SECURITY NOW!

Transcript of Episode #453

Certificate Revocation Part 1

Description: After catching up with the week's security events, Leo and I examine the history and operation of security certificate revocation and attempt to answer the question: What do we do when good certificates go bad?

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-453.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-453-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson's here. The first zero-day exploit on XP is here. We'll talk about mitigation. And then we're going to dive deep in certificates, how SSL/TLS works, how the certificate system works, and in particular the problem with certificate revocation. It's coming up next. It's a propeller-head episode, next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 453, recorded April 29, 2014: Certificate Revocation Part 1.

It's time for Security Now!. Get ready to protect your loved ones and yourself online with the Explainer in Chief himself, your very relatable, highly geeky, Steven "Tiberius" Gibson. Hi, Steve.

Steve Gibson: I got a bizarre email out of the blue from Roger Wagner. And I don't think there was any connection with the fact that I was talking about him in context when we were doing our - actually, I think it was in our second, maybe it was in our first, Triangulation. But he and Woz were together, he wrote, last Thursday. And I don't know what the occasion was. But they were talking about the hang-gliding trip to Mexico that I referred to. And what I forgot, which they both remembered and were laughing about, was that, when we were done with our day of hang-gliding down in Mexico, we were stuck in the border traffic, trying to get back into the states. And in the same sort of mood that I sometimes get into, like when I was dancing with the cardboard cutout of James T. Kirk, I got out of the bus, and along with all of the Mexican vendors who were trying to sell piatas and roses and so forth, I started selling light pens to the...

Leo: Get your light pens here. Get...

Steve: ...to the people at the border.

Leo: That's funny because you have to have an Apple II for them to be of any use. Am I right?

Steve: Well, yeah. I didn't sell any. But...

Leo: Did you have a little box that you had hung around your neck?

Steve: I don't even know why I had them with me. But apparently I had some.

Leo: What's your sales pitch for the light pen?

Steve: Everyone on the bus got a big kick out of it, which was of course the point.

Leo: That is hysterical. That sounds like fun.

Steve: Ah, yeah.

Leo: Well? You're...

Steve: So we have a not-crazy week, which is nice, for a change, although we have some significant news. We're going to talk about the first not-going-to-be-patched zero-day flaw found affecting IE. We've got another critical update from Adobe that they're pushing out; a new version of Firefox. We're up to 29 now. Some interesting news of a bunch of major industry heavyweights jumping in to fund important open source projects for the good of the Internet, and also just a little blurb about some routers that come with DD-WRT, the firmware that we were recommending, that way from the factory. So good news.

And then we're going to talk about - this is the first of two podcasts, as I promised last week, which I think people are going to find really interesting, talking about revocation, that is, the certificate revocation problem. We're going to go back and look at basically how it all works from a historical and evolutionary perspective, getting up to where we are now, what is probably going to happen in the future, and sort of basically understand that we built a system where we're really good about determining what to trust. But then we also need to say, uh, but wait a minute, we need some exceptions. And so how do we revoke that trust in a reliable fashion, after we've gone to all this trouble of creating it in the first place? But really, as we were saying before we began, before you hit the record button, Leo, you don't really have trust unless you're able to revoke it when you've changed your mind.

Leo: It's meaningless to say "trust me" if you can't say "don't trust me."

Steve: Precisely.

Leo: It has no meaning.

Steve: Precisely. So this week we're going to cover the mechanisms and the theory. And then we'll follow up next week with our look at what specific browsers and operating system platforms do in this regard. And it turns out no two of them are the same. And it's really confusing because they interact with each other. So, I mean, it's like, in fact, in some cases I won't be able to be definitive next week because it's just hard to know exactly what's happening. But there's lots of interesting specifics that we will talk about. So we have a couple great podcasts, first one today.

Leo: All right, Steve. Let's get to the security news of the week.

Steve: Yeah. So with having passed the end of XP service, we're all waiting to see what's going to happen. And what was found by the guys at FireEye was an attack, a targeted attack using IE versions 9 through 11 - by the way, none of which run on XP. But the vulnerability is present from IE6 through 11. So basically, like, IE6 goes way back. So all versions of IE, essentially, although at the moment the targeted attacks don't affect XP.

So here the expectation is that Microsoft will not be fixing IE8, which was the last Internet Explorer that runs on XP. Still, my contention has been that we'll have to see what develops in the fullness of time. This is not a vulnerability in XP, but yet again another vulnerability in IE. And of course, as I've always said, Internet Explorer has always been sketchy to use. Firefox and Chrome, in terms of vulnerabilities, are probably a better place to be.

So, many people were surprised when I tweeted that I had just blogged about this yesterday because what I got back was a lot of feedback saying, "You have a blog?" And it's like, yes. There are situations where I have something to say that won't fit in a tweet, but it's just - it's transient, or it doesn't make sense to create a web page. And of course I guess the point is that, since I have GRC.com, most of the serious anchor things I do are on web pages. But there's a place for a blog, too. And it's steve.grc.com is my blog. And so what I wrote there, this was just two paragraphs of it, I said:

"Web browsers are growing insanely complex. It's pretty clear that they will be our next-generation operating platforms. And as the last annual 'Pwn2Own' contest showed" - which we of course covered on the show a few months ago - "none of them can currently withstand the focused attention of skilled and determined attackers, especially when some prize money is dangled on the other side of the finish line.

"With most recent exploits, the path to exploitation is convoluted and complex. In this case, it depends upon somehow encountering malicious web content with IE's ActiveScripting enabled, which loads an Adobe Shockwave Flash file, which in turn uses JavaScript in this vulnerable version of IE," which is presently all versions of IE, in order to leverage a deprecated standard, VML, which is the Vector Markup Language. It's been disabled from IE10 on because we've replaced it in the industry with HTML5. So Scalable Vector Graphics is there, and HTML5 sort of put the final nail in its coffin.

So, but, again, attackers are very clever, and they figured out how to do this. This FireEye guys have very, very detailed coverage of this and explain how Shockwave Flash is used with some known attacks that are characteristic to set the stack up in a way that then the vulnerability in this Vector Markup Language, which is still hanging around, is able to be exploited.

So the reason I made a blog entry for this is that Microsoft has a remediation that is effective right now. But I've not been able to get it to work under 64-bit Windows. It worked perfectly for me on XP. And so I needed a place where I could quickly explain it and give this to people who were interested. And it's just a crazy deregister VML from the registry so that it's not available to be invoked by IE. And so you can, if you're using XP, 32-bit XP, you can simply do this, which is explained at steve.grc.com, and the problem is solved.

And in fact I also provide a test link there where, before you do it, you can click the link, and it brings up a Vector Markup Language floor plan in IE. And it's like, oh, look. I mean, you have to - assuming that you are using IE, I mean, if you have Firefox as your default URL handler, you'll have to copy this into IE in order to make IE render the floor plan. And it will. Then you run this, restart IE, and it will no longer run it. So that's your confirmation that you're no longer vulnerable on this version of IE which we now believe Microsoft is not going to patch. I cannot get the same thing to work. Yes, now, see, what you're showing on the screen there was where it said a VML-capable browser is required to display this image.

Leo: This is in Chrome on Mac, so...

Steve: Which is not surprising. So Chrome has probably never supported VML. I get the same thing under Firefox. Now, just so people don't misunderstand, over on Win 7/64, which was the other platform I tested, I initially got that. But this is in IE 11. And I knew that IE 11 had deprecated VML. So under the menu I went to compatibility mode, added that domain to compatibility mode. Then I got the floor plan. So you're able to get, I mean, it's still there. And so the attack must involve getting around that. And knowing Microsoft - and again, I'm at just the beginning of looking at this. I had to work all day or this morning so far on the podcast, so I couldn't dig any deeper. I plan to go further this afternoon, and I will probably be updating that same blog entry, unless I just decide to scrap it and start over and do another one once I get this untangled because there are typically headers you can put in web pages to invoke these old modes that IE uniquely understands. And so that's probably part of the attack.

So I may end up quickly throwing together a web page with some simple VML and this invocation, just to allow people to verify that they've been able to make themselves safe. But now that I've got this thing under Win 7 rendering the floor plan, I can't get it to stop rendering it. Microsoft's own advice isn't working over there. Works great under XP. So for 32-bit XP people, we have an answer. And probably later this afternoon, once I have had a chance to look at this a little closer, we'll have more.

So anyway, so it is targeted. It's in the wild. It's in use. We don't know if Microsoft is going to produce an out-of-cycle patch or what. We have good remediation. Really, nobody needs VML. It's no longer being used. It's not even available in IE10 and 11 unless you somehow explicitly go back and sort of turn on something that has been deprecated by the industry, yet it's still there. So, and I thought, well, how about just renaming the DLL because it's in two places. There's a 32-bit version and a 64-bit version. It's vgx.dll. But Windows is protecting me from myself, and I don't have the privileges to rename the file. So it's like, okay, well, that won't work. Anyway, I'll figure it out. And I'll have something updated on my blog. So, yes, I have a blog. It's very seldom used. I think it was August of some other year that I - actually my prior posting was about Ladar, and it was...

Leo: Oh, yeah, yeah, August last year, yeah.

Steve: Yeah.

Leo: That's not bad.

Steve: Yeah. So it makes sense. For some things it's perfect. When I want to do something quickly, get something out that won't fit in Twitter, it's not worth doing a page at GRC, so it's on my blog.

Leo: Steve.grc.com.

Steve: Right. And all that is, is that bounces you over to WordPress. It's just a WordPress-hosted blog, so nothing special. But, you know, handy. And it does allow people to comment. And there is a whole comment thread off of the blog right now, about 27 responses I think. Some people reporting success, some confused. And I've responded a couple times saying, okay, I'm going to figure this out, and we'll come up with some answers soon as we can.

Leo: Good, good.

Steve: So, new Firefox. Happened yesterday. We went from version 28 to 29. And we got the somewhat controversial, what they call their "new streamlined look," which everyone else says, wow, this looks a lot like Google Chrome. And in fact some people don't like it enough that there is also an add-on that you can get, the Classic Theme Restorer, in order to bring it back to something that you're more comfortable with. I guess I have enough tweaks that I've already made to mine. It didn't really change mine. The back button is now kind of big and round. But otherwise I still have my crazy list of tabs down the right-hand side instead of across the top, which is the default. And I've got the menu line with File, starting at File and ending in Help. So all of that was preserved as I went from 28 to 29.

There weren't a huge number of improvements. They've got a new version of Firefox Sync, which provides end-to-end encryption, so good encryption for synchronizing instances of your Firefox on different platforms. They have a new customization mechanism that makes it highly customizable. There's sort of a new walkthrough that introduces users to the new features. And otherwise they've just - they've continued to push and move forward with HTML and CSS standards. So they're, like, the -moz suffixes have been dropped from things where they no longer make any sense. So they're just sort of moving forward.

So it makes sense to stay current. But if you don't like the changes that have been made in terms of a UI look, you can get all of the security features and enhancements and stay with the latest Firefox, yet use the so-called Classic Theme Restorer, and you can find it under that name, in order to stay with an older look, if you prefer that.

Over the weekend another zero-day critical flaw was found. So Adobe has a multiplatform patch. IE10 and 11 will update. Chrome will update for you. But if you're running older versions of IE and/or Firefox, you may - actually I guess Firefox is supposed to be checking now and prevent you from using old versions. So there is a new one, and you can just go to download.adobe.com, I think. But be careful because it's trying, I think right now it's trying to give you McAfee scan or something, if you don't explicitly uncheck that box. So make sure you don't get junk installed that you wish hadn't been. It's interesting how easy it is just to overlook the checkbox that you wish you'd remembered to turn off.

And I did, after talking last week about the return of our port 32764 problem and how Ethernet can enable an Internet backdoor, and how it just really seems at this point that, if you can use a router that's got firmware with a known history and set of goals, and basically some trusted providers, it makes more sense. We've talked about DD-WRT, and of course another very popular firmware is the Tomato firmware. And several people sent back that Buffalo Tech ships their routers with DD-WRT out of the box. And someone said that his n600 dual band he's very happy with. And so I just wanted to pass that on. I did get some notes back from people saying, hey, well, how do I do that? Where can I get a router that's got that? So it's very often the case that you can just go to eBay and get an old, I can't remember, it's 54 - that Linksys...

Leo: Yeah, the DD, what is it, WRT54G, I think, is the one that's best.

Steve: That sounds right.

Leo: But, well, go ahead, keep going because I know what you're going to...

Steve: Well, anyway, and so you can often find an old refurbed one for very inexpensive.

Leo: Right. We have tons in the basement. We have, like, six of them in the basement. I don't know why. Well, they all have - because they all have DD-WRT on them.

Steve: Yeah.

Leo: The WRT54GS2.

Steve: You mean that you have tons operating in the basement, running. You're actually...

Leo: You know, I don't know how many of them are in use. We put - we were using them. And I think Russell took them out of circulation. I believe, I'm not sure. But, yeah, no, we have four or five 54Gs running DD-WRT.

Steve: Yeah, for a long time that was the insider's router.

Leo: Absolutely.

Steve: That was what you did.

Leo: Absolutely.

Steve: So in a very nice move - responding essentially to sort of a wakeup call is how I'm thinking of the Heartbleed incident. It shocked everyone. And I think it was, when the future Internet history is written, it's going to stand out as a moment when sort of the complacency that we had drifted into was shaken out of us because the Linux Foundation announced the start of a three-year initiative which has already got commitments for $3.9 million over the course of three years, specifically to fund underfunded open source projects which are considered to be important to the health and future of the Internet. So Amazon, Cisco, Dell, Facebook, Fujitsu, Google, IBM, Intel, Microsoft, NetApp, Qualcomm, Rackspace, and VMware have all stepped up and pledged to commit $100,000 each per year for at least three years to the Linux Foundation's new, what they call "core infrastructure" initiative.

And apparently the companies were really willing. I mean, when Linux came knocking, when the Linux Foundation came knocking and said, hey, would you, they said, oh, my god, we're so glad you asked. We're sorry it didn't occur to us to do this sooner. And what was really interesting, the counterpoint to this is that the guy who's sort of been tending OpenSSL previously, that project, commented that, yeah, we've been getting about $2,000 a year. So, I mean, it had really - OpenSSL had been on a shoestring, basically unpaid, for all of this. And then everyone realized, wait a minute, we're all using it, and no one's paying for it. We just sort of assumed, hey, look, there it is, it works, we'll use it. Yay open source. And so now it's really going to get some money.

Now, this doesn't all go to OpenSSL. The foundation will be identifying a number of worthy open source projects, of which OpenSSL is the first, but not the last. So this is really a great outcome for this. And it probably took something like this, I mean, of this sort of breathtaking scope to really wake everyone up and say, wait a minute. Everybody, I mean, is using it.

I ought to say, however, that Microsoft's not using it. Yet they're stepping up and going to contribute their $100,000 a year, too. So even companies that aren't directly using this are saying, hey, you know, we want to support this. And I do think I remember that Microsoft's Internet stack got changed in the past. They didn't have a very mature one. And I don't remember now what version of Windows. It might have been when they went to Windows 2000. Everyone looked at it and thought, wow, where did this come from? And there was some speculation at the time that this came from one of the BSD Unixes, that Microsoft "borrowed" the open source code and made it their own.

So in that sense, if that was the case, because I do remember that at the time it was like, you know, you just don't come up with a mature, incredibly well-behaved TCP/IP stack out of nowhere. And it was clear that they weren't using the one they had. And there were some characteristics of it that the people who were looking at it closely at the time thought, eh, this kind of is behaving suspiciously like, I don't remember now, OpenBSD or FreeBSD or one of them. So anyway, it's clear that, as an industry, we get a huge amount of benefit from all of the basically volunteer work that the open source projects do. I think it's great that they'll be able to get funding and tackle some of the needs that they have.

I want to share a nice note. I try to find, when I talk about SpinRite, different ways it's being used, sort of that people may not have thought of before. And I found another one. A guy named Gary Foard, F-o-a-r-d, who's in England at TheBroadbandEngineer.co.uk, he sent a note on the 12th of this month just headlined or titled with the subject "SpinRite testimonial." He said, "Hi Steve. Long-time listener, et cetera. Just thought I'd drop you a note that you might like. I get pleasure from keeping old computers working and appreciate basic but good programs that have a purpose and do a good job. So I was very happy when I saw how SpinRite worked and behaved. I use standard Ubuntu nearly everywhere now," and he says, "since Windows ME ran out, and was then relying on Firefox to keep me safe." He says, "I installed a full working Ubuntu OS on a USB thumb drive, not a live install ISO, so I have a fully patched computer rattling around in my pocket on my key ring. However, once in a while it becomes slow. So I get an old 256K RAM Compaq Presario 700, which was otherwise only used to practice Linux server commands on, boot SpinRite, and run SpinRite against that plugged-in USB drive. I have to run it on Level 3 to do any good, but the difference is unbelievable. So well done, Steve, for giving a very old computer a practical use, and keeping my portable Ubuntu disk running sweet."

Leo: Why do you think that works?

Steve: "Regards, Gary Foard." Well, it's interesting. We talk about Level 2, which is read-only, and Level 4, which is the more grueling process where SpinRite reads the data, inverts it, writes it, reads it, inverts it, writes it, and then reads it, so that it double inverts it. It comes back the way it was, but it's just given it some real exercise. Three is a lighter version. It's like Level 4 Lite. And I designed it just to do a refresh. So it just reads and writes. And so that's - probably it's gentler on a USB drive, or let's just say on a solid-state drive, certainly than Level 4. Yet in his case, just doing a standard read doesn't, like, shake the dust out of it. So it does a read and a write. So essentially, probably what's happening is the drive is getting soft read errors, so it's having to perform error correction, which it may not be able to do very quickly. But by reading it and then rewriting it, essentially he's writing it strongly again. Remember that, essentially, solid-state drives are like little capacitors. They're little...

Leo: I wouldn't think they'd have the same problems magnetic drives would, as for that.

Steve: Correct. And so what happens is, if these little capacitors are tending to bleed their charge, you'll begin to get soft errors as they're no longer reliably read. And so that is how SpinRite fixes solid-state drives is by rewriting them before they become completely unreadable, it's able essentially to recharge the little capacitors in the memory which are fading over time.

Leo: Yeah, because usually when solid-state drives get slow in my experience it's because of trim not being supported.

Steve: Oh, and that's a different instance.

Leo: That's something else.

Steve: Because there what's happening with trim is that solid-state drives can only write in large blocks. And so if you modify one sector, for example, in eight, it's forced to read all eight, change the data in one, and then write back all eight because it only operates in blocks. Trim is a way for the operating system to say, I'm only actually interested in this one sector, so don't bother dealing with the whole block. Only fix this one. And so it provides additional information to the drive, allowing it to operate more quickly. And so this is different from that. This is actual bits softening over time. And just by having SpinRite read and rewrite it...

Leo: That happens on SSDs.

Steve: Yep.

Leo: Huh.

Steve: Yup. And we've covered SpinRite actually recovering SSDs. It's the reason I'm convinced now it really has many, many, many years of additional life is it's fixing SSDs for people.

Leo: Interesting. Well, let us - are you ready? Because I don't...

Steve: Yeah.

Leo: I don't have any more ads. So let's get right into it.

Steve: Okay. So first a little bit of a review on how revocation, well, on how trust works, and then we'll talk about how untrusting works. So trust is based - and this is - it's a very clever system. It's known sort of generically by the acronym PKI, the Public Key Infrastructure, which underpins a lot of the way the Internet works, and probably more of it in the future because it's turned out to be a handy and a robust technology. So, and we've talked about this in bits and pieces, but I'll sort of give a quick overview so we have a foundation.

The idea is that we want to connect to a server whose identity we can verify. We need to verify its identity because there are bad things that attackers can do that could cause us to go to the wrong server. We've talked about ARP poisoning in the past, a way of essentially getting your traffic filtered, where your traffic could be redirected somewhere; the problem with DNS poisoning or router poisoning. For the infrastructure, the whole Internet to work correctly, all these pieces have to be functioning correctly. And there's been a lot of attention focused to hardening them over time so that we can more reliably trust them. But ultimately, the best test is for the machine we eventually connect to through DNS giving us the IP, and then routers getting us to that correct IP. We want that machine to be able to provide us an assertion of its identity that we can verify. So that's the security certificate.

What happened is it proved its identity to a third party, to a so-called certificate authority. And the certificate authority made it jump through whatever hoops were necessary so that the certificate authority was convinced who they were. Now, there are grades of that. For example, the weakest kind is called a DV, a Domain Validation. And typically all you need to do is prove control of a domain. So, for example, if you've got - and that's a low level of proof. So to get a domain validation, the certificate authority just emails, like to webmaster@grc.com, and I just prove by clicking a link in the email that I'm able to receive email at the GRC.com domain. And then they go, okay, well, he's in control of it. And it's interesting, too, because if, like, for some reason you're unable to receive email, they'll give you something that you can put in a page on your server, and then they'll go look at a page on your server at GRC.com or whatever domain you're registering. And again, you have proven you have control of that domain.

So anyway, so the idea is that one way or another you prove whatever level of guarantee is being offered. For the so-called EV, the Extended Validation certificate, they put you through much more. They'll call the phones. They'll try to find Dun & Bradstreet records that verify your address. They may phone your business number during business hours, send you a fax to the fax number, check your WHOIS, and then use the phone number that's in your Internet WHOIS records for the domain and so forth. So generally a higher level of proof to develop a higher level of trust. So ultimately, though, once that's done, this certificate is signed. And essentially what's being signed is your public key.

So the server generates a public key pair, keeps the private key to itself, and says to this authority, here's my public key and some identifying information. Please sign this. Now that I've proven to you who I am, your signature essentially asserts that. And that's the other point. The proper terminology here is "binding." That certificate contains the domain that the certificate is for, other organizational information like the name of the company, the city and state and province and so forth, depending upon the country they're in. All of those things and the public key are collectively signed.

So that creates a security binding, binding all those things together. So that is what it sends us, the web user, when we connect to this machine. We're hoping we're connecting to the right place. We're hoping that DNS wasn't poisoned, our hosts file hasn't been screwed with, that routers between here and there have all taken us to the right place. Ultimately, though, we get the certificate from the server we're connecting to, and it contains all that information. We verify that the URL that is in our browser matches one of the URLs in the certificate. Certificates can have wildcards in some cases where it'll say *.grc.com, in which case it's any subdomain of GRC.com. Or it can enumerate specific ones.

Some certificates are horrifying. They've got, like, hundreds of domains that they're valid for. I only say it's horrifying because it just seems a little scary, in the same way that trusting too many certificate authorities seems a little scary, just from a - security wants to generally use the minimum solution that is sufficient.

So the point is that we get this assertion from this server at this IP address. And it's been signed by the certificate authority on behalf of that site. We contain, in our browser, we've got all of these certificates from the various certificate authorities which we use to verify the signature. It allows us to cryptographically check that the certificate provided by the site has not had a single bit changed from what the certificate authority signed, meaning that we can rely on those bindings of domain name, public key, and all the other organizational information that is contained in that certificate. So that's the way the system works. We trust the server because someone we trust, that is, the certificate authority, verified the server's identity, signed their name to that verification. Then the server has sent that to us, and we're able to verify the signature. So the system works.

So we've also talked about how there's a natural expiration to all of this. Typically two or three years - and I did notice, I encountered for the first time longer lived certificates. If you use a larger hash, the SHA-2 family rather than the SHA-1 hash, for example SHA-256, on some lower security certificates, like a domain validation, you're able to get beyond three. I think I saw six years. So those live a lot longer, though they have a strong - because the idea is you're going to trust them more because they have a hash that we believe is stronger, and ultimately we're relying on the cryptography bound into the system for lasting the length of the certificate. Otherwise these certificates expire. And we'll see that that's a good thing because that's part of the revocation system is we don't want these things to last forever.

Now, what all of the certificate authorities have begun doing is using an intermediate certificate, which people are probably seeing if they look at so-called "certificate chains." What I just described, where the server certificate is signed by the certificate authority, I didn't explicitly say it, but if the so-called "root certificate" had done the signing, then there would just be two nodes in this chain. The problem is that exposes the certificate authority's root crypto on a daily basis, sort of online, in real time. And actually, many years ago, that's the way things were done. But as we became sort of more aware of how we want these models to evolve, and as it was more convenient to have the root certificates last longer - now, for example, they're expiring 15, 20 or more years from now - the certificate authorities said, you know, we want to keep our root offline. We don't want it online where it's, I mean, and by "offline" I mean literally not electronically accessible, so that no kind of network penetration could put it at risk.

So what they've done is the certificate authority creates a so-called "intermediate certificate" which sits in the middle. It's the intermediate certificate which signs the end certificate, that is, the domain name, the web server certificate. And that solution allows them essentially to camp or park the absolute golden master certificate authority root in a safety deposit box somewhere, not electronically available. It doesn't need to be electronically available because it only needs to sign the intermediate certificate once. Then it can be completely taken offline, and the intermediate certificate is then used for all the bulk signing. The benefit and the point of that is, if anything did happen to the intermediate, that it somehow got loose or was compromised, then that's not a compromise of the root, and it doesn't completely - it doesn't create an upheaval of that certificate authority's complete infrastructure.

So today, if you look at any of the chains, you'll almost certainly see at least one intermediate, and sometimes more than one, depending upon sort of the provenance of the certificate, where it came from and who signed it. Sometimes, for example, there are some newer certificate authorities which will themselves be signed by an older certificate authority. That's the way they bootstrapped themselves into business. When a new certificate authority comes along, the web browsers don't know about it. But they want to be in business, so they get their certs signed by somebody who's been around for a long time, that all the servers trust. And that allows them to get themselves bootstrapped while they're working on getting their own root certificate to be trusted throughout the so-called PKI, the Public Key Infrastructure.

Okay. So certificates live for some number of years, at which point their own date expires them. So they're no longer going to be trusted. So the problem that the original guys designing this really cool infrastructure faced was what happens when a server that's got a certificate installed on its server - and that's one of the problems with this is that there is inherent exposure. I mean, and Heartbleed is a perfect example. The reason that there was this mass of revocations is that what we discovered, and several hackers confirmed, is that it was possible with the Heartbleed vulnerability to find a server's keys in RAM. And thus those keys had to be revoked. Those keys had to be canceled prior to expiration, and those companies were issued replacement keys.

But now we have a problem because, if bad guys got a certificate which is still valid until it expires - and that's going to be, on average, years. If we assume that there's no relationship between when the attack occurs and when this particular certificate was first minted, then a certificate, for example, with a three-year life - which is what these domain validation certificates typically have because you get a lower price if you buy it for three years, sort of the per-year discount. So that means on average it's going to have an 18-month remaining life. Some will be shorter; some will be longer. But on average, half that time is what we would expect.

So for 18 months now, bad guys have a certificate signed by an honest-to-god certificate authority that everybody trusts. We used to trust that certificate when it was being offered by the real server. Now the danger is bad guys could set up a clone server, if this was an Amazon certificate or eBay or PayPal or something. They would set up a clone website that was offering this certificate. And they would still have to somehow arrange to get traffic there. They'd have to divert traffic. So they'd have to poison someone's hosts file or poison DNS or somehow - we were just recently talking about how people's SOHO routers were getting their DNS redirected.

So what normally happens when you connect to a router at home is you're given its IP address, 192.168.0.1 or 100 or whatever it uses, you're given its IP as your DNS. And then your requests go to it, and then it forwards them, hopefully, to your ISP's DNS, or wherever it's configured. But if you were to redirect that, you would have no knowledge that you were going to a bogus DNS server. And so what would happen is your browser would say, would think it's going to www.amazon.com. It would ask for the IP. It would get the wrong IP from a bad DNS server, which would then take you to the attacker's server, which has been set up to look exactly like Amazon.com and to deliver Amazon.com's security certificate, which was stolen from the real site. You would have no reason to mistrust it. Your browser would look at the certificate and say, oh, yeah, look, it's got - it's signed by somebody that we're all trusting. It says www.amazon.com. That's the URL in the browser bar, and I'm assuming that that's the IP I'm at because DNS told me that. And so you would end up essentially on a fraudulent site with no reason to suspect this wasn't the case. The padlock would be closed. Things would be all shiny and bright and looking all secure. And this would be a fraudulent connection.

So clearly the guys that designed this system knew that's not good, that, if not for Heartbleed, certs are going to get revoked. And in fact some studies have been done because, when reasons are provided for why certificates were revoked, 44% of the time it's key compromise. And that is by far the most cited reason for certs being revoked is, for whatever reason, we don't know the details, but companies are believing that their keys are loose. Maybe they had a bad termination experience with somebody who might have had access to their keys, and they're worried, so they just say, oh, you know, we're going to say that our keys may have been compromised. Who knows? But we do know that it is the most often-cited reason. So if that's the case, then we really need to no longer trust this otherwise trustworthy key.

So essentially what we need is some sort of out-of-band solution. The in-band system is asserting trust. We need some way to verify that. So that's the creation of the so-called "certificate revocation list." The idea is that, when a certificate is revoked, and the company says to its certificate authority, whoops, we still have two years left on our cert, but we can't trust it any longer, we won't go into details, but we think we need a replacement. Most CAs have no problem with that, certificate authorities have no problem with that. You give them new keys. They will rekey and reissue your certificate for the balance of the lifetime that you had on the first one. So it's not a big deal.

What the certificate authority then does is they add the serial number of the previous and now-revoked certificate to a list that they publish. And the cool thing about the system is that, bound into the certificate that they issued, that is, they signed and returned to the server, is the URL of the list which, if that certificate were ever to be revoked, its serial number would appear on. So when you think about it, it's a cool, clever system. So the idea is that the company getting the certificate initially sends them their private key, their assertions about who they are and the domains they want covered. The certificate authority not only verifies all that, but includes in what they return additional fields. And among them is the - it's called the CRL distribution point. The certificate system has the ability to have extensions added to it, that is, basically just additional information. And we're going to be talking about additional information on certificates in a few different contexts here in the next two weeks.

So in this case it's just sort of an additional field which is defined in the industry that all of the systems know how to read, which is the URL of the file, the CRL, the Certificate Revocation List, which, if this certificate ever does need to get revoked for whatever reason, its serial number will be added to that list. So what that means is, when we receive, we the Internet surfer wanting to trust the server where we're visiting, when we receive the certificate, one of the things that we can do, and we will do if we're concerned about revocation, is we will look at that certificate's CRL distribution point's entry, get the URL, query the URL which is being offered by the certificate authority, and quickly, hopefully quickly, scan down this list looking for our serial number, to verify it's not there.

Okay, now, there are some problems with this, which is where the story gets really interesting because notice that the whole system now is we trust by default. Now we're trying to override that trust. And even then we're needing to get a file that's listed in the certificate that may contain the serial number warning us not to trust the certificate that's provided all this to us. Now, it's not a problem that it's provided it to us. It can't change that URL. Remember, this is all - this is what the hash protects is it protects the contents of that certificate containing the URL. Nothing can change that without breaking the signature on the certificate. So if bad guys are trying to use that certificate, then they need somehow to prevent us from being able to find the serial number which may be listed on the CRL, the Certificate Revocation List published by the certificate authority. So there we get into the what's needed to defeat the system, which is clearly an important part of this.

So this is the way the system was originally built. Over time, as we got increasingly busy, as the Internet exploded - and this is a Security Now! podcast. We talk about, I mean, we spend a lot of time talking about the growing use of SSL. Remember that none of this comes into play with regular HTTP connections, none. There is no protection at all, that is, against DNS poisoning and router hijacking and so forth. Unless we have a secure connection, unless we get an assertion of the identity of the far point we're connected to, we could be connecting to anybody. So thus the reason that in recent years we've really been pushing for SSL everywhere. It's only with SSL, now called TLS, that we're able to rely on a lot of these assertions, which are sort of otherwise implicit. This makes them more testable.

So imagine a system where normally you've got some churn in the CRL. That is, over the course of decades, certificates are being issued. Certificates are being revoked. They're going onto the CRL, but they don't have to stay there forever. Remember that the date will expire the certificate also. So as certificates are being added to the certificate revocation list, they only need to stay around until they expire, and maybe with some fudge factor because we want to make sure that the clock, the calendar of the person doing the testing is correct because they're going to look at the certificate. And if it's out of date, then they're not even going to bother checking the CRL, which that's the point. That's why it's not going to be on the CRL well after its expiration. But if it's almost expired, and their clock is set wrong, and it had been removed from the CRL exactly on expiration, then they might look at the CRL, thinking that it's not yet expired, and trust it when they shouldn't. So they normally stay on the certificate revocation list for some length of time after expiration, just to make sure we don't have that boundary condition uncertainty around expiration.

But the point is that the CRLs are not growing forever infinitely because certificates do end up expiring off the end of them. At the same time, the life of certificates has been relatively stable. Although, as I mentioned, the six-year certificates, well, that's going to tend to keep these revoked certificates around on CRLs longer because they're going to have a much longer life. But even notwithstanding this recent increase in maximum certificate lifetime, as we get busier, we're issuing certificates more often. Yet we're not shortening their lifetime. We're leaving it pretty much the same.

Well, if you think about it, that means that - and if revocation is as a percentage of total certificates issued, sort of staying around the same, but we're issuing more, we're going to be revoking more. Which means we're putting them on the list at an increasing pace, yet they're expiring from the list at a fixed pace. Which means the lists grow. And that's what's happened over time is these certificate revocation lists have been getting bigger and bigger and bigger. So that creates some problems, just practical problems because, remember, the way this is set up now, this is wired into us clicking on a link and first seeing a page. Before we are given the page we want, all of that has to happen in the background.

We have to establish a connection to the server. It's got to give us a certificate. But before we present anything to the user, we've got to make sure we can trust that certificate, which may require a fetch of the Certificate Revocation List listed on the certificate. Well, if that thing is huge, just in terms of just raw bandwidth and data transfer time, it's going to delay us getting the list and create some overhead in making sure that our certificate is not listed anywhere on the list before we can first give this to the user.

Now, one of the mitigating factors here is caching. And caching has been incorporated throughout this in order to, again, to create a tradeoff. Certificate revocation lists are generally published daily. So most CAs create a new CRL every day. And they give it a one-week expiration, meaning that they're saying this is good for a week. If you've got one that is expired, that is, if you have a certification revocation list which is expired, you ought to get a new one. But every day they make new ones available.

So here we have some tradeoff in security because obviously this creates a theoretical window of opportunity. If the user was going to Amazon.com and checking Amazon's security certificate against its CRL, the CLR listed in the Amazon.com cert, then they would be pulling the CRL from Amazon's authority, the signed Amazon certificate. But they'd only be doing it if it needed updating, that is, if it was more than typically a week old. Which means in general it's going to be, what, three and a half days. It's going to be typically half that age on average.

So in theory, if the bad guys were really quick, if they could set all this up, get the certificate, arrange their fraudulent site and traffic redirect and everything, and it happened that they fooled somebody who had picked up a copy of Amazon's certificate revocation list that hadn't yet expired, then even though the browser said, oh, I already got a copy of that, I don't need to go fetch it because it's in my CRL cache, the browser would look through it, see no problem with this revoked certificate, which is actually on the current CRL being published by Amazon's certificate authority, but it's not in the copy in the cache that the browser has because they're allowed to be up to seven days old before they're refetched.

So that creates, again, a tradeoff in security which the industry has felt was the best they could do for a long time. Until we went to the next step, which is called OCSP, which is the evolution of the certificate revocation list model, which solves the problem. It's worth also noting that one of the inefficiencies in the certificate revocation list model is that we're getting a huge list, in the case of this certificate revocation list growth over time, only to check for the possible presence of one serial number out of however many are on the list. So it's sort of fundamentally inefficient. It made more sense in the old days when there were fewer certificate authorities, and the whole PKI infrastructure was much smaller because then the cool thing was you might go to Amazon and download its CA's CRL - well, assuming that Amazon existed back then. You'd go to something in the early days of the Internet that was...

Leo: Zombo.com. That's where you'd go.

Steve: Exactly. And then you'd go somewhere else that was using the same CA. That is, their certificate was signed by the same certificate authority. And it's like, oh, look, I already have that. Not because you'd ever been to that site before, but because you'd been to a different site that shared certificate authorities. So when there were only a handful, it was an elegant system because the idea was it was a way of distributing these lists to users incrementally, maintaining them over time, but they'd all get shared by all the different sites you'd go to.

The problem now is there's been an explosion of certificate authorities, as we've talked about, hundreds of them, and of course the oft-maligned, and undeservedly so, Hong Kong Post Office. We had fun with that years ago when I discovered that my browser was trusting things that the Hong Kong Post Office had signed. It was like, wait a minute. I mean, there were 400 certificate authorities back then. And so the point is that that original concept has lost some of its value as everyone said, hey, we can make money selling certificates. Let's get in the business. And now there's many more than there were in the beginning. So that efficiency is lost. And the certificate lists have exploded. So then there's this overhead associated with getting these big lists downloaded.

So what was created, the next generation, the solution to this, which is also a decade ago, I mean, it's been around quite a while, I think it was in '99, so, what 15 years ago, they said, okay, we need something - we need to replace, we need an alternative to this certificate revocation list. And they came up with OCSP, the Online Certificate Status Protocol. And with all of the foundation we've just laid, with everything you've just heard, this will be a small step forward. You'll get this immediately because another extension was added to certificates. This one is called an AIA extension, which is the abbreviation for Authority Information Access. I don't know why they choose these obscure names because that doesn't tell me what it is, Authority Information. I mean, it doesn't say OCSP URL.

Leo: That would be good.

Steve: Which would be handy. Anyway, under the Authority Information Access extension - and anybody who's curious, by the way, you can see all these things. If you open up any browser, have it display the certificate, look at the details, scroll through the information, you'll see something that says CRL distribution points. And if you look in there, you'll find a URL that is the certificate parent's, the original signer's URL. Now you'll notice another one, Authority Information Access. That's a different URL for the certificate authority's online certificate status protocol server. And you can probably guess what it does.

The beauty of this is you're making a specific query to that server about this one certificate. Instead of saying, oh, look, give me the list, I don't know how big it's going to be, I hope it's not big, that might contain this serial number. Now you're saying I've got this serial number. What do you think? And so your browser is now making an explicit query back to the certificate authority, saying I'm about to trust this which you signed two years ago. Is it still good? So that's an elegant solution. It's saying, I mean, again, it's still trying to overcome the implicit trust in the system, so it's a little sloppy in that sense. That is, we're first trusting. Now we're being asked to back that trust off. So, but that's the only thing we've come up with so far that works.

There have been some proposals, for example, of very short-term certificates, that is, change the whole system so that, rather than having certificates last for multiple years, we have the certificates themselves last only for days. And then the idea would be that there would be a new system where the web server would constantly go back and update its certificate on the fly in near real-time so it's always providing a short lifetime certificate. Meaning that, if it ever stopped doing that, the certs that it has will die in a short time.

The problem is, there are huge dependencies in the system. For example, we've talked about certificate pinning, where you make special case exceptions to specific certificates. You say, okay, this certificate is - I'm going to trust this one explicitly due to its hash or its serial number, period. And typically its hash is what you trust. That's called "pinning" it. And there's instances where we're relying upon long-lived certificates throughout the infrastructure. So unfortunately, switching over to this rapid short-life certificate breaks all kinds of other assumptions that are longstanding. Thus the idea of leaving all that alone and then checking to get a near real-time, can we really trust the thing that we're being told to trust, that's the solution that the industry has ended up choosing.

So now, with OCSP, Online Certificate Status Protocol, it's very much - it's sort of the same concept of a certificate revocation list, but no more lists. We say, can I trust this certificate? And what we get back is, again, a yes you can, for a day. That is, there is again caching involved. So browsers can do, they don't all do, but they can have OCSP caching, which just speeds things up in the short term. Normally you're not losing much there because, even though the certificate authority may update its OCSP server instantly, in some cases they're aggregating updates. And so they're not updating their server for, maybe daily, normally no longer than that.

And in my own experiments that I've been performing the last couple weeks, the online update seems to be very, very quick. And so, but there's also caching involved. You guys may know that I set up that revoked.grc.com site. It turns out that, when I first put that certificate up, it was not revoked. Then I had DigiCert revoke it for me, and the Windows server refused to acknowledge its revoked status. Nothing. I mean, I struggled for a day to, I mean, flushing caches and restarting it, and I could not get it to let go of it.

And in fact, finally what we ended up doing was we pre-revoked a certificate and made sure that it was in the system, in OCSP and in CRL, before I ever let that special revoked.grc.com server even see it because it was so aggressively caching it. And that's finally the way we were able to get it to show the proper status. And I'm sure it would have, in fact, I did, I did go back and put that original cert in, and Windows now saw, oh, yeah, that's revoked. But it did take some time. So there is caching built in throughout the system. Normally, though, down around the scope of a day or so. So again, the system operates.

Now, OCSP gives us - it lightens the load. And it means that the browsers aren't pulling back big lists. But it does create some problems. So one problem we touched on already, which is response time. That is, the user's feeling of snappiness, of speed. They enter the URL. They hit Enter. They want an answer. But look what's going on behind the scenes now to be able to give them an answer they can trust. And that is that most of the time another query needs to get made, either to a hopefully not too big CRL, or to the OCSP server to verify. If they're coming back to the site they visited an hour ago, probably no delay because either the CRL or the OCSP is cached. And it's saying, yeah, we checked an hour ago. Everything looked fine. We're going to assume - and the world hasn't ended in the last hour. And so then it's snappy. But it's still the case that there can be some delay introduced, certainly the first time you make this access.

Now, good, and I guess I'll say, responsible certificate authorities have stepped up and realized they can help everybody by segmenting their CRL. That is, since the certificate specifies the URL for the CRL list in the original model, there's no reason that every certificate they issue has to have the same URL to specify the same list. And thus that list has to be big enough to encompass all of those certificates that might be revoked. There's no reason. You could easily have 50 lists. Just stick a zero through 49 embedded in the URL somewhere, and sequentially assign the URL to certificates, and remember which certificate went to which URL.

Suddenly, now your lists are one-fiftieth the size for everybody. That's good for the bandwidth of the certificate authority because they're not having to deliver this monster list to every single person who asks. Now the people who are asking get a list one-fiftieth, or even smaller, depending on how many segments you create. And so there have been workarounds that just make sense, that lighten the load. But speed is still an issue.

The other problem is, what if you get no response? And historically, certificate authorities have been better about sending out certificates and cashing people's checks than they have been about making sure that their backend response on their revocation systems, whether CRL or OCSP - and by the way, all certificate authorities have both now because we're in this dual-standard mode. So there have been instances, and also claims made, that these certificate responses can be really slow, or maybe entirely not forthcoming. So that begs the question, what do you do for no response? And the standard answer has been we're going to do what's called "fail soft." We're going to fail the verification soft, meaning we're going to assume the best. We're going to allow the page. If we haven't been able to confirm it's bad, we're going to assume it's good.

So we've been living in this world for a while. And this is the crux of Adam Langley's argument, that revocation is completely worthless and broken because, in a world where you fail soft, he contends that an attacker can defeat your ability to verify whether the certificate is bad or not. And if that's possible, in a fail soft world, then you're still going to visit the fraudulent website. And it's absolutely true that, if an attacker can defeat the revocation checking, and you're set to fail soft, then you'll trust a certificate you shouldn't. My argument is it's certainly the case that that's possible. But there are many scenarios, for example, where you might already have the certificate cached from your use of Amazon the day before, and it already has the revoked certificate in the cache, and there's no fetch needed to find that that certificate should no longer be trusted.

So there are situations where it absolutely works. There are situations where it absolutely doesn't. And then there's probably the larger situation where it depends upon where the bad guy is placed. If they're right next to you in a networked sense, they have more power over controlling you. But if they're over in Russia and relying on poisoning DNS and managing to fool one record, which is difficult enough to do, it's much more difficult to fool more than that, and they may have no access to the other path that your browser makes to get to the revocation.

So it's certainly not the case that it is always possible for an attacker to block revocation. But the whole problem of the browser making an additional query hasn't set well with the engineers of the Internet. And so they came up with a brilliant solution, which is called "stapling." Stapling, as in stapling papers together, it's in that sense. I don't know, actually, if there is another sense.

Leo: Well, yes. Remember? Oh, no, I was thinking of Apple's word for - entanglement. No, stapling has no technological use. Staple is staple. You can staple a stomach.

Steve: So here's the beautiful solution. The web server, the same server you're connecting to, does the OCSP lookup, not per query, but occasionally. It makes the query to the certificate authority for a fresh OCSP assertion. The certificate authority signs these in any event. No matter who asks for them, they're signed on the fly by the certificate authority, saying, yes, here is a fresh assertion that this certificate is still good. So the web server maintains a non-expired OCSP assertion. And say that these things expire daily. So every three hours, to be safe, or maybe every eight hours. But whatever, three times a day it independently makes a single query to the OCSP server, gets a fresh assertion, and keeps it. Then with every query that the browser makes, that is, when the browser connects to the server, the server is able to not only give it the certificate, but it's able to staple to that certificate a non-expired, fresh, recently received by the certificate authority and signed, assertion that, yes, the certificate is still good. It staples the response to the cert.

So this, I mean, this is where we're going to end up being. This is the right answer. Because look at what this means. First of all, in the browser queries, the certificate authority model - say that you have a busy website. It's Google. And that means that everybody's browser visiting Google is having to also, as many initial connections as Google makes, there are the same number of those connections going to the certificate authority to verify this certificate. Now, of course that's mitigated hugely by caching; also by, remember, that we're reusing connection credentials, secure connection credentials. So all the other resources and all the other pages that you get after the first one, they're going to be running over the trust that the first connection establishes. But still, all the initial visits to that site will result in individual queries going to make, you know, redundant, essentially, to make sure that certificate that everybody is getting is good.

So it's so much cooler and cleaner if that server instead occasionally, three times a day, and that gives it plenty of time to fall back because it's got a certificate lasting - it's got an OCSP assertion lasting 24 hours. So after eight it starts making sure that it's going to be able to always present a fresh one. That gives it lots of time, dramatically lessens the load and the bandwidth consumed over on the certificate authority side, and eliminates all delay. With the certificate you get a fresh stapled assertion that it's trustworthy. And when that certificate is revoked, then obviously the certificate authority will never again issue a, "yes, this is a trustworthy certificate." So it's impossible for the bad guys to staple a fresh assertion to the certificate.

So a couple things this thing needs. First of all, this thing needs support of standards. Now, the fact is it has been available in servers for years. Microsoft was one of the early leaders. And at least Server 2003, I don't remember, I'm not sure exactly when, if it was in the very first release of '03, but Server 2003 and '08 and '12, they've all got it, and it's enabled by default. So you guys all know the term I've coined, the "tyranny of the default." Default settings tend to be the way things stay. And Microsoft has it on by default. All Microsoft servers on the 'Net are offering stapling. Apache has it. Nginx has it. LiteSpeed has it. Most of them have it turned off because it's disabled by default. So one of the things we've got to get the industry to do is to start turning stapling on.

Now, the server won't offer stapling unless the browser says, I know about stapling. That is, so in this initial connection handshake, one of the things that the so-called "client hello" message, which our listeners who remember talking about SSL and TLS will remember, that's the first, after establishing a TCP connection, then we bring up the TLS connection. So the client hello package from the client needs to say, tell me about stapling support. Like, I want it. I know about stapling because I'm a hip client over on this end. So if you can provide it, I want it. And that's necessary for compatibility's sake. Unfortunately, there are clients where, if the server gave them information that they didn't understand, they would break. They would just - their connection would collapse, or the thing would crash, or who knows what horrors. But the idea is, for compatibility's sake, we have to ask for it.

So what we'll be talking about next week is what level of support on the client side and in browsers apparently exists. It's been around enough now, and long enough, that the underpinnings, so to speak, are in place for this. And so, when we ask, all Microsoft servers are now able to provide a stapled response. Which means zero delay and affirmative confirmation of the status of the certificate. And the other servers, the majority servers on the Internet, the Apaches and the nginxes, can have it turned on. They ought to have it on now. There is no reason for it not to be on. It's obviously not breaking anything. Microsoft servers run with it on, and it's working perfectly.

What anyone who's curious can do is the DigiCert guys added a page that allows you to check any website for this kind of revocation, which is stapling, OCSP, and CRL. It's digicert.com/help. So it's just a Help page at DigiCert. You put the URL or the domain name of a site in, and it will very quickly show you whether stapling is supported in addition, I mean, almost everybody supports OCSP and CRL. I can't imagine that there wouldn't be support. But it's nice to see that that's there. Also SSL Labs I should mention because they were quick to respond to the Heartbleed side. They also will show you whether stapling is supported. GRC has it because I'm on a Microsoft server. And Google doesn't because, well, and therein lies a tale. That's a complex story. But again, most of the servers on the 'Net don't. And we've just got to get this fixed.

Now, bad guys can attack this, too, because we don't know, as the browser, that the server supports stapling. That is, the client can say in his client hello, hey, I want to get OCSP response, and I know all about stapling because I'm modern and hip. So, if you can, send me a stapled response. Now, the problem is we don't know that the server that we're connecting to supports stapling. So we don't know if it responds, sorry, I don't staple, whether the legitimate server does or not.

So, for example, say that - we'll use GRC, for example, because I do staple. So you connect to the real GRC.com, stapled response, you know the certificate's good. Everything's great. Somebody steals my keys, sets up a fraudulent GRC server. Well, all they have to do is not staple. That is, clients say, hey, we're modern. Tell us, reaffirm your certificate status with stapling. And the fraudulent GRC server says, sorry, we're in the Dark Ages, we don't staple. Well, in fact that's the majority response you're going to get from the Internet today. But specifically, it's true for the real GRC site, not true for the fraudulent one. But the browser doesn't know that it should have a stapled response. So it says, oh, shoot. Okay, you can't tell me. I'll go use the traditional means of querying the certificate authority for either its OCSP or its CRL.

So the final solution is called "OCSP Must Staple." And that's where we have to go. That's where we are going. We're not there today. But we're surprisingly close to having it in pieces. So the way OCSP Must Staple will work is, when the site, like mine - and believe me, I'll be on this immediately - when a site like mine gets a certificate, either in the so-called CSR, the Certificate Signing Request, which is what I provide to the certificate authority, there might be a new field in there saying "must staple." Or it might be part of the form, the online form you fill out where you say, I want to use SHA-1, or I want a key length of 4096, or I want to use SHA-2, you know, there are some parameters you can choose in the certificate with the certificate authority. One of the things there could be a checkbox saying "OCSP Must Staple."

So what happens is, using again these certificate extensions, there would be another extension, something like - and it would be wonderful if they would name it "OCSP Must Staple," but they'll probably call it some gobbledy-gook. Who knows what they're going to call it. The point is its presence in the certificate would assert that the server serving this certificate staples. Now the bad guys are toast. Nothing they can do because, if they steal a certificate which is marked "must staple," then they can't remove that.

Remember, you can't mess with the fields in the certificate, or that breaks the signature. So if it says "must staple," and the browser receives a certificate that says "must staple," then the browser - well, in fact, the browser, the sequence would be the browser in its client hello would have said, hey, I'm hip, I know about stapling, so bring it on. The server, trying to fake it, says no, no, I don't know about stapling, and sends a certificate back that says "must staple." The browser says, sorry, something wrong here, not trusting you. So it is absolutely bulletproof.

Now, there is an interim measure that I believe Firefox is not far away from stepping up to. A guy named Brian Smith has been working on this for a while, and it's very clever. And it's very much like something we've talked about before. Remember HTTP Strict Transport Security, HSTS. That's a header which the server provides to the browser saying all of its connections must be secure. And, for example, GRC sends that header out. With every single reply, we assert only connect to us securely.

The beauty of that is that - oh, and with that there's an age. We're able to say, I don't remember what the phrase is, but essentially it's a huge expiration, and mine is set to maximum because I'm never not going to do SSL anymore. So the browser gets that and caches it. So even absent a certificate, because we don't have that infrastructure in place yet, I mean, actually apparently we're close. The OID, it's that weird-looking code, the 3.1.2.9., you know, it's one of the standards-based identifiers, that's apparently close to being assigned by the committee that creates all of this stuff. So we're near to having that, maybe.

But then again, remember, I mean, it's got to get through, I mean, all the software has got to get modified, all of the certificate authorities have to modify their issuing, they've got to create UIs, they've got to start issuing certificates that have this. So it may just be, no matter what we do, two or three years, unless people want to explicitly go and revoke and renew their certificates in order to get this, it's going to be some time before we get that assertion broadly present in certificates.

So in the meantime this - and it may just be called must-staple: may be what the header is going to be. And it'll specify the domains where stapling must be provided and the age, the duration for which this assertion will remain true. And every time it's received from the server, that renews the browser's knowledge. Now, this doesn't help the so-called "first contact" situation, obviously. And this is the same problem that the HSTS header had. It's one thing to say I am never not going to have a secure connection. But if the absolutely very first connection you have is not secure, or there's a way to intercept it and do what's called a downgrade attack, to take the s's out of the http's so that you establish a nonsecure connection, that even though the responses are screaming, we only want security, we only want security, the bad guy could strip that response so the browser never is told that the server was sending an HSTS header for security.

Similarly, if the server is sending "must staple, must staple, must staple," but if the bad guys are able to intercept that, the first time the user goes to that site with that browser, or before a previous assertion has expired, although they last for years, in that case that's the first-visit problem. Once we have this in certificates, that gets defeated, too. Nothing can override the assertion in the certificate. But until then, an extremely good solution is for us to - oh, and servers can do it. In the same way that it's trivial to add static headers into responses, I mean, all the servers on the planet, you can just have them say whatever they want to in the response headers. In the same way that many servers added the HSTS header, we will add must-staple and the various arguments immediately, as soon as we settle on what the name is going to be. And then we're asserting, GRC and whomever else wants to support it, that our server, the server at GRC.com, will always provide stapling. And it makes us absolutely immune to anything the bad guys can do.

We still have some caching window, like a day. Although, wow, when you consider that a certificate is sitting around for 18 months, and it actually does take time to set up infrastructure and so forth, it's unlikely that a day represents much of a weakness. But no one, I mean, essentially that's the tradeoff we make. And there's nothing to prevent high-value sites from arranging with either their server or their certificate authority to say we want, you know, we're super high value. We want to even minimize that one day. Let's make it an hour. We want to receive from you OCSP responses only valid for an hour. Give us that, we'll pick up an update from you, starting after 30 minutes until we get one. And then we've just pretty much squeezed this window down to nothing.

So that's really where we are today with how certificate revocation works, why the list concept grew over time, how certificate authorities could fragment the list into many sublists in order to solve the big list problem. One notable example of that not being done just happened a couple weeks ago with Heartbleed. The CloudFlare guys, after they had the confirmation from their CloudFlareChallenge.com site, and they revoked that certificate, famously - I'll talk about why that was famous next week - they then revoked on the order of 140,000 GlobalSign-originated certificates, and the GlobalSign CRL is currently more than 5 million bytes. It exploded the CRL's size. And unfortunately, that's 140,000 sites that are represented by all of those expired domains.

And to be responsible, CloudFlare did the responsible thing. They had to revoke them because it was proven then that any of those keys could have been obtained by bad guys without any record that anyone knew to be making at the time that that was being done. But there's an example of how this blob suddenly has exploded this CRL. And now, unfortunately, it's going to drift through time. As those certificates randomly expire, they'll be removed from that CRL. And so over time, because there certainly won't be nearly as many coming in the front end of that list, they'll be expiring off the back end of that list, and it'll return to a manageable size.

The good news is that the CRL is really now no longer being used to nearly the degree that OCSP is. OCSP, for all of these reasons, is the preferred mechanism. It is faster. It is smaller. You're making a specific query for a specific certificate. And you get a yes, that's fine, or a no, that's not. Unfortunately, we still have the situation of non-answer. What if you get a non-answer? Which is where soft fail comes in. The only browser that's currently available that supports hard fail as an option is Firefox. Arguments are that, oh, no, you can't use that, absolutely not, it'll just - it won't work. You'll get all kinds of false blocks. For what it's worth, I've always had it on, and I've never encountered a failed page, a page that comes up and says I can't deliver this page to you because we cannot determine, I mean, I don't even know what it looks like when it says that because it's never happened.

So if anyone is curious, I mentioned this last week, but now we have the foundation. And if you're a Firefox user, by default Firefox is doing OCSP revocation checking. You can turn on "Must obtain a response or treat the page as bad." And then you are protected, I mean, really, really well protected from any kind of spoofing. Next week we're going to go into, now that we've got a good sense of theory, what the various operating system platforms and browsers actually do today. And then, as this evolves in the future, you know that we'll come back to it and keep everybody updated.

Leo: The hard fail that I had with the Mac, with the Keychain thing, I did experience massive problems on that.

Steve: Yeah. And I had some conversation with Ryan, who's in the Chromium project. He indicated that there is a problem with the Mac. So I think specifically there's something that the Mac is not doing right that causes that problem. I think he called it a "thread deadlock" in some circumstances.

Leo: Yeah, because it wasn't with browsers, it was with just apps of various kinds, maybe apps that are doing some sort of certificate validation. I don't know.

Steve: Yeah. So again, this is something that - the reason I did the revoked site, the reason I've created some pages now at GRC.com, is I want to explain this. I want to just shine a light on this because one of the things that I've seen people saying, I mean, like the engineers are saying, well, there's really no demand for improved revocation. It's like, no, there's no understanding that it's broken. There's no knowledge in the user community that it doesn't work. Everyone's just been assuming that it works. And unfortunately, it really has a ways to go. We're very close. We just have to sort of get off our butts and make this work. And actually, Apple's got to fix this, certainly on the Mac platform, so that it works. I mean, we need to - it's going to take the end-users to tell the websites they care about that they want to see OCSP stapling enabled, and the browsers that they care about to say, look, I want certificate revocation working, rather than not.

So anyway, I have some amazing news to discuss next week that, again, I think people will find very interesting. There's another page on GRC about CRLsets which is linked - I'll stick it off the Main Menu. I haven't yet. It's down - there's a series of links at the bottom of the revoked page. I finished it Sunday. It's a very careful analysis of the solution that Chrome has chosen, which is one of the many things that we'll talk about next week. But if anyone wants a sneak peek - and actually, everything I just talked about I also have documented on what I call just the Commentary page. The title is "The Case for OCSP Must Staple" because that's really where we have to go. That solves this problem. And it's one that just no one seems anxious about. But it's just we're so close to getting it fixed. We just have to want to have it work.

Leo: Well, there we go with Part 1 of the Certificate Revocation saga, a new...

Steve: In fact, I did create - I created a bit.ly link, come to think of it: bit.ly/crlsets, all lowercase, bit.ly/crlsets. And that'll take anyone there who's curious about the topic for next week.

Leo: Cool. Study up. Bone up for the next episode. God knows you'll need to. Steve. We're not going to do a Q&A next week, then.

Steve: Week after.

Leo: Week after. So questions for Steve go to GRC.com/feedback, and we'll answer some in a couple of weeks. You can hear the show or watch it, if you wish. Somebody said, why am I watching this? It's just Steve's head. Well, it's up to you. There's also PDP-8 lights going on.

Steve: I've always wondered the same.

Leo: Yeah. You resisted video harder than almost anybody.

Steve: And even worse, Leo, it is my head.

Leo: Yeah. Yeah. Even you didn't want this. Too late, Steve, too late. You can watch or listen at, let's see, 1:00 p.m. Pacific, 4:00 p.m. Eastern time, 20:00 UTC on Tuesdays, right after MacBreak Weekly. But you can always get on-demand video and audio after the fact. Audio, 16Kb audio available at Steve's site for the bandwidth-impaired, along with transcripts. So you could watch, listen, and read, if you wish. GRC.com. You can also find SpinRite there, the world's best hard drive maintenance and recovery utility. And lots of free stuff and research. It's just a - it's an encyclopedia of geeky stuff.

Steve: Indeed. I forgot to mention that we had a - one of our listeners did a really cool project. He graphed the running time of the podcast from its inception. We'll show it next week. I just...

Leo: That's a one-way street, I'm thinking.

Steve: I didn't get it - yeah. It's really...

Leo: All uphill.

Steve: And he separately graphed the Q&A and the non-Q&A episodes. And so it's fun actually to see it. So we'll show it next week. I just - I didn't have it written down in my notes, and so it just didn't go out over the air. But I've got it in my notes already for next week. So we'll show that. It's pretty fun.

Leo: And already "Ventman," sitting in the chatroom, has added OCSP stapling to his Apache server. So you...

Steve: Yay.

Leo: One server at a time, you're changing the world.

Steve: Yay. Yeah, and you can go to digicert.com/help, put your URL in, put your domain name in, and DigiCert should verify that you are now stapling. Very cool.

Leo: Yeah, yeah. He's obviously got a good server. He's getting an A+ on SSL Labs and everything. He knows what he's doing. We do have full-quality audio and, yes indeed, high-definition, crystal-clear video of Steve's moustache available at TWiT.tv/sn for Security Now!. Or you can always subscribe to all the different versions, everywhere that you can find netcasts - iTunes, in fact all the apps on all the mobile devices. Or get the special TWiT apps. There's TWiT apps, all official, all approved by me on Android, on iPhone, on Windows Phone, as well. I don't know if we have a BlackBerry app. I think there's less demand for that than there used to be.

Steve: I don't think that's a valuable expenditure.

Leo: And you, the BlackBerry man himself.

Steve: I gave up. I can't go back.

Leo: Thanks for being here, Steve. Great stuff. And we'll talk again next week on Security Now!.

Steve: Thanks, Leo.