HTTP/2

Description: Leo and I catch up with several VERY interesting security events and stories of the week. Then we take a close look and a deep dive into the operation of the industry's first change in the official HTTP protocol in 15 years - the finalization and emergence of the HTTP/2 IETF specification which significantly streamlines web browser and web server interaction.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-495.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-495-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here with the latest security news, including that new hacking ring that Kaspersky discovered. What did they call it, the Equation Group? Details on that and a lot more, plus Steve's going to take a look at the new web protocol for speeding up page loads. It's from Google, and it's called HTTP/2. Details ahead on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 495, recorded Tuesday, February 17th, 2015: HTTP/2.

It's time for Security Now!, the show that protects you and your loved ones online, your privacy too. And here he is, the man in charge of Security Now!, Mr. Steven Gibson from GRC.com.

Steve Gibson: Yo, Leo. It's great to be with you. Well, we're 495 episodes in, in our 10th year, and we're closing in on Episode #500.

Leo: Amazing.

Steve: Yeah. I figured with all of this recent discussion of HTTP and the forthcoming spec for HTTP/2 and SPDY and how HTTPS over SPDY is faster, outperforms HTTP, it was time to take a look at, formally, at what is HTTP/2. We got no changes in the HTTP protocol since Version 1.1 in 1999, so 15 years ago. And we've been sort of tolerating it and coming up with workarounds for its various problems ever since.

So this week we're going to do - first of all, an amazing week of security stuff, largely because Kaspersky's having their security analysts conference down in Cancun, Mexico

right now, yesterday, today, and tomorrow. And so they've been dropping some bombs on the industry of the things that they've been tracking and aware of for some time. So we're going to talk about actually a little bit of a compromise that Google has made in their Project Zero ideology; the fact that Kaspersky has sort of unveiled a very longstanding advanced persistent threat in the global banking world; also some details on something that we actually found out about, believe it or not, on December 29th of 2013, but we didn't really - it didn't really catch our attention. And that's this hard drive firmware rewriting.

Leo: Unbelievable. This is the Equation stuff; right?

Steve: Yes.

Leo: Oh, man.

Steve: The Equation Group is the name that Kaspersky has given to them. And they're, like, the elite group who apparently feed the Stuxnet and the Regin group their technology. Also a fabulous tip about credit freezing to protect yourself from identity theft. And then we're going to plow into HTTP/2 and explain exactly what it is that the industry has pretty much already moved to, kind of quietly, but it's being finalized here.

Leo: A lot of interesting stuff.

Steve: It's a great podcast.

Leo: As you said, a great week.

Steve: Yeah.

Leo: And I think we should go to Cancun next year.

Steve: Yeah, when I found out that was going on, it's like - and there was Kaminsky, who I last saw when I was presenting. We both were, actually, on November 7th in Las Vegas for the DigiCert conference. And I thought, oh, Dan.

Leo: He went down there, huh? Yeah.

Steve: He's talking about who knows what. So, yeah.

Leo: Well, maybe next year. I just assumed Kaspersky would have its conference in Moscow in the middle of winter, and I thought, well, I'm not going to that. But

Cancun, well, now...

Steve: I think we understand why everybody would want to have a high attendance for that conference.

Leo: Yeah.

Steve: Now, I should mention we'll be talking about Carbanak.

Leo: That's something else.

Steve: That's the cybercrime group that's been stealing maybe as much as a billion dollars from the global banking market. But that's not Carbonite. Carbanak is something different.

Leo: Yeah.

Steve: I wanted to point out this picture of the week, courtesy of our often contributor to the show, Simon Zerafa. It's just such a perfect picture. It's the first page of the show notes for the week, showing a kid on a unicycle driving down a little commercial strip of shops, having just passed a sign that has huge blocks letters, "NO," and then underneath it "bicycle riding, rollerblading, roller skating, skateboarding, and scooter riding." Clearly, they were trying to say no to everything.

Anyway, the point is the caption of this is "Why Blacklisting Fails." Which is - this is just perfect because this kid is on a unicycle, not a bicycle, not a rollerblade, not roller skating, not skateboarding, not scooter riding. He has obeyed the sign. Yet he's riding his unicycle where they don't want him to be because they blacklisted certain things rather than saying "Only pedestrians," for example, only people using their feet. No, they went the other direction.

Leo: That would be whitelisting, wouldn't it, if they said only walking people.

Steve: Exactly. Only people walking. Oh, just perfect picture. So I love that and wanted to commend that to anybody who's interested. It's in the first page of the show notes.

So Google, apparently reacting to the backlash to their rigid, or I should say previously rigid, 90-day "we will disclose vulnerabilities whether they're been patched or not" policy of Project Zero, has softened their stance. In a rather, eh, sort of snarky blog posting, they start out by substantiating the success of this policy. They use Adobe as an example, talking about how Adobe has managed, with all of their multi-platform, multi-reach stuff, the Adobe Flash team has managed to successfully fix 37 Project Zero vulnerabilities within the 90-day deadline. But on the other hand we see Flash constantly doing out-of-cycle patches to fix these things. And the problem that we've discussed several times is that Microsoft, because they really, unless it's something dire, have to

stick with a 30-day policy with their standard second Tuesday of the month schedule because their patches need to be vetted by corporate IT. You could argue that the corporate IT team can just block Flash. But they can't block Windows. They are using Windows. So they have no choice.

So anyway, so Google says, you know, sort of saying everybody else manages to do this. And then they say, stepping back more generally, talking about the success of their Project Zero, that the 154 Project Zero bugs fixed so far, 85 percent were fixed within 90 days. And then if they further restrict that to essentially half of those, the 73 issues filed and fixed after October 1st of 2014, so that's sort of saying once we came up to speed, once everybody fully understood what Project Zero was, that is, the more recent problems, that number jumps to 95 percent of those 73 issues fixed more recently within the 90 days. And then they finally say, "Furthermore, recent well-discussed deadline misses" - and in those phrases they have three links which correspond to the Windows problem and the two OS X problems, they said - "were typically fixed very quickly after 90 days."

So what they've done as a consequence, I mean, so basically they're saying, okay, here's where we stand. We think we're doing something good for the industry. People have improved their responses, given Project Zero, that is, the most recent responses have been faster than the earlier ones. The 90-day deadline misses have dropped from 85 percent originally to only 5 percent misses, so 15 percent misses down to 5 percent misses. And those that missed were quickly fixed. So, okay, fine, Google, thank you very much. So now they're saying, "We've studied the above data," and I love the way they said this. They wrote, "and taken onboard some great debate," sort of like a small dinghy takes water onboard.

Leo: Some engineers wrote this.

Steve: "Taken onboard."

Leo: They've taken onboard some debate.

Steve: "Some great debate and external feedback around some of the corner cases." These are corner cases; right? Like did you mean greater than one or greater than or equal to one? Oh, well, we really meant to include the equality case. Okay, good, "for disclosure deadlines," writes Google. "We have improved the policy in the following ways. Weekends and holidays, if a deadline is due to expire on a weekend or..."

Leo: Oh, man.

Steve: I know, "or a U.S. public holiday, the deadline will be moved to the next normal working day." Okay. So, you know, like when the mail is delivered, except you do get mail on Saturdays still. But not on federal holidays. So if the deadline falls then, they'll bump it. They'll give you one or two days extra.

Then, grace period. And this is the big one: "We now have a 14-day grace period. If a 90-day deadline will expire, but a vendor lets us know before the deadline that a patch is scheduled for release on a specific day," so the vendor's got to come back to Google and

say, "Please, Google, we're ready, but we just need some more soup. I mean, we just need two more weeks." So if they'll tell Google the specific day within 14 days following the deadline when the patch will be released, "the public disclosure will be delayed until the availability of the patch. Public disclosure," writes Google, "of an unpatched issue now only occurs if a deadline will be significantly missed, by more than two weeks."

And then finally, the last change is Google wasn't dropping their Project Zero stuff into the standard CVE pot which is maintained by the industry. So the third change is the assignment of CVE numbering. Google writes: "CVEs are industry standard for uniquely identifying vulnerabilities. To avoid confusion, it's important that the first public mention of a vulnerability should include a CVE. For vulnerabilities that go past deadline" - thus the public disclosure - "we'll ensure that a CVE has been pre-assigned." So those are the changes. Basically it means another two weeks added to the existing 90 days.

Leo: I kind of understand the need to set a deadline because, as they point out, the three times that we had to reveal it publicly, it got patched right away. And so if you don't set a deadline, and you don't reveal it publicly, then companies just drag and drag and drag. But there's two things. This first strikes me as some real arrogance on the part of Google. Like, well, we're going to do it right, and obviously these other companies just don't care.

Steve: And somehow they have a problem.

Leo: They have a problem. I think you've got to give credit to Microsoft and Apple and whoever else that they do care about security, and they're going to try to patch these flaws. Give them some credit.

Steve: Certainly. And...

Leo: They're like a schoolteacher who says, "Well, you know, you didn't hand in your report, so we're going to give you an F."

Steve: Yeah.

Leo: We're grownups here. It also feels a little bit, I don't know, I don't want to say something politically incorrect. But it feels like it's snarky.

Steve: I just said, I used the term "snarky," yeah.

Leo: It's engineers who are being a little too literal-minded. You know?

Steve: Yeah.

Leo: You could just say, look, we're going to give you about 90 days. We're going to contact you, and we're going to beat the drum louder and louder if you don't fix this or something. You don't have to - I don't know.

Steve: Yeah. That's my reading, too.

Leo: We're not in school.

Steve: At the same time, thank you, Google. Thank you for...

Leo: Well, yeah. They got it fixed.

Steve: ...being a little accommodating. It's clearly, I mean, as I've said, from my perspective, the big problem is the timing of the 90-day counter relative to Microsoft's fixed 30-day schedule because it could be that just - what would it be? It would be just before a 90-day - I'm sorry. Just before one of Microsoft's second Tuesdays, a problem is found, and Microsoft is notified. Now there's three fixed cycles, but they're going to miss that because the 90 days will expire just before another one of Microsoft's scheduled releases. So it does make sense to add a little bit of forgiveness in there because Microsoft's challenges, I think, are well understood. It was driving people crazy.

Remember, once upon a time Microsoft would just drop these all the time, and IT was scrambling around trying - and there were some problems where they would patch things that would break IT software and then have to roll them back and so forth. So this has all been sort of negotiated over time. This is the way it makes sense to work. And other companies have followed suit. You know, everybody else now tries to do things on some schedule or in batches to minimize the problems this creates. So again, thank you, Google.

Leo: For being flexible.

Steve: For being flexible.

Leo: Or really more like for not being too inflexible.

Steve: Well, you know me, I mean, I've arranged my security certificates specifically to fudge Google. I'm thumbing my nose at them, keeping SHA-1 signed certs because there's nothing wrong with that. And my certs expire on December 31st at the end of the year so that I get the happy Chrome indicator.

Leo: It's that same kind of arrogance, to me. It really feels like...

Steve: Right, it is.

Leo: ...a real arrogance about - the subtext of this is that, oh, really Apple and Microsoft don't care as much as we do about your security.

Steve: Right.

Leo: And that's, I think, patently false.

Steve: Yeah. Microsoft had already announced a drop-dead, in 2017, at 2017, of SHA-1, in a context of there being nothing that anyone knows that's wrong with it. And Google just decided to push that forward because they've got the most popular browser on the planet now, and they can arbitrarily enforce this. I don't know, did you see that wonderful video of some guy trying to figure out what Chrome's trying to tell him with a red slash through the HTTP?

Leo: No.

Steve: Oh, it's been floating around for the last couple weeks. And it's like, here's a typical user, scratching his head.

Leo: Right, a normal user, yeah.

Steve: And he's like, okay, well, now - and he, like, figures out how to work around it. He, like, and it was so painful because this was some marketing spam in email. And if he clicked on it, he got this warning from Chrome because it was HTTPS. But if he, like, removed the HTTPS, which he doesn't understand, and just put www.marketingacceleratorsareus.com and some other crap, you know...

Leo: It worked.

Steve: Then it worked.

Leo: It's magic.

Steve: Without any security, now I don't have a notice. So, I mean, so this was just - it was just like, oh, so painful. But this is, you know, it'll be interesting to see how Google flags this stuff in Chrome and what problems it causes because the problem is Google's going to be trying to say something that users will notice, saying that websites are not secure, and the websites are going to be saying yes, we are. There's nothing wrong with our certificates. So, yeah, exactly as you say, Leo.

Leo: Hubris. It's hubris, yeah. But that's all right because, you know what? What

goes around, comes around, boys.

Steve: So the Carbanak cybercrime group, which is what Kaspersky has named them, there was a lot of news about this in the last week. So here's what we know about this. They've been operating silently since 2013. It feels Russian because it's sort of a social - it's a very sophisticated group doing social engineering phishing to get a foothold into a bank's network by getting someone to click on a link in email. That gets them in. Then they spend between two and four months patiently infiltrating the banking network. They get onto the network. They get to an executive's computer who has access to the video camera feeds, which they then start studying in order to learn how the bank operates, in order to impersonate the normal processes and operations of the bank so that, when they start acting, those actions will fit right in with the normal flow of what the banking is doing.

So we're talking sophistication. They have silently attacked at least a hundred banks, e-payment systems, and other financial institutions across 30 countries which Kaspersky has identified. And while this began back in 2013, it is still going on today. They limit their attacks to $10 million per bank, once they finally engage on a specific bank. And so they demonstrate enormous patience in essentially working to remain covert until they're ready to act. They have done things in two main ways. One is that they will get control of the bank's ATMs and cause the ATMs to emit cash, and then have a mule, knowing what the cash and the ATM schedule is, come by and pick up the cash as it's coming out of the ATM. And one of Kaspersky's unnamed banking clients lost $7.3 million through ATM cash withdrawal using a coordinated system of mules to go to specific ATMs at specific times and pick up the cash that was just magically being spit out by the ATM.

Then the other thing that they do is, once they've infiltrated a bank's cash management and account management network, they will change, they will edit account balances, for example, take a balance of a thousand dollars and add a zero to it. So now the balance says $10,000. But before anyone notices, and Kaspersky noticed that the banks typically only perform a verification suite every - I think I saw 10 days was the number. So very quickly after adding a zero, for example, to a thousand-dollar balance, turning it into $10,000, they then transfer 9,000 out of the bank to one of another set of accounts that they have set up, returning the original balance to 1,000. So the user's balance now looks correct, yet $9,000 is gone. And they've identified JPMorgan Chase as one of the recipients of funds from this group and also, in China, the Agricultural Bank of China as another recipient.

So this has been revealed, there was a press release just yesterday at the start of Kaspersky's security analysts conference, which is just going on now, as we mentioned, down in Cancun. And they said: "The cybercriminals began by gaining entry into an employee's computer through spearphishing, infecting the victim with the Carbanak malware. They were then able to jump into the internal network and track down administrators' computers for video surveillance. This allowed them to see and record everything that happened on the screens of staff who serviced the cash transfer systems. In this way the cybercriminals" - so they were essentially spying on the spies, the internal spies of the bank. So "the cybercriminals got to know every last detail of the bank clerks' work and were able to mimic staff activity in order to transfer money and cash out."

So, and what's really interesting, too, is that banks have never acknowledged this previously. They're all keeping mum about it. And one of the things that I guess Obama was talking about last Friday when he was in Silicon Valley, talking about the cybercrime

bill that he wants to get made, is that there would be reporting requirements that would not have allowed banks to keep this quiet as they have been. But this has been silently going on for probably maybe two years.

Leo: Or more, goodness knows.

Steve: Or more. Yeah. And in fact...

Leo: I mean, it's been going on forever. This particular hack, though.

Steve: Yeah, Kaspersky did acknowledge that their own verification, they've been able to verify $300 million, but expect triple that. And now we're seeing numbers like a billion, which was about triple 300 million. So a lot of money. And also, interestingly enough, the banks are absorbing this. I mean, this is not something that they're blaming their accountholders on because it's not their accountholders' fault. It's their own internal network security which is being compromised.

Leo: Wow.

Steve: So the biggie is this hard disk drive firmware overwriting. Kaspersky has named this group, which is apparently a group within the NSA, they call them the Equation Group because they have a great affection, apparently, for using cryptography and all the various cipher technology in order to obscure and manage the malware that they're using. The Kaspersky group has put together a riveting, I think it's 44 pages, an FAQ in the form of a PDF. I created a bit.ly link because I wanted to turn people onto this. It is really interesting. So it's bit.ly/bad-hdd. I called it that because people were all concerned recently about the BadUSB, which is what that was called.

Well, this is essentially the equivalent of BadHDD, that is, bad hard drives. So what we know is that there is - this has been specifically done in targeted attacks. There is malware which is deliberately overwriting the firmware in hard drives. Kaspersky has found two different versions which they call version three point something or other and four point something or other. The four point something or other understands not only hard disk drives, but also SSDs because there were names like OCX and OWZ and, you know, the various SSD manufacturers, more than a dozen different makes and models of drives.

It turns out that hard drives probably all have the same problem that some thumb drives have, and that is they've got, we know that they've got very sophisticated controllers on them. It was a bit of a wakeup call that USB thumb drives would have sophisticated controllers. But we know that hard drives do. There is a well-documented standard in the so-called ATAPI interface. The ATAPI API is what, for example, I with SpinRite use a great deal. There is a download microcode command that allows external microcode to be provided to the drive.

The problem is there is no upload microcode. That is, the API itself is write-only. The hard drive manufacturers do that on purpose because they don't want anybody reading out the microcode. I mean, this is proprietary. They've got fancy error correction technology. Essentially, each hard drive manufacturer considers the firmware on their

drives to be crown jewels. Their intellectual property, you know, the exact management of defects, the exact management of the technology on the drive is theirs. They don't want anybody reading it out, getting access to it. Some hard drives have the firmware stored off-chip, that is, in a separate little serial flash which does make it downloadable. And some hard drives also use the ARM architecture, which means that they have a known instruction set. And if the external firmware is not stored on the same chip, it makes it easy to access it and reverse engineer it.

Now, Kaspersky believes that this Equation Group is part of the NSA because they have seen that this Equation Group firmware or malware, which is laid out in this 44-page PDF, apparently was employing zero-day flaws before Stuxnet and Regin were both employing them. It's not clear whether the Stuxnet Group and Regin Group got them from the Equation Group. But this looks like a group within the NSA.

And the other thing which really sort of puts the frosting on this is we read about this and were chuckling about it in that large dump - you'll remember, Leo - that Der Spiegel made back late in 2013 as part of the original multiple rounds of Edward Snowden dumps. This is, and F-Secure reminds us of that, this is the IrateMonk NSA slide. The NSA slide that disclosed this IrateMonk project had it tagged as TS/SI/REL. And I'm sure that TS is Top Secret. I don't know what the SI or the REL are.

But that slide said, "IrateMonk provides software application persistence on desktop and laptop computers by implanting the hard drive firmware to gain execution through master boot record substitution. The technique supports systems without RAID hardware that boot from a variety of Western Digital, Seagate, Maxtor, and Samsung hard drives. The supported file systems are FAT, NTFS, EXT3, and UFS. Through remote access or interdiction, UnitedDrake" - which is another one of the NSA programs in these slides - "or StraightBlaze are used in conjunction with SlickerVicar to upload the hard drive firmware onto the target machine to implant IrateMonk and its payload, the implant installer. Once implanted, IrateMonk's frequency of execution" - that is, dropping the payload - "is configurable and will occur when the target machine powers on."

So this sort of closes the loop on this. We saw that slide. We talked about it. But it was a little bit maybe sort of like overwhelming. I mean, there were just so much of this NSA disclosure back then that this was just one of among many. Remember we were talking about passive eavesdropping and things you could beam microwaves out that would bounce things back and just, you know, this was among all that technology. Here, though, we've got Kaspersky, who has found samples in the wild. They've been able to pull this stuff out of drive firmware. As I said, sometimes the flash ROM is not in the CPU chip itself, which would have made it much more difficult to access. But it's a little eight-pin surface-mount chip that they're able to pull. So they were able to reverse engineer this and see the signatures of all of the various drives that this thing is aware of and that it knows how to, essentially, how to operate.

Now, someone at Kaspersky said that, without the source code, this would be impossible. I would argue that's not the case. It would be true if you could not read out the firmware because, if you're dealing with flash that's on the same chip as the processor, and you only have write access to the firmware, you would absolutely have to know everything about the hardware architecture like only the manufacture could in order to knowledgeably perform a write-only overwrite of that processor's firmware and have anything survive. If, however, the serial EEPROM is outside the chip, then you could certainly obtain what it has. And if you knew that this was a standard CPU architecture, for example, they seem to be using ARM throughout the industry, then you'd disassemble it and start reverse engineering it. So it certainly is possible.

However, the other way that it may have been that the government or government agencies had access to source, not through reverse engineering, is they may have been doing large drive purchases and said to these various drive manufacturers, look, we're happy to sign a nondisclosure agreement, but we're buying from you essentially computers with software in them when we are buying hard drives from you. We need, under NDA, you to provide the source for the firmware. And in order to make a large sale to the government, we know that manufacturers like Western Digital and Seagate and Samsung and so forth, they'll turn over their source under a promise that it will never be disclosed. So it may very well be that an agency would have said, okay, NSA, we need you to vet this source code of these WD drives we're buying, and the NSA would have been more than happy to do that and probably would have retained copies.

So it's foreseeable, we can understand how the law enforcement agencies with the U.S. government could have obtained the source, and this is very powerful. Essentially it gives them three things. It gives them persistence, invisibility, and local caching. For persistence they get extreme hardware lock persistence which survives everything - disk reformatting, OS reinstallation. If you suspect something a little funky is going on in your computer because you're not sure, what do you do? You reformat the hard drive. You reinstall the OS. This thing is able to live through that. It gets into the firmware of the drive, you can't get rid of it.

Invisibility, of course, once the drive is infected with a malicious payload, the drive itself, the firmware, you can't scan for it. It doesn't appear. Only its behavior, if you happen to catch it misbehaving in this fashion, then you'd have a chance. The problem there is that the misbehavior is going to happen as the first thing during the system's boot-up. It's going to replace the master boot record, and maybe not every time. So you could watch it looking just fine and say, okay, well, looks good. But it only takes doing it once in order for it to swap a master boot record with something else in order to preload its own malware, which then is there before the OS and, as we know, essentially functions like a rootkit. But this one you can't get rid of because now it's built into the hard drive. And that would be a pre-boot rootkit that is then able to subvert the operation of the operating system because the operating system has to rely on the first things that load in order to check that everything else that loads subsequently is all right.

And then, finally, local caching. What this could do is it could set aside some chunk of hard drive space. So it's not constrained to just the write-only firmware. It can take back some space from the hard drive which, since it's now the modified firmware, it no longer allows access to that region. If it decreased the apparent physical size of the drive, that could be a tipoff that this drive has been infected because it would be reporting a smaller sector count than exists. On the other hand, drives do report in some cases a reduced size if their spares pool of bad sectors that are being swapped out for defects, if the spares pool becomes exhausted, the drives have the ability to reduce their reported size, essentially taking back unused sectors. So it's not the case that all drives of a same make and model always even report the same physical size. So you can't rely on the drive saying that it's a little bit smaller as a means of detecting that. So that's a problem.

But what that does is it gives the drive huge amounts of space, that is, it gives the malware huge amounts of space where it could sequester sort of pre-exfiltrated information, waiting, for example, for a system that doesn't always have network connectivity, or doesn't have a USB plugged into it. It could store the stuff there so that, again, no scan could find it. And then, under a certain set of circumstances, it is able then to dump that out when it has access to some sort of communications facility. So, wow. These are the details, it looks like, of the IrateMonk project that the Snowden slides talked about, but we never actually had samples found in the wild. And now we know that this exists, and it has been used globally, essentially, around the country and around

the world.

I did want to say also that, again, our friend of the show Simon Zerafa retweeted a link that I remember him tweeting some time ago. Wait a minute. And I'm thinking that - oh, no. I was going to say I thought I created a bit.ly link for it, but I didn't. I did tweet it just now, so anyone who's interested can check my Twitter feed. There is an interesting project at SpritesMods.com, S-P-R-I-T-E-S-M-O-D-S dotcom, where somebody reverse engineered a hard drive. So it's all of this stuff. If you want to see, if you're interested, a real-world look at reverse engineering a hard drive, where essentially there's a standard interface called JTAG which is an embedded processor debugging interface that allows an attachment to essentially read and write firmware, set breakpoints, start and stop embedded processors. It turns out that hard drives often have JTAG interfaces which are enabled. And if you know what you're doing at the hardware level, you can connect to a hard drive and play with it at a level below the officially sanctioned ATAPI API. And this SpritesMods.com site has a really interesting multipage report about that being done.

Okay. Lastly, and this is - I probably should have done this first, but these other stories were just too interesting, and the industry's been buzzing about them. I created a bit.ly link for this, and it's important: bit.ly/freezecredit, all lowercase. So bit.ly slash F-R-E-E-Z-E-C-R-E-D-I-T. This takes you to a page where the guy explains that a service that is available to anyone for all three major credit bureaus - Equifax, Experian, and TransUnion - allows consumers who are not the victims of identity theft to lock their credit reports. It's not free, but it's not too expensive. The cost varies, depending upon where you are, from $3 to $10. In California it was $10. And because I was curious to do this, although I wasn't worried about this recent Anthem breach, I locked my credit reports. It was $10 for each of them.

Leo: You may have to pay to thaw it, as well, and this is important.

Steve: Yes, that is correct. It is not an annual fee. It is a one-time lock. But you do also - and it's annoying. First of all, it's annoying that this costs anything because they're making money from being a credit aggregator. We're not making any money from the fact that they have these reports. And of course the danger is that somebody could - and in fact, from the Anthem breach, all the information is required to apply for credit in somebody else's name. And so the idea is that somebody would impersonate you, apply for credit in your name, then the credit-granting institution would pull your credit report to see if your credit is good; and, if so, grant this person credit in your name. They would then run up a big balance, whatever they were able to, take the money and run. And then you're stuck holding the bill, potentially, because they've been able to impersonate you.

By freezing your credit reports, you are preventing that credit-granting agency from obtaining your credit report. It's locked. And so that's a double-edged sword, of course. It means, first of all, you prevent new credit from being acquired or granted in your name. At the same time, you prevent that, and maybe it's what you want. So, for example, in my case, I haven't applied for credit in decades. So I was more than happy to lock my credit report across all three agencies. And if at some point in the future I need to get credit or have someone look at my credit, then I can pay $10 each for one or more of these companies to either temporarily - you're able to, per query, allow a query to be made or to remove the lock. And when you apply the lock, you are given a long PIN that they generate which you must keep because that now becomes your out-of-band means of thawing this freeze on your credit.

Anyway, I wanted - I understand that there are people for whom freezing is not practical because they're busy in their lives. They're getting more credit cards or buying homes or cars or whatever, all of which this would be a problem for. But if you're at a point where you don't mind blocking the world from seeing your credit report, and I just did that, I think this is a great option for people who are concerned about identity theft. So bit.ly/freezecredit will take you to a page that explains this and contains links, which is what I followed, to all three of the major bureaus. So I'm glad to know that service is available. I wasn't aware of it before.

Leo: Yeah, I mean, I think this is probably something new. You used to be able to do this kind of thing, lock your credit report. But they didn't make any money on it, and they still pretty much discouraged it and changed the laws and - these companies, these three companies, have huge legislative clout because they make a lot of money. And I agree, this should be free. But they don't do it for free because they're replacing the revenue that they get from giving your information to companies with revenue from you.

Steve: Exactly, from you saying yes and no.

Leo: Right.

Steve: Yup. Yeah, and I...

Leo: But it's a good link. Save it because it does explain how to do it for each of the three credit unions, and each is different.

Steve: Correct.

Leo: Credit reporting, not credit unions, credit reporting systems.

Steve: Yeah, and in fact one of them, which one...

Leo: I think it's TransUnion.

Steve: I saw it here a second ago.

Leo: You have to call them. Oh, no, that's not it.

Steve: No, I was able to do it online. Oh, but it was TransUnion that made me create an account, which was so annoying.

Leo: Right.

Steve: The other two don't. You have to prove your identity by knowing - but unfortunately you're proving your identity knowing everything that Anthem just released about you.

Leo: Well, but remember, at TransUnion, Equifax, they know it anyway. You're not giving them information they don't know. That's how they can ask you that question.

Steve: Correct. Yeah, but my point was that anyone who got the data from Anthem also could, for example...

Leo: Oh, could do the same thing, yeah, yeah, yeah, yeah.

Steve: ...could do the same thing. So, and then you receive this long PIN, and you need to hold onto that. But it was - I didn't have to talk to anybody. I did it online. They did take $10, each of them, from my credit card that I provided them. And now my three - my credit report is locked for all three of them. And I'm glad. So, and I know our listeners, I'll bet this appeals to a bunch of them. So I wanted everyone to know because I didn't know about that before.

Leo: Mm-hmm, mm-hmm.

Steve: I just finished the second book in the Expanse series, "Caliban's War." And if anything, it was even better than the first one. So I'm at the point now where I can at least recommend the first two. Mark Thompson, who's read them all, tells me that number three is at least as good as the first two. I haven't started yet on number three. But it really is fun. So I just wanted to commend it to everybody.

Also, there's now a trailer for the Expanse series coming to Syfy on the Syfy.com, S-Y-F-Y dotcom site, called "The Expanse." And it looks like it may be pretty good. I'll tease people just a little bit. And this is not a spoiler. I hate spoilers, so I wouldn't ever do one. But you learn this right off the bat, so this is not spoiling anything, and that is that another, an alien intelligence intended to essentially cultivate the Earth with its own organism, and so some billions of years ago launched a biological package at the third planet from the Sun, except that just by chance it got captured by one of the outer planets and became an ice moon. And it sat there for billions of years while life evolved on the Earth, and then we colonized the solar system. And it got discovered. So I won't say anything more about what happens, but it is a lot of fun.

And then the other thing that this book spends time on, which is I think just intellectually interesting, is that, as the solar system is colonized, we colonize Mars and then the so-called "Belt" further out - and "Belters" is a common topic in sci-fi, you know, who are like mining for minerals and ice and so forth in order to supply the required raw materials for a solar system-wide civilization. As is sort of going to happen, the needs of the various groups of Earthers and Martians and Belters differ, and that creates political divides, and they sort of drift apart over time. So anyway, it's just good science fiction.

And I do want to add it to my list of recommendations. I like it. And we're going to get, you know, it's going to be a TV series on Syfy. As people know, I wanted to read it first, and I'm in the process of doing that.

I wanted just to mention actually sort of a follow-up from a testimonial from an Igor Koveshnikov in New Jersey. I didn't mention this last time when he shared his story. Remember that he, we talked about him only a few weeks ago, he had installed an SSD in his boss's laptop that went bad, and he was really curious to find out that it went bad. The good news was that SpinRite fixed it.

But he said: "Related to my testimonial, as far as I understand, SSD completely hides its internal structure behind wear leveling and optimization in firmware and represents the drive externally as a standard mechanical drive. When SpinRite runs on Level 2 and higher, it reads a sector and then writes it, supposedly to the same sector. But SSD is using wear leveling and writes data to whatever its algorithm decides are the most appropriate cells in memory and may not be the same as where the data was read from.

"Is there any way to get closer to the hardware of SSD" - oh, and he says and SSHD, meaning a hybrid - "or will it always be a black box? Will it be addressed in the new version of SpinRite? Right now, according to my own and many other users' experiences, we know that SpinRite repairs SSDs just as it does hard drives. But I think we can only speculate on how. If you ever come across more info on internal mechanics of SSDs, please share it with us, preferably on Security Now!." Then he said, "Shout out to my wife. She's a software security specialist and listens to every episode of Security Now!."

So anyway, just to quickly answer Igor's question, as we know, it is the case that wear leveling will cause the sector which SpinRite may have recovered to be written somewhere else. But we really don't care. In fact, that's almost - it's almost exactly what happens with hard drives that have spare sectors, when the sector goes bad to the point that it can no longer be safely corrected. And as we know, modern hard drives allow a lot of correction before they get scared about how much correction they're having to apply and then decide they need to relocate that to somewhere else. If it happens that a sector is written to a different location, it's the recovered data after SpinRite has recovered the data from the bad spot that the drive, the SSD, may itself choose not to use. But even if it thinks that the sector is still fine and rewrites it somewhere else, the wear leveling mapping logic will know where it went so that, when that same numbered sector is read again, it reads from the right place. So the badness will sort of have been opportunistically mapped out, even if it wasn't deliberately mapped out.

So again, wear leveling helps us. And the SSD is preserving itself by making sure in general that we're writing the same amount across all of the SSD's physical surface, even if we're actually tending to write specifically numbered sectors much more often than we are others, which as we know is the typical pattern of hard drive use. So SpinRite does its job, and the SSD does its, and it all works out for the best.

Leo: It's so easy.

Steve: Yeah. Nothing to it.

Leo: Nothing to it.

Steve: Nothing.

Leo: All right. Back we go to the subject at hand, in this case HTTP/2. SPDY, S-P-D-Y, was also from Google; right? But they're abandoning it now.

Steve: So, yes. I wanted, for the sake of "fair and balanced," to say that I thought it was interesting that Microsoft on their pages is taking credit for HTTP/2 and somehow manages never to mention Google once.

Leo: Did Microsoft have anything to do with it at all?

Steve: Nothing whatsoever.

Leo: Oh, dear.

Steve: Nothing whatsoever, no. They are the absolute beneficiary of what I think is Google operating in the best of all possible ways. We know that I'm annoyed with Google when they throw their weight around, as we were talking about earlier, with arbitrary deadlines on the speed at which they would like to see the industry moving forward on various things.

But Google is at their best when they are spinning off groups to experiment with new protocols. And nowhere ever have we seen a better example of that than with what started off as a research project, SPDY, which was not an acronym or an abbreviation, it just meant "faster," to examine what could be done to improve HTTP. We've been with HTTP/1.1, which is essentially HTTP/1. There were a few changes that were made. For example, technically with 1.1 you could do something called "pipelining," that is, you could send other queries down the connection, even though you hadn't yet received the results from the first query. But there were problems with making that happen.

Leo: [Sneezes] Excuse me.

Steve: Of course. There were problems with making that happen, and it never ended up being practical. And it turned out that it wasn't of much benefit. So for 15 years we've been living with the same thing. So what's wrong with what we have now? I'll talk about that, HTTP/1. Then we'll do another break for a commercial so I can rest my voice, and then I'm going to tell you about HTTP/2.

So HTTP/1 is what we've had. It's what today virtually everybody is using. The problem is that, in the last 15 years, web pages have gone crazy. They've gone from a page of text that maybe had a few pictures to ridiculous, asset-heavy, graphics-heavy, script-heavy, where all kinds of stuff, the page's text, its layout, all kinds of ads and images, scripting coming from libraries being sourced from all over the place, are all being sucked in and pulled together to display one amazingly complex multimedia page event, essentially.

So the way this happens is the browser requests sort of the raw content, the description of what the page will have, essentially in a resource from the server that hosts the page

in HTML. Then the browser looks through all of the other assets that the page wants to have - links to advertisers, links to the JavaScript that it needs to run, links to Flash objects, so links to plugins that it may need to load, I mean, just everything. And today's pages are just packed with all this stuff. So then the browser needs to turn around and start asking this array of other websites - lots of stuff no doubt from the same website, but also lots of other ones. It needs to ask all of them to start supplying all these things so that it can assemble the page.

So the problem is that the HTTP protocol is the only thing we have, the HTTP/1, for doing that. So the browser opens up, because it's now in a huge hurry, because pages have gone crazy, and because HTTP can effectively only be used for one thing at a time, that is, the protocol is a request for something, and then that something is returned, and then often the connection is dropped. Now, one of the things that we did get that is used in HTTP/1.1, thank goodness, is something called "keep-alive," where at least we don't need to create another TCP connection. We can make another HTTP request, a query, over the same TCP connection. So that's good. Otherwise we'd be in big trouble. But even so, it's ask for and then receive; ask for the next thing and then receive; ask for the next thing, then receive.

So it was originally the case that there was a two-connection-per-server constraint placed on browsers, just by convention. The industry said it's impolite to open more than two connections from a browser back to a single web server because that would just overload it. It would be too many connections. Well, servers got faster, so that that wasn't such a problem for them. And it was just a matter of demand, really. Pages got so complex that web browsers began cheating. They said, well, you know, Firefox wants to look faster, so we're going to allow six connections to a server. And then Microsoft said, wait a minute, we're going to go for eight to make our pages load faster. So it was clear that the notion of trying to ask browsers to behave themselves and not open lots of connections back to the same server, that just wasn't going very well.

So there were several problems with that. The benefit was that now you had parallel pipes back to the server. And so that was good because now you could, for example, be receiving at least six things at once, rather than having - and they could be done in parallel rather than needing to individually wait for each one to come back over a single connection. The problem is that opening a TCP connection is not fast. That is, TCP - and we've discussed the way TCP operates in detail in the past. TCP has a problem, and this is just inherent to a packet-switching network like the Internet. That is to say, how fast should we go?

We know that if we go slower than we know we can, then we're not utilizing the available bandwidth. But if we go faster than we can at the chokepoint that may exist somewhere between the client and the server, if we go too fast, then buffers will overfill in routers, and packets will start getting dropped, and that's bad because there's a delay in us finding out that packets have been dropped because we only know they've been dropped when they're not acknowledged by the receiver in TCP. So TCP adopts sort of a careful exploring the bandwidth, where it slowly - it's called "slow-start," where it slowly begins sending more and more packets and waiting until it senses that it's beginning - that some are being lost, and then it backs off a little bit. And then it starts creeping forward again until it has a problem, then it backs off.

The point is that, if we open lots of connections, they all have to independently, individually go through the same TCP slow-start process in order to sense the bandwidth between the two points. And so that's not optimal. And if we're connecting to a secure server, every one of those TCP connections, after establishing TCP, has to then establish the TLS connection, go through the whole security handshake rigmarole in order to

establish security. And only once that's done can they begin doing their one-request-at-a-time, HTTP/1-style requests.

So it's been a big mess. There's been pressure to move more things at once because pages have become huge. Browsers have to manage all of this at their end, and that's becoming a big problem. There is sort of this fight between how many connections, how many parallel TCP connections in order to then do TLS, and that puts a security burden on the server for handling all the crypto, and then all of these HTTP connections to the browser. So that's the world that we've been living in. And Google decided we must be able to do something better.

One of the other problems with HTTP, and we've talked about the HTTP protocol, there are request headers that we never see. The user doesn't see them, but they're things that specify, for example, the user agent, the name and model number of the browser that is making the request. And if the browser has a cache, that is, it may have a copy of some of these things, so it's able to say "I want this, but only send it to me if it's newer than the one I have in my cache. And let me know, in which case I'll know that I still have the current thing." So there's a whole bunch of other stuff going on behind the scenes.

And of course we know about cookies, too. Browsers often have lots of big, long cookies which are relatively static for a given server. But every single request the browser makes sends these cookies back to the server as part of the request to say this is the state that I currently have with the server, if that changes the server's behavior relative to the browser. So there's a huge amount of redundancy in what the browser is sending to the server, and no effective way to remove it.

So that's the problem that we've been living with for 15 years. It's been getting worse and worse because pages have just gone crazy. They've exploded with complexity. And it's been time to come up with a replacement, an improvement. And to their unending credit, Google has done it, and they have done a beautiful job. We're going to talk about that next.

Leo: Oh, how exciting. It's not "S," is it. Can it not be "S"?

Steve: It's a very good point. I was going to bring that up, but that's a perfect segue. The definition for HTTP/2 does allow it to operate non-securely, that is, without a TLS tunnel. Although it's in the spec, nobody currently does that. So it's going to be very interesting to see whether servers that offer full HTTP/2, when they finally make the migration from SPDY - right now everybody technically is still running SPDY. For example, the draft that I read last night in order to bring myself completely up to the minute on what /2 is doing was Draft 17, which is the latest one. And it is dated February 11, 2015, meaning six days ago.

Leo: Wow. Wow.

Steve: Yeah. So they're at the point of crossing their T's and dotting their I's. So it's not like there are any big changes remaining. And there is already support for HTTP/2 among, I think, I know that there's an Apache mod. Nginx supports it. Microsoft has announced, even though they don't talk about where it came from, they do announce that somehow it will be appearing in Windows 10 Server. And it is right now in the

Windows 10 preview is HTTP/2 per the current spec. And all the browsers support it. As we mentioned last week, Chrome will be dropping support for SPDY next year. There'll be like a year overlap.

And I think it was with Chrome 40 we talked about last week, Chrome will be formally supporting HTTP/2, which of anyone in the world, the Chromium people have the smallest changes to make in their browser because they already had SPDY. Of course, Firefox supports it. Safari supports it. It's essentially ready. So I think we're going to see it happen soon. The reason is it really works. This is not, you know, one of the things we've talked about is why is it that so many better things take so long to get adopted or never really get off the ground. It's because there is adoption inertia. If you've got something that is, well, this works well enough, then there's just no impetus to change. It's the reason, for example, that I have believed that SQRL has a chance, because usernames and passwords just don't work well enough. I mean, they're really broken. But if something's not really broken, it's like, eh, you know.

And as it is it took 15 years to fix HTTP/1. And while you couldn't really say it's really broken, look what I've just described as what the industry has had to go through in order to keep ahead of the fact that the protocol really was no longer up to the game that it was being asked to play. So SPDY, which is for all intents and purposes HTTP/2, is what you get when an independent team who really understands the way the Internet and the web work can say we're going to not do 1.2, but we're going to do 2.0. And by the way, it's not .0, it's just /2. And that was the case deliberately. They didn't want to create some notion of subversions and things.

This is a complete break from what we have had. There is no backward compatibility between 2 and 1.1 or anything before it. Zero. This is we're going to start over and do it right, do it well. And you could argue /1 was right for the times; /2 is right for today. So the first thing that they do is one connection, period. Now, at first you think, wait a minute, wouldn't two be faster? And if you stop and think about it for a second, no, because 2 is going to have to move over the same bandwidth. If you've got two connections to something remote, they're still going to have to come over the same channel. So why not run one connection twice as fast than two connections half as fast? So one connection.

And the other reasons we want only one connection is, as I was saying, all the other problems associated with multiple parallel connections. TCP, you don't want to have to ramp it up and have six or them or even two of them independently figuring out how fast they can go because they're both going too slowly. Instead, have one connection and have it figure out how fast it can go, and it'll be going as fast as it can. So that makes more sense, one TLS security negotiation, one, rather than even two, just one.

And the other thing is that, if you think about it, the existing HTTP protocol is a hierarchy of standards. And as we know, the whole layered networking model, we'll skip some of the lower layers because the ones we're concerned with are you first make a TCP connection, then maybe you bring up a secure tunnel on top of the TCP connection, and then inside that tunnel you put HTTP requests. The problem is that we still have to make a TCP connection. We still have to bring up a secure tunnel. But what if there was some way of signaling that what we were going to be doing with the secure tunnel was HTTP/2?

And again, the Google guys did not miss a trick. They've added a pseudo cipher suite to the TLS v1.2 spec, which is the one we're at now that has wide adoption. Even GRC supports it, so you know it's been around for a while. The pseudo cipher suite is a clue to tell the server, during the security negotiation, we're going to be doing HTTP/2 over this

connection. So they short-circuit another round of sort of protocol negotiation delay, and the server then is able in that negotiation to acknowledge it's able to do HTTP/2, let's go.

And because this is a fundamentally asynchronous protocol, and I'll explain what that means a little bit later, but the server has something called "speculative push," which is sort of the equivalent of speculative processing. The very latest CPUs, when they're executing instructions and come to a branch, if they're so far ahead by doing what they can of, for example, of the instructions being finished which will determine which branch to take, these CPUs take both branches. It's not the road less traveled. It's, well, just travel both roads. And so that's called "speculative execution." The chip will take both branches, start executing instructions down both. And then, once the results from a previous instruction are known, it discards the branch that turned out not to be taken in that case. So this is a little bit different. This is speculative push, where nothing prevents a server from sending ahead some things that, for whatever reason, it knows the browser is ultimately going to get around to asking for. So this is just really cool.

So we have one connection, ever. There's no reason, no benefit conceivable beyond that. And so we only have one connection between the client and the server. And we short-circuit the argument or the questioning about whether each end supports HTTP/2 by tucking that information into hints in the security negotiation. So that's all resolved immediately.

The next thing we do is agree that we're going to divide this single connection into frames. So frames is an abstraction that HTTP/2 lays on top of its single connection. And what that does is it allows it to support simultaneous multiple streams. You have a 9-byte frame header. Now, remember, this is distinct from packets. TCP, the underlying communications protocol, that's chopping things into packets in order to move them around. And TCP guarantees that lost packets will get replaced, and that out-of-order packets will get reorganized in the proper flow by the time they need to get used. So what HTTP/2 can rely on is that its communications is just seen as a single stream, that is, what it sends, what either end sends comes out of the other end eventually in the same and proper order.

So what it does is it divides the TCP stream into arbitrary length frames, so frames on top of the TCP stream. And there's a header that is 9 bytes on the top of the frame. So there's a little bit of a per-frame overhead of 9 bytes, meaning that you wouldn't want to have lots of itty-bitty frames because then the payload of the frame starts getting large relative to the per-frame header. On the other hand, you don't want to have really monstrous frames; otherwise, whatever is in that frame is dominating the stream. So the point is you get a benefit from chopping what would normally be a single-stream connection into multiple virtual connections using this frame abstraction.

So you have this 9-byte header. The first 24 bits of it, so 3 bytes, is the frame's length, the frame's payload length, meaning except for the header - you ignore the header - the length of the payload that the frame is carrying. And you only ever use 14 of those 24 bits without permission from the other end to use more. So they've given themselves plenty of growing space. Up to 16MB is 24 bits. But normally they only use 16K bytes, which is 14 bits, as a maximum size of a frame. And most frames are probably going to be a lot smaller than that.

Then there's an 8-bit, that is, 1-byte frame type, so that gives you up to 256 possible frame types, of which right now they use only a handful. Then you've got a 1-byte set of flags which are frame type specific. So different frame types may or may not need some flag bits; and, if so, this gives them to them. And then, finally, a 32-bit stream ID, that is to say, what of many streams is this frame the next one of? And for whatever reason,

the high bit of that is always zero, must be zero when being sent and must be ignored when received. So technically it's a 31-bit stream ID in a 32-bit space. So that's the 9 bytes.

But basically that means we've got frames that can range from very small to up to 16K, unless we receive permission from the other end to use even bigger frames. And then basically a frame type, some flags for that frame if it needs them. But then the cool thing is a 32-bit stream ID. So the 32-bit stream ID, and that's 4GB of possible stream, so that's just, you know, it's big because, not that we need that many, but we don't want to run out of them. What that says is that, as I said, this frame is the next one of that stream. And the simultaneous multiple streams allows essentially a multiplex conversation between the two endpoints. That is, it allows requests - it allows, for example, the browser to just, at its whim, to arbitrarily create new streams by successively numbering stream IDs and to stick requests for web resources, scripts and assets and so forth, from that same server.

That is, remember, one connection is between one client and one remote server. You might have other SPDY, or I should say HTTP/2 connections, one each, to other servers that are also contributing content to this page, but only one to that one server. So the browser just sits there emitting queries for assets, giving them successively numbered stream IDs, over the same connection. Those all then begin arriving at the server. It starts finding them and sending the response over an otherwise similar HTTP protocol back to the server.

Now, that's worth mentioning. The actual content of these streams is HTTP protocol, that is, the familiar one, the text-based protocol with request headers and so forth. So the nice thing is that we haven't redefined the HTTP protocol on the wire, that goes into the wire. What we've done is we've sort of created - you can think of it as a shim. That is, you could have something on the other side that is still just generating standard traditional HTTP queries. Once upon a time, that would have had to wait for essentially available connection space or a free connection in order to put the query on the wire. Now that standard HTTP query, thanks to HTTP/2, which manages how we handle wires, it's able to send them all out as fast as it's able to generate them, give them individual stream IDs and shoot them out there. But my point was the format of the query data, the query itself, that stayed the same.

Now, the server, then, is sending all this stuff back. The other thing that the client can do is give streams priorities and interdependencies. It's able to say, I want this stream at high priority. For example, the main, the base page's main HTML, which the browser needs to have back immediately in order to have all the other URLs that it needs to request, it's going to say, get that back to me absolute top priority. Don't slow that down on behalf of other assets that I may also be asking for or may start asking for even before I've got the whole main base page back. So it's able to say that all the other assets are dependent upon this first stream, or it's able to say they're lower priority. So there's a sort of an interlocking semantics that allows priorities and dependencies to be resolved by whichever end is doing the sending.

Now, so what we've got is one connection, optimized for speed. It comes up, we get going immediately, and the client is able to start sending requests down, packetizing them in individual frames, giving them unique streams. They come out at the server end. The server starts, it looks at the dependencies and the priorities, and that allows it, as it's accumulating this increasing block of stuff it wants to send back, it allows it to send them back with the priority and dependency that the client has requested and, essentially, for there never to be a dull moment.

Notice that it's accumulating a big blob of stuff, and it's going to be squirting it through the connection as fast as it can, given the constraints that the client has put upon it. So it's going to keep this one connection absolutely maximally busy. There will not be pauses in between assets being requested and sent back. So we get absolute much better utilization of that one connection even than if we had six connections or more as we at one point would have because each one of those six would have still been an ask and get, ask and get, ask and get. And the server would have been sitting here, like essentially waiting for the things it sent to get there so that it could receive another request over one of those connections for the next thing that the client knew that it wanted. So this completely transforms the nature of the way data is able to move from the server to the client.

Now, the other cool thing that I just - I didn't ever pay attention to this before, but I had to in order to understand how header compression was being done because, as I mentioned, there's a huge amount of redundancy in the way headers are handled. The client is almost always emitting the same header contents. And that's hugely redundant when you're asking for a whole bunch of things from the same server. They're coming from the same user agent. They're going to be carrying the same cookies. It's going to be the same time of day. There's all these things that are not changing. Yet, as I said, the HTTP textual protocol is still carried unmodified, which is very clever.

Okay. To understand header compression, we need to know a little bit about the way compression works. That is, these two geniuses, Lempel and Ziv, and that's where the initials "LZ" came from, back in the '70s, I think it was '77, acquired a patent on a concept they had which became named Lempel-Ziv compression, LZ compression. And that's where ZIP came from, the Z in ZIP and GZIP and LZW and LZA and so forth. All of these are descendents of this original concept. And the concept is as follows: You are sending something to the other side that the other size knows nothing about, has no knowledge of it. And in fact you don't either. This is called "stream compression" because the idea is that you are receiving bytes in a stream, knowing nothing about it in advance.

So what you do is, as you receive bytes of data, if you can do nothing more, and I'll explain what that means in a second, you just send the byte on. But you also keep a buffer of the most recent X number of bytes you have sent. That is, you send them on, and you put them into a buffer that you maintain of what you've just sent. And as you receive them, you look upstream a little bit to see what's coming, and you look for matching patterns in what you've already sent.

And the genius of what these guys came up with was the idea that, if you were getting ready to send something that you had not that long ago sent, even a piece, even three or four characters, or five or six, you could instead send a reference to that string that you have in your buffer to the other side. The other side is doing the same thing you're doing. As it's receiving characters from you, it's maintaining a buffer. Separate from storing these characters in the destination file, it's maintaining a buffer.

And it turns out, and this was the cool part, these two buffers are synchronized. You're maintaining a buffer of what you've sent. It's maintaining a buffer of what it's received, which is what you've sent. So when you send it a reference to the buffer, to a substring that appears in your buffer, that's a reference to the same substring in its buffer. The point is you've only had to send a reference, not the actual substring. You've got compression. So that's the way compression works. All of this LZ compression is based on that, the idea of synchronized buffers which both ends are able to maintain, never having to actually share the buffer, but evolving it on the fly.

So that set of buffers is known as the "compression context." That is, it's the context that

each end maintains, one while compressing the other to decompress what the other has compressed in a communications channel. And the same thing happens if you, like, store a file on a hard drive and so forth. If all of these connections were being made separately, and if you even had GZIP, for example, dynamically doing communication stream compression on the connections, they would all have separate contexts.

The genius of what the Google guys did is to use one compression context per connection - not per stream, per connection. And what that means is you instantly get cross-query compression. That is, an HTTP query goes out in the first stream and is compressed. It won't compress much because we haven't seen those request headers before. But the second query goes from the same browser to the same server with the same cookies, the same user agent, all the same redundant headers, it goes out over the same connection, using the same compression context as the connections, which is the same compression context that just got through compressing the headers for the first stream. And all it is, is pointers into that context. It gets the most perfect compression you could get for free. And the same thing happens at the other end.

So the beauty is you're not redefining the HTTP textual protocol, where it's got headers and cookies and user agents and all that. All that stays. But the beauty is, essentially, you send it once, and then all you're ever sending again is very short pointers into the compression context that was established by the first query that went. You could still change things if you wanted. You'll get slightly less compression. But you'll be getting as much compression as you possibly could because you have one compression context for all the simultaneous multiplexed streams. And just because I didn't say it yet, to make sure it's clear, when the server is sending a whole bunch of things back, it's able to interleave these if it wants to. It's able to send chunks of different stuff, each with their own stream ID. They are received at the other end, and they're demultiplexed back into the original assets that requested by stream ID, and the client knows what to ask for.

So that is it. That is HTTP/2. It's worth noting also a couple things. First of all, that the speculative push is fun because it means that the server can anticipate what the client might ask for. Right now, in HTTP/1, it's not possible. First of all, there's no way for the server to send anything that hasn't been requested. But notice that that's always been something lacking. That is, the server is sending a page whose contents it has. It has to wait for it to go to the other end for the client to get it, then for the client to read it, and the client to request the things that the server already knew was on the page that it was sending to the client. So the server really does have more early knowledge of what's on the page than the client has.

Now, caching in the client prevents it from having to ask for things a lot more. So the server just wouldn't want to send everything on the page because the client may very well have a lot of that stuff already cached and know that it hasn't expired or just ask if it is expired, depending upon how long that object said it would be good for. So you don't want to overuse speculative push. But, for example, if there was a period of time where the connection wasn't in use, and the server believed it knew other things that the client might want, it could use that time in order to send them.

It's worth mentioning that, while we've got this great protocol, this does put a substantial obligation on the client and the server to make maximum use of it. That is, yes, now we can ask for things all at once. We can, like, try to figure out what the optimal priority and interdependence is, but that's not for free. That requires a kind of logic that clients and servers haven't needed to have built into them yet. So it may well be that what we're going to see over time is an improvement in the performance of HTTP/2 as each end gets better and more clever at using to its maximum capacity and capability the features, the rich feature set that HTTP/2 for the first time ever makes available to each end. And

that's what it is. And full credit to Google for this. This is a beautiful piece of work.

Leo: I suspect that a lot of this is in response to the capabilities of modern servers, in particular servers running Node and other Ajaxy solutions, where they really would love to be able to do this. You know, Google basically created AJAX 10 years ago with Google Maps and Gmail, the idea that stuff would load in the background so that when you slid a map around, those tiles would already be there. And but of course modern server technology of the time didn't really support it very well. And I think a lot of this is really in response to what servers can do and what sites using AJAX technologies have wanted to do. And now it's in the server; right? In the client, yeah.

Steve: That's certainly part of it. Mostly, it's just this connection burden. Now we have...

Leo: Yeah, eliminating that, yeah.

Steve: Yeah, we can multiplex now much more data over a single connection. And we already saw, remember that httpvshttps website where, side by side, you run SPDY with encryption compared to HTTP without. And is it any surprise, now that we know what this does, is it any surprise...

Leo: It's a lot faster.

Steve: This just blows its socks off.

Leo: Yeah, yeah, yeah.

Steve: Yeah.

Leo: That was a fun thing. We did that a couple of weeks ago, yeah.

Steve: Yup.

Leo: Steve Gibson does it again, ladies and gentlemen. I hope you've been listening. Yeah, I hope you learned something. We do Security Now! every Tuesday, 1:00 p.m. Pacific, 4:00 p.m. Eastern time. That would be 2100 UTC. You are certainly invited to listen live. I think it's a lot of fun to listen live, especially if you're in the chatroom and you can chat along behind the scenes and help each other...

Steve: Interact.

Leo: ...understand what's going on. But if you can't, hey, don't worry, we've got on-demand versions available for you. Steve hosts a 16Kb audio file, MP3 file that's pretty light, pretty small, for the bandwidth-impaired. He also has transcripts, which are excellent if you like to read along. Or if you want to search, that's the real value of it. You can search the topic and find the part you want. All of that's at GRC.com. That's where you should go if you have questions because I think, technology news permitting, we're going to have a question-and-answer session next week.

Steve: Q&A is on the schedule for next week.

Leo: So the best way to ask questions, don't email Steve: GRC.com/feedback for the feedback form. Or if you can do it in 140 characters, tweet @SGgrc. He follows the Twitter. You can also, when you get there, get SpinRite, the world's best hard drive maintenance and recovery utility. You can also get a lot of other freebies, those things Steve gives away, they're awfully good: GRC.com. Our website for this show is TWiT.tv/sn for Security Now!. That's where we put the high-quality audio and the video files of the show. Show notes, too, other things, and all of the other shows we do on the TWiT Network. You can also go to YouTube.com/securitynow. We have a copy there suitable for sharing with others. And of course wherever you get your podcasts because we're on everywhere. Being as we've done nearly 500 episodes. By now they've figured out we exist. Don't forget to vote for Steve at the Podcast Awards. You didn't mention it.

Steve: Oh, I didn't. Thank you, Leo.

Leo: PodcastAwards.com? Is that right?

Steve: Yup.

Leo: Did you get nominated? We don't know yet.

Steve: Don't know.

Leo: Okay. So right now you're voting to get him nominated. And then I'll expect you to vote to make him the winner.

Steve: Please, I would love that. It would be fun.

Leo: Steve cares about these things, so we want to support him.

Steve: I do. Make me happy.

Leo: Make him happy. He deserves to be happy. We'll see you next Tuesday. Thank you, Steve.

Steve: Thanks, Leo. Talk to you then.