beginning from the ending, and add one.

And in that code that you've got on the screen now, Leo, from the show notes, that is exactly what they're doing. They have two 64-bit values. And this is 32-bit code because, for example, the EAX register and EDI, those are 32-bit registers. But the range, but 32 bits only gives you 4.3GB, as we know, and things can be bigger than that. And so IIS is treating them as 64-bit sizes. And so that first subtraction is subtracting the low 32-bits of each of the ends of the range. And then that second one, the SBB, that's subtract with borrow. In case there was a borrow from the first, the result of the lower side

subtraction, that then subtracts an extra one from the high side subtraction.

And so once they're done, then you can see they add one to EAX, and then they add with carry a zero to ECX. That's because, if the add of one to EAX does overflow, EAX goes back around to zero, it'll leave the carry bit set. And so when you add with carry a zero to EAX, that essentially adds that one that fell off the high end to ECX. So what they missed, and you can see it right there, no test to see if that add of ECX overflows. They didn't check for overflow.

So it turns out that in practice, simply using a crazy range header, and I've showed here in the show notes, pretty much anything, and then as the starting location, and then this special number, 18446744073709551615, that's the maximum value that a 64-bit quantity, unsigned quantity, can contain. And so I'm sure this code was written in C. And so in C they were subtracting a long long, which was the starting point of the range, from a long long, which is a 64-bit quantity, from the ending point, and adding one. And they didn't check for overflow. And as a consequence, anyone making a request of IIS with that range line will crash the server. It just BSODs.

Microsoft called it a remote code execution exploit. Nobody has figured out yet how to actually make that execute code. There are some people who have figured out how to get some information disclosure. Apparently ranges can also have multiple components. It's crazy. You can say, like, I want from two to five, and then 27 to 46. And so you can create, like, compound ranges. And it turns out that people who've been playing with this have been able to get the server to send them chunks of memory beyond the object that they are requesting. So that is definitely information leakage, and we don't yet know how bad it is. But turns out it's much easier to crash the server than it is to get it to give you additional information. Nobody's figured out how to make it execute. Maybe if you had - I can't even imagine how you could do that from what we know.

Leo: But we've talked about this before. The first step is always a crash, which can be used as a denial of service. Second step is to see if you can get it to jump somewhere and execute some arbitrary code.

Steve: Right.

Leo: And that takes a while. But the fact that you can get it to crash is always a very, very, very bad sign. It's interesting - so we were showing assembly language. It was written in C, most likely.

Steve: Yup.

Leo: It did say, I noticed the subroutine was @32.

Steve: Yes. Actually, that says UI, a pointer to parse range, and then it says @32 [U1pParseRange@32]. That's a notation for the number of bytes of arguments. So whatever this parse range routine...

Leo: The disassembler gave you or something.

Steve: Yes. Yeah. So it's a clue to say, essentially, once you're done, pop 32 bytes off of the stack in order to clean up the stack because the caller pushes the bytes on the stack, and so the routine itself cleans up the stack afterwards. So that doesn't refer to the bitness of it. For example, it wouldn't say 64 if it were 64-bit code. It's how many parameters were required by that subroutine.

Leo: It's interesting. Whoever did the disassembly must have had access to a symbol table because usually you don't get real symbol names, real routine names.

Steve: That's true, although the developer tools do have the debugging versions of the code.

Leo: Oh, okay. There you go.

Steve: And so you are able to figure out what all that stuff is. And the way Windows works with dynamic link libraries, they sometimes need that also. I mean, they link by name. And so the DLLs will have the names of the routines in order for the linker to glue it all together. So there is some of this inefficiency. Netcraft estimates that there are about 70, seven zero, million publicly exposed IIS servers on the Internet. Apache and Nginx have way surpassed IIS, IIS basically sort of losing the war for dominance. Not that it ever had it.

Leo: But you still use IIS; right?

Steve: I do because I'm a Windows programmer, and I know Windows, and I started. If I were doing it, if I were starting it today, I would never consider using IIS. But I have a huge investment in server-side code. And I've gotten to know it, and I can't quite say that I've gotten to love it because Microsoft keeps going this to us. It turns out that I would have been vulnerable except that one of the workarounds is to disable that kernel caching. And I generate so much content dynamically from my server-side code that I never had kernel-caching on. And so I was never vulnerable to this, just, you know, happily. Otherwise I would have immediately patched.

So anyway, this is just, you know, this is a perfect example of a problem that Microsoft would not have had, had they not moved what arguably should really stay up in the user land down into the kernel, purely for the sake of performance. They're under pressure to make IIS competitive, and the architecture that they've got necessitates that they do everything that they can. And putting more of your web server in the kernel, horrible as that is, is a way to achieve that. And the risk is a little tiny mistake like this can really ruin your whole day. And what's happening is people are getting - oh, this is, by the way, when this was released, Microsoft said there are no known attacks. It's been completely reverse engineered, and it is being scanned for actively on the 'Net, and IIS servers are crashing constantly because they have not been patched. And the default is to cache content in the kernel for the sake of performance.

Leo: Although crashing the server usually means just it reboots, restarts, you know.

Steve: Right. Right. And that's why it's called a "denial of service," because you're denying users service while it gets itself going again.

Leo: Right, right.

Steve: Now, this is interesting. Google is allowing us, anyone who has a relationship with Google, to download their search history. Back, I'm not sure how far back it goes. In my case, because I was curious about this, you go to, guess it's Google.com/history. Or the link I have is history.google.com/history. And there's a nice little page that shows some stats about you. And if you click on the little gear icon in the upper right, there's the download option. And you can have Google prepare a ZIP file containing every single Google search you have made, in JSON format, for - in my case it goes back to January of 2012. So I'm looking here, and I just stuck it in the show notes here, so quarterly summaries of every search I've made through 2012, 2013, 2014, and up to now in 2015. They caution you that there's a lot of potentially sensitive personal information here. I mean, this is everything I've searched Google for.

Leo: I don't think this is new.

Steve: This isn't. Okay.

Leo: So Google, well, and I'm sure some tech blog discovered this and said it's new. But Google's always had a site called "Google Takeout," where you can download everything.

Steve: Ah, yes. And that's - okay.

Leo: And you've always been able to do that. And you get to choose what you want to have in here. I don't think search history is new. There's lots more than search history, including your location history. Every once in a while somebody clicks a link - Google's had this for a long time, Google.com/dashboard - and suddenly realizes, oh, my god, they're keeping track of everything. And Google's always offered Google Takeout. They've always been upfront about this. I don't think this is new, unless I'm missing something.

Steve: And I don't know. I mean, I can't...

Leo: Yeah, some tech blog published it. I saw it all of a sudden all over the place. Google does this. It's like, yeah.

Steve: Anyway, from my standpoint...

Leo: It's good to know.

Steve: I just thought it was cool...

Leo: Yeah.

Steve: ...that, I mean, it's sort of a blast through the past when you browse through, like, what you were searching for three and a half years ago. It's like, oh, yeah, I remember that. Actually, I have some tabs that are that old, but that's another story.

Leo: I bet you do. You can clear this. You don't have to let Google save it. Most of us do because it gives us some value in Google Now and other things. And Google tailors its searches to previous searches. But Google's always been very clear about this. At Google.com/dashboard, you can see what they take, what they keep track of. And the one that scares people more often is the location history because, if you have a smartphone with Android on it, they've been keeping track of where you are, you know, unless you turned that off, for a long time. This is all in here. Even my Orkut postings are still here.

Steve: So speaking of Google and Chrome, they are moving forward in their never-ending march to tighten things up. And with Chrome 42, which is out now, they have finally disabled what's called the NPAPI. They gave plenty of ample warning, notice that they were going to do this. The NPAPI is the Netscape Plugin Application Programming Interface that dates back to the '90s. And I think, like, Navigator 2.0 is where this first happened. Essentially, Netscape realized that their browser would be able to render HTML pages and JPGs and GIFs, and I don't think there were PNGs back then, but sort of the standard resources. But there would be other content, for example, a PDF is a perfect example. How could a browser render a PDF in its UI? How could it contain it?

So what you need is you need to create a standardized, so-called API, Application Programming Interface, that other developers can use to talk to the browser guts in order so that they just have to handle, for example, if we extend this PDF notion, they would create a PDF rendering plugin which you would then add to Netscape. You register it with the browser. And then when you request something with a content type of whatever it is, like application/pdf, Netscape would look to see if it has anything registered to handle that, if there's a plugin that says, oh, yeah, I know how to render those. In which case it would hand control to the plugin, and the plugin would then get access to the physical screen real estate in order to render what it wants to.

So this has been - it was very widely supported. Even IE briefly added support, although they dropped it a long time ago, back with v5.5. So IE has not had support for this Netscape Plugin API. However, Safari and Firefox still both do and continue to support it. Chrome, the Chromium people were just, I don't know, their claim, for what it's worth, is that it was causing stability problems. It's interesting that Java and Silverlight are both hosted under this Netscape Plugin API. So if Java is going to continue to be supported in Chrome, it's going to have to probably change.

Chrome has its own API that they call "Pepper." It's PPAPI, which is sort of essentially the same thing. And, for example, that's the API that hosts Chrome's built-in support for

Flash. So the Flash that's in Chrome, it's sitting in this Pepper container using this PPAPI. So if Microsoft or Oracle chose to move their Java and Silverlight over, then they could do that. But as of this version of Chrome, 42, it's now disabled by default. So it's not gone, it won't actually be gone until Chrome 45, which is targeted for September of this year. So there's about another six months. At the moment, you may find that stuff you're using will not function. But you can turn that back on.

Leo: It's in the chrome://flags.

Steve: Yes.

Leo: But I'm curious, who uses NPAPI? I mean, really. Does somebody still use this?

Steve: Well, Java and Silverlight are both still using it.

Leo: Silverlight, right, right, right. Java, who cares?

Steve: Yeah, and there are - yeah, exactly, safer without it.

Leo: Safer without that.

Steve: Yeah.

Leo: And Netflix has moved away from Silverlight. So I don't know...

Steve: Correct. And Netflix was, as you say, the big Silverlight adopter. But they're away from it. Essentially, the effort is to move people towards HTML5 that can now do so much more than HTML4 or 3 were able to do back then, that you really needed natively written add-ons. There just isn't that much demand for it anymore.

Leo: Yeah. It seems - this does not seem premature yet.

Steve: No, and I don't think it is. Chromium has made it very clear they were going to be phasing out the NPAPI, giving people as much time as they needed to switch over, if they still wanted to do a plugin, to use Pepper, which is what Flash uses in order to get the same sort of services from the browser.

Leo: And I'm on Netflix right now in my new NPAPI-less Chrome browser, and it plays just fine. So I'm not too worried about it.

Steve: Yeah. Yeah, I think Netflix is using the WebM stuff, or the...

Leo: Yeah, VP9 maybe, and, yeah.

Steve: Yeah, yeah.

Leo: All right. Well, that's good.

Steve: So for a few months this year a very popular networking library for iOS apps called AFNetworking developed a problem. It was broken in January of this year with v2.5.1 and only fixed three weeks ago in 2.5.2. The problem is it skips certificate verification checks. Whoopsie. So at this point...

Leo: When Steve says "whoopsie..."

Steve: Whoopsie.

Leo: That's the one thing you don't want to hear your doctor say, whoopsie.

Steve: Or staring at the X-rays and going, "Hmm."

Leo: Hmm. Or the two together would be bad.

Steve: Yes, very bad. So there's an outfit called SourceDNA that scans iOS, the iOS App Store, you know, iTunes. And they have identified at least 1500 iOS apps in about two million installations which today remain vulnerable. None of these seem like really mainstream apps, although Citrix OpenVoice is one, Citrix OpenVoice Audio Conferencing; Alibaba.com's mobile app; and then things like Movies by Flixster with Rotten Tomatoes. Something called KYBankAgent, that seems a little spooky.

Leo: That doesn't sound good at all.

Steve: No, it doesn't.

Leo: Whoopsie.

Steve: And the Revo Restaurant Point of Sale app. So these are sort of off-the-beaten-path apps. But for what it's worth, they may be apps that people have, and they need to get updated. There is a search tool. SourceDNA.com created something they call "searchlight." So searchlight.sourcedna.com/lookup is a search tool that allows anyone worried about a specific app to see whether it is vulnerable and whether or not it's been fixed. This is not a huge problem because it's not a problem in iOS. And it's not clear how widespread AFNetworking is.

But if you've got a sort of a smaller custom iOS app that is doing Internet communications, the vulnerability is that it is completely open to man-in-the-middle attacks. An HTTPS proxy could just use a bogus certificate. It doesn't have to be a certificate that it got from CNNIC. It could just, I mean, it could be expired. It could be wrong. And these apps will not detect that. I was a little surprised that iOS won't detect that, either. But apparently it's been verified that it slips right past if the app is using its own, this AFNetworking library. So anyway, just a heads-up for anyone who might be affected by that. It doesn't seem, again, like it's a huge deal.

And our illustrious crypto auditors have moved on from the TrueCrypt audit to auditing the Let's Encrypt system, which is really good news. To recap, Let's Encrypt, which is over at LetsEncrypt.org, that's a forthcoming new certificate authority. They're going to be a certificate authority. I'm sure that until they get into our root stores, which may take a while, they will verify their integrity and get somebody like a GlobalSign to sign their certificate, and so they will be an authorized intermediate CA.

The point is, this is that system that we talked about a few months ago. The EFF is behind it, and it's got a lot of major league industry sponsors. The point is, let's make encryption so simple that everyone can do it. Make it, first of all, simple to configure, and free, so no longer does it cost anything, and automated. So with just two commands to a Linux system, you do an apt-get install lets-encrypt, and then you give the command lets-encrypt and then your domain name. And with those two commands, your server is online with a certificate. So to do that, essentially we need the backend management of the CA, so the server-side stuff for this Let's Encrypt certificate authority, and we need a protocol. The protocol they call ACME, which is short for Automated Certificate Management Environment. And the server-side stuff is named Boulder, which is largely written in the GO language.

And these guys, the NCC guys who audited TrueCrypt, have now been commissioned to audit this technology. So they're going to do a full verification of the code that's been written, the backend server-side stuff, and take a look, a hard look at the protocol. And this is exactly what we would like as a system like this moves forward. And once it's deployed, and they're still targeting for mid-2015, so we're approaching mid-2015 a few months from now, it will mean that anyone who has the Let's Encrypt module in their server - and I'm not sure for things like Nginx. I imagine Nginx will be right there. We know Linux will. It would be great if someone does support for IIS. The idea will be that the server has a protocol with the CA saying, hi there. I am Ajax.org, and I need a certificate.

And so you can imagine how easily this can work. It's one of those things where you just kind of want to slap your head because the server claims to be Ajax, the server for Ajax.org, to the CA. The CA probably says, okay, here's a blob. Go stick this on your root, and let me know when it's there. And so over the protocol the Ajax server receives the blob, puts it in the root so that it's publicly accessible, and then responds back to the CA, okay, blob's there. And then the CA goes and tries to fetch it, you know, through the public Internet, gets the blob, verifies it, and says, okay, you must be the server for Ajax.org because you've just proven that. So here's your certificate. And the certificate goes back through the protocol, and the server says thank you very much and binds it to the IP and port, and you now have security for free.

And this thing then is an agent running in the background, making sure that the certificate never expires. As expiration date approaches, maybe a month or two before, it says, "Hey, me again. Let's do this dance again and update me." And, you know, bing bing bing, and, okay, now we've got another however many years the certificate lives for. I mean, it could be - it's so easy, it doesn't have to be a long-lived certificate. And if

anything happens where you lose control, you're able to use the same system for revocation. So revoke that certificate. Let me have a new one. Sorry about that. And bing bing bing, now the server's got its certificate again.

Leo: Bing bing.

Steve: Bing bing bing.

Leo: Can I ask a question? Because of course we want to, I want to, DigiCert gave us a certificate, and we would love to use HTTPS on TWiT. And I've been told that we can't or shouldn't because it breaks our caching. So, you know, as many high-traffic sites do, we cache heavily. We're using Varnish right now. The new system is going to use Redis and Varnish. It's going to be a fairly sophisticated caching system. And I was told that, well, we don't want to do that because then we can't cache it.

Steve: So where are the caches located? Are they, like...

Leo: In other places.

Steve: Yeah, that's the problem.

Leo: Redis is Redis Labs. It's not TWiT.tv. So I imagine we're not alone in this regard.

Steve: No. And in fact, a perfect example is we've talked about how ISPs are unhappy about the move to security because they're depending upon caching. For example, a big ISP like Cox has all of its customers are going to Google. Well, only one copy of Google's front page decoration and resources needs to get pulled by one customer over a non-secure connection in order for the ISP to cache it locally. Then it is so much more efficient for the ISP to serve those same assets to all, you know, a big IP like Cox, to all of its customers who are behind that cache. And what that does is it dramatically reduces Cox's bandwidth that it needs because it's able to serve all of those essentially redundant queries for the same content from its own local store, which costs it nothing because it's its own network. And that way it reduces its costs dramatically. And so you're saying the same thing, Leo. Obviously, the reason you want to cache is you don't want to have to be serving all of this from your own server over a more constricted pipeline.

Leo: And, you know, this is the business of Redis. Redis Labs is one of many providers for Redis caching.

Steve: Can you move these to a different domain, like to a cache.twit.tv, and have them manage a subdomain of yours?

Leo: Hmm.

Steve: Because I wonder, I mean, they're going to have to solve this problem one way or the other because the world is...

Leo: CloudFlare, CloudFlare's a really good example of a company that does this for people all the time.

Steve: Right.

Leo: Apparently, somebody in the chatroom, Julio, says a reverse proxy will solve part of the problem. And Cloudflare is developing a solution to encrypt, not only client-to-reverse proxy, but reverse proxy-to-host. I'm sure this will be solved. I mean, there's a lot of businesses like Cloudflare that are - that's their business. And obviously Google is forcing us to do HTTPS.

Steve: Yup.

Leo: It's a little annoying.

Steve: It is. Well, and this is, you know, these changes are painful, yeah.

Leo: Is it really making everything better?

Steve: Um...

Leo: Well, if you go to HTTPS right now, https://TWiT.tv, you're going to get an error because TWiT.tv is not secure, is not a secure server.

Steve: Right.

Leo: We don't have a certificate. And so thank you, Google, you're scaring everybody with this: Go back to safety. Attackers might try to - no. That's not the case, Google.

Steve: Yeah.

Leo: Right? This is Chrome doing this. It isn't that it's - it's not - we don't use HTTPS. But this is the beginning of a war Google's going to launch on us and other

sites who I think you could make an argument don't really need to do HTTPS. Anyway, thank you, Google.

Steve: Yeah. Yeah, I think that the demand for security will come from users. For example, there was another blurb that I didn't pick up on, just because we had so much to talk about this week, which was that AT&T Broadband, you probably saw this, is saying that they will, for a fee of $29 a month, they will not watch all of their users' transactions.

Leo: Well, that's to make up for the loss of revenue; right?

Steve: Exactly. Exactly. The revenue that they get from watching their users' actions and injecting ads and digging into the content. The problem is, what this is saying is that, okay, first of all, they can only do this on non-HTTPS connections. That is, they can only do this on HTTP. So this puts tremendous pressure on this ecosystem to somehow penetrate privacy which our HTTPS gives us. And what I mean by that is, just watch. Mark my words. Here we are on the Security Now! podcast. At some point, AT&T customers are going to be asked to put a certificate, an AT&T certificate, in their browser.

Leo: Right, right. Because then they can peer inside HTTPS.

Steve: Yes, yes. And, boy, there's going to be fireworks when that happens because there are a lot of people who are going to say, wait a minute, I don't want my security cracked by AT&T. So, yikes.

Leo: This is like Superfish. It's the same exact thing.

Steve: Yes, exactly.

Leo: So am I, because if - we aren't HTTPS at TWiT. Now that, you know, no account, you don't create an account. It's a read-only website from your point of view as a user. Does that mean that we are potentially manipulable, a Great Cannon or, I mean, what is - why is it necessary for a read-only website to be HTTPS?

Steve: The only thing, the only vulnerability I could see is that, if someone wanted to inject malicious content onto your pages, they could do that. So TWiT content would be going in the clear all over the 'Net, and it's just, you know, a page is just a bunch of text. And so somebody could easily change the ads or add content of any sort that they wanted to, if somebody wanted to.

Leo: How would they do that?

Steve: It's actually pretty trivial.

Leo: Do they have to compromise the end user's system?

Steve: Well, they do have to be in the middle somehow. So I just saw a tool the other day that was demonstrating a man-in-the-middle attack in an open WiFi environment where it was just ARP spoofing. I mean, you actually can redirect the traffic of people in an open WiFi environment through your machine and then mess with their traffic. So, I mean, I guess the problem is that the world is going to encryption. But at some point there may be so much pressure put on not having encryption that we start being forced to have certificates in our browsers to allow people to proxy and cache. So, I mean, it's just not clear how this is all going to evolve. But, boy, I mean, there are a lot of people who will not be happy if their ISP starts saying, sorry, if you're going to use us, and by the way, you don't have a choice, you're going to have to allow us to look into your traffic.

Leo: Well, I'm hoping that somebody will solve this for us.

Steve: So do you have a different domain for these cacheable things? Or is all your traffic running through this other company so that there...

Leo: No, it all goes through TWiT. So I don't know. I'm not exactly sure what the architecture of it is. And it isn't turned on yet. Right now, if you go to - well, you've seen people have image servers elsewhere, where people use Amazon Web Service...

Steve: Or, for example, a CDN. But, for example, when you have a CDN, it'll often say cdn.twit.tv. And that subdomain is somewhere else.

Leo: In this case, no. If you download a Security Now!, you're going to go to - it'll be a link to Cachefly.

Steve: Right.

Leo: And so Cachefly, which I think is secure - I don't know, anyway. But anyway, Cachefly will serve that content to you. So you only, when you go to TWiT.tv, you get text and images in a read-only fashion.

Steve: And the media comes from big media services.

Leo: Right now the media all comes from our server. But eventually nothing will - well, it's still going to all come from our server, but it will be cached. I guess Redis will be TWiT.tv. It'll look, I mean, it won't look any different to you. I don't know how it works. It's too modern for me.

Steve: Yeah, so in order to answer the question we'd have to really take a look at the architecture.

Leo: Their architecture, yeah.

Steve: Yeah, and see. But I'm sure the problem can be solved. And, frankly, maybe now is the time for you to think about it. Rather than not doing it secure and then having to change it later, like address it now. Say, hey, look, the world is going to HTTPS. Let's not...

Leo: Yeah, that's easy to say. But if it's $100,000 to do it, which it will be, I'm messed up.

Steve: I don't think it should cost anything.

Leo: No, no, no, no, everything costs something. These are developers. I have to get them to create some, you know...

Steve: Okay, work harder.

Leo: Nothing's free.

Steve: Well, you should tell them, hey, wait a minute. This is the way it's supposed to be. Why are you charging me for doing it right?

Leo: Yeah? I'm screwed. Thank you, Google.

Steve: So a little miscellaneous note that I just caught, and that is that Twitter, as of yesterday, now allows optional DMs from anyone.

Leo: I ain't doing it.

Steve: I am.

Leo: You're going to turn that on?

Steve: Because - I did. Well, because I have a lot of dialogue. And the way I'm using Twitter is sort of as a forum. And I am a little self-conscious when there's, like, sort of off-topic dialogue that's going back and forth, and everybody is, like, seeing this nonsense. The problem is, when I was using it much more as a one-to-one communications, I kept getting complaints from people saying, "Hey, I can't DM you

because you're not following me." And they were typically following me, so I was able to DM them. And anyway, I think for me this is great because this allows someone to send something to me.

Leo: Privately, yeah.

Steve: Privately, that's not for everyone. And typically they will be followers of mine, so I'm then able to respond, and I'm just not junking up my feed with, like, random conversation that's not of greater general interest. So anyway, I just wanted to let everyone know that that just became available. You go to, under Security and Privacy, there's a Privacy section at the bottom of it. It says Direct Messages, and the little checkbox is "Receive direct messages from anyone." And mine's on. I can always turn it off if it turns out to be a problem. But I just don't see it being a problem.

Leo: For you it makes perfect sense because you're in the security field. You would want private communications from people you don't know already.

Steve: Right.

Leo: Of course they could email you, but you don't use email. I'll just let people email me because I just don't want a whole bunch of DMs from strangers. But, yeah, I mean, it makes sense for you, though, actually. So it's nice that it's an option, I guess.

Steve: I like it. And so the concern is that - because, I mean, a DM as opposed to an @ mention. I mean, that's the way everyone's communicating now is they just do @SGgrc.

Leo: Right, and it's public.

Steve: And they know, yeah, and it's public, and it'll be in the feed. So people are - so what you're saying is you don't want people knowing that they're able to communicate with you privately through Twitter.

Leo: That's what email's for.

Steve: Right. Okay. Yeah, makes sense.

Leo: Yeah. Twitter is such a cesspool for me that I don't - those are exactly the people I don't want to talk to me.

Steve: So a little bit of SQRL news. I had coffee on Friday morning with Stina Ehrensvrd of Yubico, just to sort of catch up. She was down here doing some corporate business. And whenever she comes down, she says, "Hey, Steve. I'm in town. Let's meet at your

Starbucks." So we did that. And without, like, getting anyone overly excited, because it's premature, I'll just say that Yubico remains very interested in coming up with a Yubico solution that works with SQRL. They have the technical ability to do that now with the way their technology has evolved.

I'm going to fly up and meet with the Yubico techies as soon as the client is nailed down. We're just - just in the last week we've been going back and forth on our last few issues about, like, we've removed a bunch of features and have simplified things. Then we went a little too far, so we've come back. And but it's feeling really good now. Stina also essentially just wants to solve the problem, no matter who wins. And of course she's bullish on U2F, the Universal 2 Factor, or Universal 2nd Factor, which is the branch of FIDO that she's been involved in and which, for example, Google is looking at and that we've heard is going to be part of Windows 10 for Microsoft.

And so anyway, so we just had a nice conversation. One of the things that she reminded me, you know, I was talking to her about how one of the advantages of SQRL is that it was not federated authentication, that is, it was just between the user and the site they were visiting. And I also reminded her that one of the things that FIDO wasn't and actually could not be was secure single-factor authentication, that is, FIDO cannot identify you. FIDO can only confirm your identity. You have to first tell it who you are so that the server can send you back a packet which you've asked it to hold for you. And then you authenticate yourself against that. So it's more involved. Whereas SQRL of course is your identity, and you're able to assert your identity with SQRL.

Anyway, she was saying that not having federated authentication would be a problem because there were instances where you wanted a real-world identity. That is, and we've talked, Leo, on the podcast about how maybe when there's some sort of an ID card or an identity, you'd go to the post office in order to be matched up with your ID, the idea being that the post office is a government facility. Or maybe it's the DMV, or maybe it's the TSA people who already run the PRE program would do that. And somehow, if we get to a point where there is an electronic identity which does need to be connected to our real-world identity, we need a mechanism for that.

And as I was driving - I dropped her off at the airport as she was heading back north. And as I was driving home, I realized, oh, yeah, SQRL could do that. It's not a problem at all. You could be, for example, at an IRS website, which is where you want to prove who you really are, or maybe at some future voting website, I mean, who knows, something where your real-world identity, you want to be able to assert an actual identity rather than an anonymous one. So it turns out it's very simple for that site to present you with the SQRL code for another site which knows your real-world identity, where you have, beforehand, you've matched it up. And you simply authenticate with that SQRL code, and your real-world identity is able to be associated with you and your current web session.

So anyway, I hadn't really thought about it. There was a case where we were trying to prevent fraud because there was a way that a man-in-the-middle attack might be able to leverage that, which we have prevention for. But then I realized the flipside is you could use it in a good way to provide, under the user's control, that kind of real-world authentication, too. So one more checkmark for SQRL.

I did, I just thought I would mention, I meant to have - somewhere there was a fabulous picture that someone sent me of SpinRite running on their drive with just a scattering of green R's, meaning that it had recovered the data in that location.

Leo: Nice, nice, nice.

Steve: I mean, it was just, oh, I just loved even to see it, to think that here was a system that had major damage just across the surface. And it was just a beautiful scattering of green R's saying, yup, we worked on that sector and recovered it perfectly. Your data is fine. Anyway, I couldn't find that picture, but I have a nice tweet from the 16th of April, so a few days ago, from an Andre Couture, who just sent me a nice note through Twitter. He said: "@SGgrc SpinRite just saved my wife's school computer. Your latest podcast just clarified some of those numbers I saw."

And in fact he was talking about last week with Mike I showed a screenshot of the S.M.A.R.T. system, SpinRite's S.M.A.R.T. screen, where it was very clear that this drive was in trouble, and I talked about what those things meant a little bit. So Andre was saying, hey, you know, now I understand that better. And, by the way, SpinRite just saved my wife's computer at school. So thanks for sharing that, Andre.

Leo: Very nice. All right. Let's talk Great Firewalls and Cannons. So let's talk about cannons and firewalls.

Steve: So, China. China, of course, wants to control what its citizens are exposed to. Certainly the so-called Great Firewall has been around long enough that it's just sort of in the ether. People know that there is censorship of the Internet when Chinese citizens try to get content from outside. We've never talked about it specifically, or how it works. And it turns out that the technology to implement the Great Firewall is pretty well understood.

If we flash back to previous podcasts where we've talked about the way the TCP protocol operates, TCP is the connection-oriented protocol, that is, where there's a - some packets go back and forth between the endpoints, agreeing on a bunch of the parameters that they're going to each use in communicating. The so-called SYN, S-Y-N packet, short for "synchronized," is the way for the originating end of the two connecting points to say, I'm going to start numbering my packets from X. And the sequence numbers in TCP are 32 bits, so four bytes, so 4.3 billion packets before that sequence number wraps around. And there are reasons for the sender, security reasons, for the sender to want to choose a sequence number which nobody else on the Internet can guess because there have been lots of attacks over the past where you spoof this sort of initial handshake.

So the initiator sends a SYN packet to the server, that acknowledges the receipt of the SYN and sends its own SYN back. So that's done in a single packet called a SYN-ACK. And then the initiator acknowledges the receipt of the server's SYN, sending back an ACK, an acknowledgment. So that both ends sort of have the numbering scheme that the other one will be using moving forward. And the other thing this does is it verifies that roundtrips are possible, that is, that the Internet is just a whole bunch of routers sort of loosely confederated, glued together, that sort of just function to move packets in the direction of their destination, but without any guarantee.

So one of the nice things about this is by having each end acknowledging the receipt of the other one's synchronized or SYN packet, they both verify that a complete roundtrip path through the Internet exists. So at that point the ends are able to asynchronously just sort of start sending data to each other. And in the case of a web browser, it'll be sending requests for content from the server, that will be responding.

Now, at some point, either end can decide they're done. They can decide that they're finished with their sending. And so they make that statement by sending a so-called FIN, F-I-N, short for finish, saying, okay, I'm finished. And the other side will acknowledge that and often send its own FIN. So then you have a FIN-ACK packet, which is both saying, yeah, I'm done, too, and I acknowledge yours. And then, finally, the one who first sent the FIN packet and received the FIN-ACK will send its ACK. And that's called a "graceful shutdown," where both ends start up, they agree on numbering, they each send whatever data that they want to for as long as they want to, and then they say, okay, I'm all done, I'm finished. And so they sort of tear down, as it's called, they tear down that connection with agreement.

But there is another type of solution or another way that a connection can be taken down, which is known as an "abortive shutdown." And that uses a so-called "TCP reset" packet. If any TCP endpoint that has established a connection, it's done the whole SYN, SYN-ACK, ACK dance, and they're sending data back and forth, if they receive a reset packet, a TCP reset packet, what that says is the other end is done. Don't know what happened. Don't know why. But it's just, basically, it's just hang up. Just shut up. No more communications. This connection is reset.

Now, the sequence numbers are crucial for sort of each end knowing where the other one is in their conversation. As they're sending data back and forth, each end is acknowledging the receipt of a set of packets, one or more packets, saying, okay, I've now received up to this number from where we began. And if a reset packet occurs, it has to fit within what makes sense for the other end to have sent. That is, this is a further guarantee against abuse because otherwise it would be possible for people just to spray the Internet with reset packets, causing connections to be reset. It's not quite as easy as that because the other thing that identifies the endpoints are the IP addresses and the port numbers.

So part of what the whole SYN and SYN-ACK and ACK process establishes is the - they're called "tuples" because it's sort of a multidimensional connection. It's the IP address and the port and the sequence numbering, all which have to match. So when a reset packet comes in, it has to be from the proper IP, from the proper port number - and that's a 16-bit value, that's one of 64,000 or, you know, 65,536 ports - and it has to be sent to the proper IP and to the proper port. And the sequence number has to be within a reasonable-seeming window, that is, so there's tight criteria on what allows a reset packet to be accepted as a reset.

Okay. With that little bit of primer on TCP, all that we've covered in past podcasts, and anybody who's interested can, if you just go to GRC.com/sn, there's a search box on the page, and you can put in "TCP protocol" or "TCP reset" or something, quickly find those past podcasts where we went through all of this in much more detail.

So now we have China, that somehow wants to prevent its citizens from seeing content that it objects to, so-called "banned" content. What China has established inside of the infrastructure of its major bandwidth suppliers, there are four major country, you know, state-scale ISPs operating in China that terminate all of the incoming and outgoing Internet bandwidth for the country, scattered geographically around. Inside of the infrastructure, the datacenters of all of those ISPs that have outside China connections is some equipment which implements the so-called Great Firewall. Every single packet that flows in is examined for keyword matches for anything that doesn't - that is banned.

Now, this is not - and it's important to understand this because the Cannon works differently. This is the Firewall. Think of this as a monitor, that is, it's not a man in the middle. It hasn't been involved in the connection at all. It's passively sniffing. So it's just

looking at the traffic, looking at the packets that go by. And there is certainly some sophisticated high-speed string matching going on because there'll be all kinds of keywords or phrases that China has decided, China's government has decided should not be allowed to proceed in. And they want to block it. But mostly they don't want to block things. So this is sniffing the packets that are going by for anything, any string match that China feels is banned content.

When something like that is found, a packet is found that contains a banned content string, the packet itself has all the information needed to satisfy both ends if they were to receive a reset. That packet has the source IP, the source port, the destination IP, the destination report, and the current sequence numbers for each direction of travel. So some fast-acting hardware that is the Great Firewall, which has been passively sniffing that flow of traffic, sees a match, grabs the headers for that packet, and immediately synthesizes reset packets for each end and sends them out.

So what the recipient sees is a packet, followed immediately by a reset, which is a legitimate reset for its connection because everything matches - source IP, destination IP, source port, destination port, and the sequence numbers. And so what the person inside China or the browser of the person inside China experiences is a legitimate connection reset that appears to have originated from the source of this material and will absolutely be obeyed. It just aborts that query. And similarly, a reset is sent upstream to the originator of that banned content. And once again, the source and destination IPs are reversed from the packet that went toward the China direction. The source and destination ports are reversed because this one is coming in the other direction. So what is the source in one direction is the destination in the other. And similarly, the sequence numbers are swapped.

So what the sender of that packet containing banned content receives is an absolutely authentic, perfect-looking TCP reset that appears, I mean, there's no way to tell it's not legitimate, appears exactly as it should have come from the person it was sending content to. So it resets it. I mean, it summarily drops the connection, just erases all knowledge of it, stops sending anything. The connection is terminated. No additional packets go out. That is the Chinese Firewall.

So it's not a man in the middle. You could sort of think of it as a man on the side. There's someone monitoring the traffic that just - and normally just letting it pass by. But on a string match for something banned, that one packet containing that content is all the information necessary for that firewall hardware or system, whatever it is, to immediately emit reset packets and terminate any further flow of that content. And it's effective. Essentially, it means that nobody on the inside can ever get the stuff that they were looking for if the packet contains a match. It just doesn't happen. The connections just get dropped and you're, you know, game over. So that's the Firewall.

Last month, as we discussed at the time, a few podcasts ago in the middle of March 2015, something unprecedented occurred. And again, as we discussed, two pages, two sites within GitHub went under sustained denial of service attack. And this wasn't, like we were talking before, like the denial of service of crashing a server. This was the classic absolute bandwidth overload.

And it was difficult to filter because this also wasn't the old-school SYN flood, where you'd just send a bunch of these SYN packets, these synchronizing packets, to a server, causing it to initiate, you know, believing them all because any one of them could be valid, having the server initiate all of these connections and then try to honor a flood of connection requests. Instead, these were actual queries being generated by real users' browsers all over the world. So there wasn't - it's not like you could block off China

because the attack wasn't coming from China. It was coming from browsers all over the world that were essentially doing China's bidding for this attack.

So what was surprising about this, what was unprecedented, was that this was a very public attack. That is, two pages on GitHub, which were offering software for circumventing Chinese censorship, were being attacked. This wasn't the first time China had demonstrated their unhappiness with GitHub. Earlier, the entire GitHub domain was blacklisted through DNS so that no GitHub was available inside China. Well, it turns out that that lasted only two days because of the massive outcry from Chinese developers who were depending upon GitHub as the incredible resource that it is. So after two days of just blacklisting GitHub completely, China backed off and made it again public. Clearly, though, they were still not happy with the fact that there were these two pages of software designed to help people circumvent the censorship.

So the Great Cannon was presumably designed sometime in the past. We don't know the history of it. All we know is that we saw, to devastating effect, its first public use in apparently a state-sponsored, virtually a state-sponsored cyber attack is what this was. So China is unable to shut down all of GitHub because China is making such great use of it. But they decided they would just attack two pages. Maybe, we don't really understand fully the logic, maybe to run up the bandwidth costs, the bills. Maybe to just demonstrate that they can do this. It was extremely difficult to block because the way this Cannon worked was to enlist innocent browsers all over the world.

So during the attack, people were probing the operation from outside. Security researchers were fascinated to understand exactly what this thing was and what it was doing. What they learned is that the Cannon is a man-in-the-middle attack. It is intercepting communications. It is standing in, like a proxy, for connections, and looking at the content which is passing by. It is based on the destination IP of the content, that is, where this stuff is going. It can decide to intercept the reply, essentially breaking that query and intercepting the query and generating its own response. So this is in the same location, and we'll talk about how we know that in a second, and appears to be closely related to the Great Firewall software.

In this case, though, some observers have noted that, where this attack was intercepting queries bound for some destination IP, it's very likely that China also has the ability to intercept queries from some specific source IP, meaning that at China's decision, and we don't know that this isn't even going on now because we wouldn't tend to see this in the same way that we saw this massive outbound attack. It might be that China has the ability, with the Great Cannon, to selectively target source IPs so that, if you ask for any - "you" meaning the targeted IP - asks for any resource that comes from somewhere within China, that this Cannon will intercept the response and can replace it with anything they wish, any kind of, for example, malicious content. So the architecture, from what's been observed, seems to be able to match, not only on the destination of a query, but also on its source.

So we have the Cannon, which is functioning like a proxy, looking at queries and selectively blocking them from proceeding and answering them itself. The people who were probing this during the attack used very careful packet TTL. There's another field in all IP packets called TTL, which is short for Time to Live. The idea is that because the Internet is, as I was saying, just a loose federation of routers, as packets flow from one router to the next, being essentially advanced toward their destination, this little field, it's an eight-bit field, so it can have a value up to 255, and it's known as the Time to Live. Every router that receives a packet that needs to then be sent on decrements the TTL field, this Time to Live field, in the packet that it receives. If it goes to zero, if the receiving router receives the TTL set to one and decrements it to zero, that packet will

not be forwarded to its destination. Instead, the router will send back to the packet's originator, because the packet, remember, contains a source IP, so the router will send an expired message back to the originating IP saying, don't know what happened, but this packet has expired. It's apparently been bouncing around the Internet too long.

So this was an inspired piece of design from the very first guys that created this packet-switching network. They realized they had to have some way for packets to die. They had to age off of the Internet, or there might be a possibility, for example, of having something called "routing loops," where Router A sends the packet to Router B, thinking that it's going towards its destination, and then Router B sends it maybe back to A, in the worst case, but maybe onward to C. And then C, again, through some router misconfiguration, and this does happen, C sends it on to A. So now A will send it to B, as it did before, and B will send it to C. So this packet is just going to go around in a circle. So we can't have it go forever or the Internet would just collapse.

So all packets die. And this is - it's crucial that they have to be able to. So it turns out we can use this, and this is the so-called "traceroute" capability, to trace the route of a packet. You deliberately set the packet's lifetime to one. And you put it on the Internet, just like you would if it actually had a chance to get anywhere. And the first router to receive it will decrement that time to live, which it received as one, will decrement it to zero - I'm afraid I'm going to have to...

Leo: Clear your throat, my friend, and we will put a little TWiT bug up that says, I don't know what.

Steve: A little cup of coffee, a little sip of coffee. Anyway, so that first router decrements the TTL to zero and says, oh, this expired, and sends that message back to you. Well, now you know the IP address of the first hop router, as it's called. Then you set the TTL to two, and otherwise send the same packet again. Now the first router gets it, decrements the two to one, but that's fine. It sends it on, on to the next router, that gets the one and decrements it to zero. Now it expires. But it expired at the second router. So the IP address of the message coming back to you saying this expired tells you the IP address of the second router. And so on.

So by using a slowly incrementing TTL, time to live, of the packets that you put onto the Internet, and looking at the messages you get back, you can trace the route that the packet is taking over the 'Net. And if you are clever, you can do the same thing, not, for example, just with a ping packet, but you could do that with TCP. All of the packets which are routed by the IP protocol on the Internet have a time to live, whether that's an ICMP, a UDP, a TCP, whatever protocol you're using, it will have the TTL. So these guys who were probing this Chinese boundary that was acting so strangely a couple weeks ago verified that the path was the same, and the distance was the same, that is, the same number of hops as the Great Wall had and the Great Cannon had. Meaning that they were at the same router.

And this was done from all the different entry points of China, many different ISPs, many different facilities within the same ISP. And in every single instance - sometimes it was 17 hops but not 18, sometimes it was 14 hops and not 15 - the packets got the same distance and hit this Chinese interception system behind the exact same router.

And one other thing was noticed, and that is the packets that the Chinese system generates, the packets that the Great Cannon generates when it's sending back its essentially spoofed reply, and the TCP reset packets that the Great Firewall generate,

there's something a little strange about their own TTLs which is unusual, but they are identically unusual. Which leads the people that have analyzed this to conclude, if nothing else, that they are based on a common software foundation such that the packets they're generating are both unusual in the same unusual way, and turns out to be synchronized.

So everything leads us to believe that there is now an offensive cyber attack tool that is available to China, that they can deploy at their whim. It looks like it's able to spoof a response from any asset inside China to anyone who asks or any specific set of individuals who asks. It's also able to spoof that content in a way that can cause browsers which are asking for content to behave as denial of service reflectors, essentially receiving some JavaScript which causes those browsers to then launch an attack targeted at wherever the people aiming this Great Cannon choose to point it.

And it's worth noting that all of this only works if there is no security. That is, all of this only works over HTTP. So HTTPS avoids all of this interception. None of this technology works over HTTPS. Which, again, makes me wonder how long we're going to be allowed to have secure connections one way or another. It might be that China just says, no, we're going to SSL proxy everything in order to crack open this security, and we are not going to allow VPNs to tunnel through. We're just not going to allow any means to get to us that we're not able to filter. And we see the same pressure from a legislative angle here in the U.S. It's unfortunate that privacy, Internet privacy is under attack, pretty much from every direction.

Leo: It's hopeless. I give up. I'm going home.

Steve: It's all technology. The technology works. And unfortunately, in some cases, for some people, it works too well.

Leo: Yeah.

Steve: Because it is the case that China is blind over TLS connections, over HTTPS connections. They cannot see into them. And so it'll be interesting to see how long they allow that to happen.

Leo: It's always been my thought, and I think others have agreed, that the Chinese government understands that anybody with some sophistication is going to get around what they're doing.

Steve: Right.

Leo: And they are okay with that. They don't mind if the elites can see what they want.

Steve: Okay.

Leo: They don't try too hard to prohibit that, as they might in other countries. They're more worried about the rank and file.

Steve: The 99%, not the 1%.

Leo: Yeah. So it seems like there's always been a little of a laissez-faire point of view from the Chinese government that people get around this using proxies and VPNs or HTTPS and so forth. Although I've got to say they're getting more aggressive. And maybe that's not still the case. It depends on who is in power.

Steve: Yeah. And I guess one of the things I wanted to convey is that this is not simple technology. I mean, somebody - this was in place before someone decided to attack these two pages.

Leo: Right. Interesting.

Steve: So this Great Cannon, I mean, this is cyberwarfare technology. This is a massive flood from innocent users all over the world. So, I mean, as I'm thinking about this, I'm thinking, you know, I could probably do without ever resolving a domain that's in China. I just don't think I ever need to do that. And it's sort of tempting to just say, okay, fine. I'd rather draw, you know, consider that a big black hole that none of my assets ever needs to pull a query from.

Leo: In this case, GitHub is probably the Alderaan for the Giant Cannon Death Star; right?

Steve: Yeah.

Leo: Let's give them a demonstration.

Steve: Really, I found it fascinating that they tried to block it completely, and their own people said, oh, my god, we have to have it. And I'm sure that China saw it, understood that it was too useful for their own developers not to have access to this incredible open source resource.

Leo: Wow. Steve, as always, a great education. And we've got Matt and Jeremy in here, who say this is the best show on the network. That's why they're here. They say, "The heck with those other shows. We are here to hear Steve." We do Security Now! every week at this time. Oh, hey, a couple things I want to tell you about before we go. First of all, we did mention that we are doing The New Screen Savers. It's returning, and I hope you will be, as you were on the old Screen Savers, a regular.

Steve: Yeah.

Leo: We won't be talking about the Click of Death.

Steve: No.

Leo: Something - we're updating it; right?

Steve: The Cannon of Death.

Leo: The Cannon of Death. And this is all part of our 10th Anniversary celebration. TWiT celebrated its 10th year this weekend. And we made a special T-shirt, and I want to make sure everybody knows about this. At Teespring.com/twit, that's where we sell our stuff. But this, as always, is a limited edition, so don't waste too much time getting it. We have both men's and women's styles and a variety of sizes.

Steve: Oh, you mean don't delay getting it.

Leo: Yeah.

Steve: Because when they're gone, they're gone.

Leo: They're gone, they're gone, baby. It's got a nice front piece of the No. 10 with the TWiT logo inside the zero. And then on the back it's got our official 10th Anniversary badge. It's a nice T-shirt that will be something very nice to have 10 years from now, and 20 and a hundred years from now. "I was there the first 10 years." I also want to tell people, you know, we're looking for callers for The Screen Savers, The New Screen Savers. Because just like the old Screen Savers, we're going to take calls on the TWiT Netcam Network.

Steve: Oh, cool. Very nice.

Leo: You know, they wanted to call it the Webcam Network when we were doing Tech TV. And they said, well, that's technically incorrect. It isn't a web netcam, it's a netcam. And so that's why it was called the Netcam Network.

Steve: So you'll have the technology to solicit calls, like an 800 number where people can call?

Leo: No, no, no, we'll do Skype. We'll probably do Skype so we can see you.

Steve: Oh, cool, perfect, perfect, of course.

Leo: If you have a question you want to ask - and sometimes, by the way, we'll call, you know, you can ask a question about, say, the Great Cannon, and we'll call Steve in on Skype and have him answer it; right?

Steve: Nice.

Leo: You don't have to always be in-studio. Screensavers@twit.tv is the email address. Squeen - squeensavers at TWiT, ha ha ha ha, TWiT.tv. And we'll choose the questions and line you up. You'll need to be available Saturday afternoon, 3:00 p.m. Pacific. That's when we record, 6:00 p.m. Eastern, 2200 UTC.

Steve: You think it'll be about a two-hour show?

Leo: One hour. I'm hope - well.

Steve: [Laughs]

Leo: This used to be a 20-minute show.

Steve: That's right. Just a little quick update.

Leo: Yeah. We've got the same basic rundown as the original Screen Savers, which was 44 minutes. I think we can keep it around an hour.

Steve: Uh-huh.

Leo: We'll see. So do email screensavers@twit.tv with your Skype handle. By the way, we want audio and video, so you need to have a camera. An email address so we can get back to you. A phone number for backup purposes only. We're not going to sell it or use it in any way, don't worry. Or the email. And then we'd like your name and location, obviously. And of course your question. Again...

Steve: So it will be a call-in show.

Leo: Yeah. I mean, not entirely. It'll be very much like The Screen Savers.

Steve: Yeah.

Leo: There'll be segments. It's a variety show because there's lots of stuff that we would like, like you doing coffee, that we'd like to do, but we couldn't do every...

Steve: There's no real place for it, yeah.

Leo: Yeah. So this is the place. This will be all sorts of stuff, like The Screen Savers was.

Steve: Nice.

Leo: Screensavers@twit.tv. And don't forget Teespring.com/twit. And this show, my friends, I know you will not forget, is always available in 16Kb audio from Steve's site, transcriptions, too, GRC.com. You'll also find all the other great stuff he talks about including SpinRite, the world's best hard drive maintenance and recovery utility. If you have questions for Steve, you could tweet him. He's @SGgrc and apparently now accepting direct messages. But you know what, don't use the direct message unless there's a reason you don't want anybody else to see it; right? Public messages are better.

Steve: Everybody really liked it when I finally understood how Twitter was supposed to work and just lightened up. I was DMing everyone, and everyone just said, just, no. And so this whole...

Leo: We want to see it. It's public.

Steve: Yes. The @SGgrc works really well. And really, it's difficult many times to get into 140 characters what you want to ask.

Leo: Right.

Steve: You know, the problem is people want me to do SpinRite tech support through Twitter. And I say, look, I have Greg for that. He'll respond to your email. That's what I pay him for. I cannot do tech support in 140 characters.

Leo: But questions about the stuff we talk about here, that's okay. And of course you can do it on his website, GRC.com/feedback, if you need more.

Steve: That is where I go. Yeah, on Mondays, every other Monday I go there, I dump the mailbag, and I rummage through it to find the 10 questions. So that's really the way to get on the show is GRC.com/feedback.

Leo: And we have full audio quality and video, as well, at our website, TWiT.tv/sn for Security Now!. And you'll find it wherever you find podcasts. Steve and I have been doing this nearly as long as TWiT, almost 10 years. So there's lots of episodes, and it's all at TWiT.tv or in your favorite podcatcher.

Steve: Yeah. This was the second podcast. Yours was the first.

Leo: Right. TWiT was first. This Week in Tech was first, then Security Now!. Amber with Inside The Net, I think. Or, no, it was net@night at first.

Steve: Ah, right, net@night.

Leo: Then the Giz Wiz. And I don't remember after that. Windows Weekly came soon, and MacBreak Weekly came soon thereafter.

Steve: Sure, right.

Leo: And then it got out of control.

Steve: And going strong after 10 years.

Leo: Yeah. It's going great. I'm really pleased. And it's thanks to people like you, Steve, so I really appreciate it.

Steve: And it's thanks to our listeners.

Leo: And our great listeners, as always.

Steve: Yeah.

Leo: See you next week.

Steve: Thanks, buddy.

Leo: Bye-bye.