Strict Transport Security

Description: This week, after catching up with the week's security news, Steve describes the exciting emerging web standard known as "STS" or "Strict Transport Security" which, when supported by browser and website, allows a website to dramatically increase its access security by telling the browser to only connect securely and disallow any security exceptions.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-262.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-262-lq.mp3

Leo Laporte: This is Security Now! with Steve Gibson, Episode 262, recorded August 18, 2010: Strict Transport Security.

It's time for Security Now!, ladies and gentlemen. Fasten your seatbelts. We're about to find out what's wrong on the Internet this week. Steve Gibson is here, our security guru from GRC.com. Good morning, Steve.

Steve Gibson: Good morning, Leo. It's great to be with you again, as always.

Leo: Well, thank you. We have an interesting topic for the day today, which I don't really even understand. So I'm going to let you describe what Strict Transport Security is.

Steve: STS, yes. It's an emerging, rapidly emerging standard and solution for web browsing, which is really very exciting. And you're going to be an expert about it, as will all of our listeners, about an hour from now.

Leo: So this is to replace SLS or SSL or...

Steve: What it does is, in a very useful way, it enforces the use of SSL, also known as TLS, basically secure socket layer. It enforces its use on websites. And specifically, and this is what's so very cool about it, is for so-called "user agents," which is what we call browsers, for browsers which support it - and already Chrome does, and Firefox 4 will, and NoScript has for a while, so it's already deployed, essentially, for probably the

majority of our listeners. For websites which support it, it allows the website to say "we're serious about security" to the browser, "so only connect to us using SSL connections, using secure connections."

Leo: Oh, if only everybody would do that.

Steve: Well, we're on the way there. And so I'm going to talk about the recent history of this and exactly how it works; what the problems are, which are many, if this isn't in place. And what I love about this is, I mean, it's happening now. I just checked my Strict Transport Security tag from PayPal. PayPal's probably one of the very first adopters. And they have it set not to expire for 40 years. So...

Leo: It's like, that's a long time.

Steve: So my browser will not, will not generate an unsecure connection to PayPal, no matter what I do. It will not let me click past a security certificate warning.

Leo: Oh, interesting.

Steve: It strengthens the whole channel in a really substantial fashion. And so I knew that our listeners would be saying, hey, we have that? We didn't know we had that.

Leo: Very, very interesting.

Steve: And, well, we have it a little bit, and we're going to be getting it, I think, a lot very soon.

Leo: Well, good. We're going to get to that in just a second. We also have, as usual, security updates. But before we go too much farther I should say "Happy Anniversary." Are we now finally on our sixth year?

Steve: Yes, this is Episode 1 of Year 6. And the controversy over that has subsided.

Leo: Thank god. Okay, good.

Steve: So we're into Year 6.

Leo: So, well, Happy Anniversary, yeah.

Steve: Yes, indeed. Thank you.

Leo: So let's get the security updates.

Steve: Well, we didn't have very much because of course we had the grand mal update week last week.

Leo: Right.

Steve: Apple has updated their iOS. We were talking last week about that you could jailbreak your phone just by going to a website, which gave you a PDF, because there was a font-parsing vulnerability in Apple's own PDF parsing. Some people erroneously just assumed it was Adobe's problem. But it wasn't, never was. I mean, everything else is, but in this case not that. So that's been fixed. The concern was that not only did it make the iPod, the iPhone and iPads easily jailbreakable, but much more of a concern is that all kinds of other mayhem was no doubt going to follow quickly. So Apple jumped on it. It was an easy thing to fix, and they pushed that out.

And then the only other update we have is just a little notice about Opera. It went from 10.60 to 10.61. They had a bad vulnerability, a heap overflow flaw in their HTML 5 rendering, which could be used to inject code into unprotected systems. So as soon as they learned about that, they fixed it. And at the time they also fixed a number of other little stability things and UI problems. So that's the extent of our updates for the week.

Leo: Wow. That's nice. We also have, I guess, a little security news before we get to our first commercial of the day.

Steve: Yeah. As expected, India has said to BlackBerry's creators, RIM, that they are insisting upon having access to BlackBerry's communications channels. They were rumbling about it. We talked about it last week. Then they gave RIM a deadline of August 31st, which is actually much less time than the Saudis gave RIM. And RIM of course did agree to put three servers in Saudi Arabia in order to solve that problem. I imagine something similar is going to happen.

Basically, it's clear that RIM doesn't have much choice. If countries are saying we're going to block your communications unless you give us the ability to eavesdrop on those communications, then what are you going to do? And, interestingly, India, the guy in charge of these things said publicly in a conference about a month ago that he was thinking of telling Google and Skype something similar. So it's like, okay.

Leo: Yeah. Is this a legitimate thing, or is it really that they want to spy on their...

Steve: Oh, I mean, no one's even making any...

Leo: Bones about it.

Steve: ...bones about the fact that this is specifically - they couch it in antiterrorism

terms. And terrorism is a problem for us all now. And presumably they would wrap that in requiring the equivalent that we have in the U.S. of getting a court order in order to get access to that kind of communications. You would hope that it would be protected, and it wouldn't be abused and so forth. But, you know. So that's where we are.

I was just going to say that it's better to have encryption providing those sorts of protections. On the other hand, the problem was the BlackBerry technology is so good, so state of the art, that it was unbreakable. And, I mean, even RIM, RIM says we can't, we can't eavesdrop on these messages. We designed it so that it's end-to-end secure. So now they're saying, okay, well, we'll crack this for you in order to make eavesdropping possible. So one just hopes it gets wrapped in...

Leo: It's sad.

Steve: It is. It's, well, but see, this is probably an inevitable consequence of crypto being so good. I mean, it is unbreakable when it's done right. And RIM really did it right. And as soon as governments say, wait a minute, what do you mean it's unbreakable, we have to be able to break it, it's like, well, no, "unbreakable" means unbreakable by anybody. Oh, well, that's unacceptable. Break it. So...

Leo: It reminds me, really reminds me of the early discussions about whether there was a back door. Remember there was going to be, briefly, Al Gore suggested an encryption technology, what was it called, that had a backdoor, basically.

Steve: I know the one you're thinking of.

Leo: Maybe it wasn't Al Gore. Al Gore was suggesting the Clipper chip.

Steve: That's it.

Leo: Was it the Clipper chip?

Steve: The Clipper is the name I was trying to remember, so...

Leo: Yeah. And there was a - and we - the bad guys - the good guys, I'm sorry, would have access to your stuff.

Steve: Well, and then there's key escrow, which is the other approach, is, well, okay, you can have the crypto, but you've got to put the keys in an escrow somewhere so that under certain circumstances we can pull them out of escrow and decrypt stuff. It's like, okay.

Leo: Grumble grumble grumble. Yeah.

Steve: So, grumble, yeah. The press is reporting, although I'm kind of curious about this, they're saying that the first smartphone trojan has been detected. But my sense is there have been things like this, like back in the Palm Pilot days, I mean like in the Palm Treo days, where there were...

Leo: Oh, yeah.

Steve: Software would sneak in that would...

Leo: To say "the first" is kind of absurd.

Steve: Yeah, I think that's the case, too. That's what they're saying. But it is the case that something that has been named Trojan-SMS - which I think is a little bit too broad, unfortunately, that name. Now we've used that one, I don't know what they're going to do from there. But there was a trojan detected for Android which was making premium rate SMS message calls without the user's knowledge or consent, behind their back, thus essentially transferring money out of the user's account into the attacker's account. So that's been found and stopped.

And in the various commentary that I was looking at, it was what we talked about last week. Essentially, users are going to have to take more responsibility for the permissions that they give applications. If an application isn't clearly about sending SMS messages out, then when you install it, and you get a dialogue that pops up enforced by the operating system that says the application wants the following permissions, well, really take a look at those and decide if you want to give it those permissions. And if a screensaver wants to do SMS messaging, you need to say, uh, that doesn't sound like something the screensaver ought to need to do.

Leo: We've talked about that a little bit on This Week in Google. And it's really a tough point because most people aren't going to be reading all these things. I would like to see a firewall on the phones, where it says, hey, this app - kind of like on a computer. This application's trying to access this resource. You want to permit it to do so from now on, block it for today, block it forever, what do you want to do?

Steve: Right. Very much like, exactly as you said, a personal firewall on a PC. The more granular controlling firewalls that many people still prefer today, even though arguably Windows has firewalls built in for doing incoming, and even the newer versions of Windows provide you with some outgoing, outbound support. So, yeah.

Leo: Right, right. That's what we need. The problem, of course, is these are very constrained environments, constrained resources. And so I don't know if they want things running in the background, doing that, what do they want?

Steve: I think the problem is more the user, Leo. Because most people, again, they're just going to glaze over when they see these things coming up and saying, wait, wait...

Leo: And do you want your phone to be bugging you, yeah.

Steve: ...blah, blah, yeah, exactly.

Leo: Yeah, yeah. It's an interesting question.

Steve: So we're going to have a problem. I think we can, I mean, I don't intend to turn this podcast into the smartphone security problem.

Leo: It's going to happen, I think.

Steve: It is going to happen.

Leo: Yeah, eventually.

Steve: You're going to be spending more and more time talking about, I mean, this is - the smartphone is a computer with communications. People want to just download apps. Apple brags about how many hundreds of thousands of apps there are. "There's an app for that" is now passed into the common jargon of our society. And so the problem is security. And so it's going to - we're going to see serious security threats in the future. Everyone who follows this stuff knows that.

Leo: Right. You want to do a SpinRite? And then I'll do the Carbonite commercial.

Steve: I have a nice, neat testimonial from someone who took a different approach to running SpinRite than we've ever discussed before, so I thought that was fun. He called it "The SpinRite On & Off Testimonial." He said, "Steve, before I left" - and this is from Ryan Wright. "Before I left, I worked for a St. Louis school providing network and systems support. I inherited a Windows 2000 Exchange Server with no RAID array and unreliable backups, and the computer was stored in the same room as all the large electrical equipment servicing the entire complex. Before I had the chance to get things under control and comfortable with the reliability, the worst happened. One evening I got a call that no one could access their email. I tried to log in remotely to the server. No luck. I drove to the school and found the server displaying all sorts of error messages about files being inaccessible and services failing.

"As I tried to investigate, the computer froze up on me. Upon forcing a reboot, I got the dreaded message telling me no operating system was installed. I immediately jumped out of my chair, grabbed another computer, and purchased a site license for SpinRite. I burned a CD disk and started it going on the server. It found errors very quickly and began their recovery. I've heard many stories of SpinRite taking quite some time to finish, so I was worried that, since the next day was a school day, if SpinRite wasn't finished, or if this didn't work, no students, teachers, or staff would be able to log into their computers on the domain. As I said, I inherited a much-less-than-desirable setup.

"So I got up early the next morning and arrived before most everyone else. SpinRite was still chugging away, fixing errors, but only 12 percent through the large drive. I decided to try my luck, quit out of SpinRite, and rebooted the server. It booted. But during the startup sequence, Windows noted several problems causing various services to fail on startup. Clearly we weren't out of the woods yet. But the server ran well enough for all the functions the school needed most, and we passed the day without anyone else even noticing. That evening, I booted into SpinRite again and started it up where it left off that morning."

Then he says, "The next morning I noted the position and stopped it again. This start-and-stop process went on for four days before SpinRite completely repaired the drive. I have since improved many things in this setup. But because of SpinRite we averted catastrophe, and even averted inconveniencing all the computer users school-wide. Thanks for such a great product and especially for building in the features to stop and resume SpinRite partway through its operation. You saved a whole school."

Leo: Wow.

Steve: And that's something we've never talked about, was SpinRite, when you interrupt it, it gives you the percentage it's complete to four decimal points.

Leo: Because we can.

Steve: Because we can. Well, and because on a large drive, those fractions of a percentage represent large chunks of data. And so the way I designed SpinRite, when you interrupt it, if you tell it you want to quit, it'll say, wait a minute. And it makes a point of reminding you to note where this is, in case you want to resume.

And then I was very careful with the math such that I'm rounding in the right directions, so that if you then resume SpinRite and tell it where you want to pick up from, it makes sure that there's a little bit of overlap between the number it gave you when it stopped and the number it uses when it starts again so that there's no gap that you miss. That is, it starts, rounds backwards to where it starts from. But that does, just as Ryan used it, it allows you to take what would otherwise be a multi-night or perhaps a multi-day run, when you can't afford to have SpinRite tying up your drive, and do it in multiple pieces. So it ends up being very effective.

Leo: Such a good idea. So let's talk about STS. I've seen that setting. Sometimes when I confirm mail settings and things like, I've seen STS. Is it implemented now? Is it out there now?

Steve: I think you must be thinking of something else. Oh, you're thinking of...

Leo: Oh, TLS I see.

Steve: ...of STARTTLS.

Leo: Oh, that's what I'm thinking of, you're right.

Steve: And that's an email setting. It's a way of allowing a POP and SMTP client to initially connect without SSL, and then to negotiate bringing up an SSL tunnel to protect your email on the fly.

Leo: Precisely, yes. It's a form of authentication, yeah.

Steve: Right. So, okay. Here's the problem. The nature of the browser-server relationship is transactional. We've talked before, the browser makes a query to a remote server; the server returns its response. That's where the web began. Today, 20 years later, that's where it still is. That's the way HTTP, the Hyper Text Transfer Protocol, functions. Whether it's secure or not, that is, whether the query and the reply are wrapped in an SSL-encrypted tunnel or not, the browser is this query-response relationship, has a query-response relationship with the server.

Well, that creates a bunch of problems when we want to move beyond passive browsing the 'Net, which we did in 1980-something, into this Web 2.0 world, where we want essentially applications running over the 'Net, where we want to have a secure relationship with the server at the other end. And so there are much more secure ways, fundamentally more secure ways to do this, for example, SSH tunneling, where you create a connection, you authenticate, and that connection stays up, it stays persistent. And so everything that you're doing is inherently secure through that connection. You can't, in that situation, drop the connection, reconnect, and just say, oh, it's me again, don't worry about it. The technology there is such that it says, wait a minute, you have to reauthenticate if you're going to reconnect. Not so with HTTP. Not so with web browsers and web servers. So that just isn't the way they've ever worked.

So the problem is, over time we keep burdening this model of query and response with things it was never intended to do. For example, it was never intended to send data back to the server. So how do we solve that? Well, we add stuff to the end of the URL. Or we add - we have, like, a blank line after the query headers, in the case of a post, and we add data there in order to sort of sneak it back into the server, overloading the original concept of how the web works.

Similarly, when we wanted to have a persistent logon relationship with a remote server, the architects said, well, how do we do that? Because individual pages that a user goes to are separate connections, queries, responses, and then end of connection. So we know that the way this was then created was this notion of cookies. Cookies are these tokens which the server gives the browser, which the browser holds. And in subsequent queries, it sends the cookie back, sort of to re-identify itself from one query to the next.

And the good news of cookies is that allows us to, even though we don't have a stateful connection, that is, this is called a "stateless" relationship with the server because each query and response stands alone, as we do things with a website. We're reminding the server with each query by giving it the cookie back that it gave us, oh, this is us making this query. This is us again making this query. Oh, and here we are again. And so the server is able to sort of track us among all the tens of thousands of people that might be simultaneously using a busy server.

These individual queries come in, each with a cookie that helps disambiguate all these

connections. Basically they're all coming in on the same port. They're coming from different IPs. But you might have three or four people in a household all using a NAT router, so their IP is all the same, their public IP is all coming out, but their browsers would have different cookies. So the server, even though they're coming over on port 80, or 443 if it's secure, even though they're all appearing to come from the same IP address, their browsers will have different cookies that allows the server to know who they are.

Well, that was fine until bad guys got into the act. And the problem is that even sites which do allow you to set up a secure logon, and Google has historically been a classic example of this, and we've talked about this in several contexts, Google will take you to a secure logon to ask for your credentials. It then gives you back a cookie, which identifies you, just as I was saying, in subsequent queries to, like, Gmail, for example. And then it drops you back out of security. It drops you back to an HTTP URL.

Now, the problem is this credential which you were handed is now being sent to Google as you mess around with Gmail in the clear, that is, nonencrypted. And a passive eavesdropper, somebody snooping at Starbucks over a non- or any - I don't mean to pick on Starbucks, but they're very well known. They do not have encrypted connections. It's open WiFi. So a passive eavesdropper can simply be sniffing traffic, and they will see this cookie going in along with your queries to Gmail. Well, nothing prevents them from grabbing that cookie and impersonating you. Literally, your logon-ness is that cookie. That's this magic token that says, I'm still me. I'm giving you back this little gizmo that you gave me before just so you're able to track me, you're saying to the remote server.

The problem is, this is fundamentally insecure. I mean, it's frightening. So a couple of security researchers at Stanford University in the web security group saw a demonstration of this back in '07 at a security conference where it was made very apparent that the fact that these logon relationships with remote servers were being maintained with insecure cookies, it made it so clear to them they thought, okay, well, this is a serious problem that needs to be solved. There are other problems because, for example, cookies can be marked as secure. You're able to say - the remote server during a secure transaction is able to hand the browser a cookie with a secure tag on it which says "never send this cookie unless we have an SSL connection." And so the browser is able to protect this credential that you've been given by the server in the form of this secure cookie so that it will not disclose it unless you have a secure connection. Now there's, it turns out, though, some problems even doing that because we know that users click through warnings that their browser gives them.

Leo: Oh, yeah.

Steve: Yeah. I mean...

Leo: Of course.

Steve: Now, and sometimes you want that. For example, there are sites that use a self-signed certificate. They're typically maybe non-high-value, non-commercial sites. They want to provide encryption, but they don't want to go pay VeriSign or anybody else hundreds of dollars a year for a certificate. And it's hard to blame them. They may be old curmudgeons who just figure, hey, I'm not giving my money to the man. So they haven't - they have an SSL certificate, and they've signed it themselves. So it's valid, but it isn't

anchored by a certificate authority that the user's browser trusts.

Consequently, when you go to one of those sites, all browsers will pop up a warning saying, hey, this certificate was signed by somebody we don't know. Well, yeah, it was signed by the guy who owns the website, probably, rather than by VeriSign, who would have loved to have a few hundred dollars for the privilege of signing that certificate. So the browser pops up a warning. "Warning, this certificate is not something we can verify. It hasn't been signed by a certificate authority."

Users will go, look at that, and this thing's in their way. There's a warning, and it's like, I want to go to this website. "Click here to ignore this warning and proceed." Which everyone does. And in fact I had this - our listeners may remember that I was a little embarrassed when GRC's main security, www.grc.com, my own security certificate expired without me being aware of it. Maybe it was about six months ago. We still had SpinRite sales. People were still buying SpinRite. The only way, I mean, the only way they could have been buying SpinRite is they got a warning that said, "This certificate has expired, what do you want to do about that?"

Leo: And they just ignored it.

Steve: Now, they ignored it. Now, maybe they were like me, if I saw that happen, for example, I would look at it and go, oh, it expired two hours ago, or it expired yesterday, so I could sort of like forgive the webmaster. In fact, even Twitter's security certificate expired...

Leo: That's right.

Steve: ...a couple weeks ago.

Leo: That's right, I noticed it on TweetDeck. It kept complaining, there's no certificate, there's no certificate.

Steve: Yeah. So it can, it happens to the best of us. But the point is, users have been trained, if something comes up and says you can't have what you want, just press the button that gets you what you want, and get that annoying interruption out of the way.

Well, the problem with having been trained that way is that a completely possible man-in-the-middle attack could happen at any moment. Somebody at Starbucks does an ARP spoofing attack, which is trivial on open WiFi, that is to say, simply sends a broadcast out and gets the other people at an open WiFi hotspot to treat their computer as the gateway, so that now the traffic runs through that computer. When the bad guy sees a connection being made to PayPal or Bank of America or whatever, they return, on the spot, a self-signed certificate. That is, it's possible instantly to generate a certificate for anyone and just to have the certificate self-sign.

Leo: Oh, boy.

Steve: So that pops up the warning that the user kind of looks at briefly, and they've been trained, unfortunately, to just say, okay, fine, whatever, I need to do my business. So they click "okay." Well, what they have just done is authorized illegitimate security credentials for the attacker, so that the attacker is now the terminus of their SSL connection, and the attacker can simply filter everything they do over SSL because they're actually connecting to the attacker's computer, not to the remote PayPal or Bank of America or whatever, where they believed that they were going. And they've just - now they have this problem that they're being eavesdropped, even though they see that they've got an SSL connection. So there's an example of another way in which this current security model we have, unfortunately, with users participating, really is broken. So how do we fix this?

Well, the good news is, people who really understand that, like, the reality of the way people are using secure connections on the web, got together, and they have made an incremental change, an incremental improvement to the HTTP protocol which is in the process of being supported. And I'm very excited about it. As I mentioned at the top of the show, this so-called STS, Strict Transport Security, also known sometimes as HSTS because of HTTP, so HTTP-STS or just HSTS or STS, support appeared in Chrome with v4.211, so it's been in there for a while. It's been in NoScript, Giorgio's famous add-on for Firefox, since 1.9.8.9. And we now know that NoScript went to 2.0 a while ago. And that was, like, last September, September of 2009 is when he added it to NoScript. So quite a while. And it will be natively supported in Firefox v4, which we now know is at least at beta 3, last time I looked, and is moving along quickly. It is entirely foreseeable that it will be supported in every browser shortly because one of the things I like about it is it doesn't require any sort of major revamping. It's simple for browsers to support.

So here's what it does. When a secure connection is made to a server, which is to a site that is serious about security, I mean, that really wants security to be paramount, like PayPal - I use PayPal actually because an engineer, a security person at PayPal, I think it's Jeff Hodges, has been one of the driving forces behind this. When you have an SSL connection made, that site is able to provide, with its response, in response to a query, because the HTTP protocol is all, as I was saying, query and response, when the response comes back there's an additional response header added to the reply.

Now, the response headers are things like this expires for so long, the following content has the following length, there's like a number of so-called metadata, not really part of the page you're viewing, but sort of a preamble to that which the server and the browser transact. A new header is added that is strict-transport-security: and then a max age equals, and some number of seconds. And then optionally you can also - you can tag onto that that you want to include subdomains of the domain where this header was provided.

When an STS-aware browser receives that header in a response over a secure connection, that is, that information is cached permanently by the browser. So it remembers, for example, in the case of PayPal, PayPal sends back, when you go to www.paypal.com over a secure connection, it sends this header back. And I was poking around, looking at it this morning. Sure enough, back come these secure, or, I'm sorry, strict-transport-security headers from PayPal. An STS-aware browser, which are beginning to emerge, stores this on the hard drive, along with the length of time it is to live. And it will then never again make an insecure connection to that domain.

Not only that, it regards any certificate, any SSL problem as fatal. It will not allow you to click around it. It will not allow you to accept a self-signed certificate from that domain. It will not allow you to make a connection, even if the certificate is expired. So it certainly requires diligence on the part of the web server which is serving these

certificates. But in return for that, essentially it's a way for a website to specify to all web browsers that are aware of this header that the website's policy is absolutely no exceptions to using SSL connections.

So what that means is that any time you attempt to make a non-SSL connection, that site will redirect the browser to an SSL version of the page. When the browser then comes back with an SSL query, it gives it this strict transport security header, informing the browser, never again make an insecure connection to this domain. And what happens is, on the browser side, not only is the browser then told that there will be no exceptions, make no exceptions for certificate errors, but also transparently convert any non-https accesses to https. So, for example, if the user put in just www.paypal.com, or even http://paypal.com, if there is a Strict Transport Security token that has been received previously, the browser has permission to ignore that non-secure query request from the user and to transparently make it secure.

Now, there are some implications to this. One is that a site needs to be accessible, that is, all the resources that the site is offering, all the GIFs and JPGs and widgets and other things that are coming from that domain have to be available over SSL. So, for example...

Leo: So you use an image server. So you might not have an SSL image server.

Steve: Very good point. If you didn't have an SSL image server, then you'd need to provide a certificate for that. If that image server was a subdomain, for example, images.paypal.com, or in fact PayPal has something called PayPal Gadgets, I think is the domain where a lot of their stuff comes from.

Leo: Right.

Steve: There has been an attempt at sort of things like this, the EFF, the Electronic Frontier Foundation, in conjunction with the TOR project that we've talked about, The Onion Router project, they have created a Firefox extension called HTTPS Everywhere. And it's instructive that many sites fight against this. Many sites don't have, just can't be sort of, like, have every query turned into a secure query. And so what they've had to do, what the EFF has had to do with its HTTPS Everywhere extension is they've had to sort of craft a set of rules per site in order to figure out how to make the site operate as securely as possible.

So, for example, I just looked at their page this morning. They have managed to support Google Search; Wikipedia; Twitter; Facebook; most of Amazon, they say, meaning some parts of Amazon cannot be made to be secure. So their rule set has to be smart enough to know what to exclude. They support something called GMX, Wordpress.com blogs, The New York Times, the Washington Post, PayPal, their own site, the EFF site, TOR, and Ixquick are the domains that they have supported currently. They have an open protocol that allows users to add support for additional sites. And so that's an example of both an early attempt at doing this, recognizing the value of that, but also demonstrating the problems of doing so.

What I like so much about Strict Transport Security is that it's transparent to the user, completely. There is probably not a user interface on the browser because you don't want users to cavalierly delete these STS tokens which they receive from a site that wants

them. The one glitch in this is that, in order to get one of these STS tokens, which then ramps up the security, it may very well be that there is that first opportunity where you don't have a secure connection. If you first - most people are just going to go www.paypal.com or probably just PayPal.com, which if they didn't already, if their browser hadn't received a Strict Transport Security token in the past, the browser would make an http connection by default. PayPal's server would see that and immediately redirect them using a 301 response, telling them that that page has moved permanently to https://. At that point they would then - it would ask for that page, receive that token, and then they would be safe. The problem is that little window of opportunity, that first time they connect. If there was a bad guy, that's their opportunity to get in.

Leo: Would it be like a man-in-the-middle opportunity, or how would they...

Steve: Yeah. For example, they could simply prevent https from ever coming up and fool the user - in fact, we've talked about this kind of a problem before - fool the user into assuming PayPal is going to be secure. All the bad guy has to do is strip out https, strip out the "s" of everything that comes back. And, I mean, even the logon form, all they have to do is strip the "s" out of that, and then the logon is not secure because unfortunately, as I said, we're using a system that was never really designed to be robust from a security standpoint. And so if the bad guy got a foothold, if they were able to intercept that first query, then they simply filter the response and remove all notion of security from the response. The page looks the same to the user, they log in, and they're logging in over http rather than https, so the bad guy gets their login credentials.

Leo: Wow. But that we should say is - don't get scared. That's not easy to do.

Steve: No, no, no.

Leo: It's a nontrivial attack.

Steve: And again, all you would have to do is log in once, just go to PayPal once with https. PayPal sees that you're secure, and it gives you that Strict Transport Security tag. And then your browser knows, always elevate any...

Leo: From now on, okay.

Steve: I mean, not just for the page, but for all the assets, everything, anything going to that domain is going to be done over SSL.

Leo: Now, if you clear your cache, do you have to do it again? I mean, where is it storing that information? Is it a cookie?

Steve: No, well, it's funny. There has been some evolution of this over the last couple of years. The original proposal by those two guys that I mentioned at Stanford, Collin Jackson and Adam Barth, they presented in Beijing in early 2008, it was April 21-25 was

the big WWW 2008 conference in Beijing. They presented a paper, and they called it ForceHTTPS, where they were proposing that a special cookie could be delivered to the browser, where the browser would modify its behavior in that fashion. And so there has been evolution through this. The decision was made, though, as this thing - I should mention this is now a full-on IETF proposal for adoption. It's just happened in the last few months, so it's new, and it's going to no doubt take a while to happen. But the spec is solid. And as I mention, we're already seeing browsers supporting this.

The feeling was, cookies is not where you want to store this because it's too easy. There is a UI that allows people to delete their cookies. Many people delete their cookies out of habit because they know that they're tracking tokens. This technology cannot be used to track anyone because there is nothing being sent back from the browser, as is the case for cookies. It is a one-way policy statement from the server to the user saying we really are serious about security. No exceptions for certificate errors. No exceptions for connections. Only connect to us over SSL. And never accept any excuses from the security chain.

So, for example, in the case of NoScript, and I did this, if you look under, in the case of Windows, it's somewhere squirreled away on any platform. In Windows it's under Documents & Settings, and then the particular user you're logged in as, then under Application Data, under Mozilla, under Firefox, under Profiles, then there'll be a unique tag.default. And in that directory you will find a NoScriptSTS.db file. And mine had two entries, one from that secure site we mentioned last week, informaction.com, and PayPal. And it was funny because it's www.paypal.com, then there's a semicolon in this file, 1282142055. That's the number of seconds PayPal would like this to be honored. That's 1 billion, 282 million, 142 thousand, and 55 seconds.

Leo: I'm guessing that's the biggest number they can give you.

Steve: I got out my calculator. Actually, no, because secure.informaction.com is 1,439,000,000. So they're just choosing something.

Leo: Oh, okay.

Steve: Anyway, so I divided it down. It's just about exactly 40 years' worth of seconds. So PayPal is saying, from now, for any time foreseeable. Oh, and every time you're using PayPal it's sending you the same thing. So this is 40 years from the last time you accessed PayPal, not the first time you accessed PayPal. So this is a moving 40 years into the future that my browser, because I have NoScript, even though I don't yet have Firefox 4, my Firefox with NoScript will absolutely never connect me to PayPal except over SSL, no excuses. Which I think is tremendous security. And it's very clear to me that this is something that's going to catch fire and catch on very quickly.

Leo: Very interesting. So the browsers currently support it? It's something that - or do you have to update?

Steve: No, it's in NoScript, and it's on by default. If you had, for some reason, to, like, delete this, you could simply edit that .db file in NoScript. And Giorgio, who is the father of NoScript, talks about that in his blog. So you could make a change there. You can

also, he does allow you to disable it. If you put in about.config into the address bar, it's NoScript.sts.enabled, which is normally true. You can set it to false in order to turn this off if you had to. And in Chrome I don't think there's any UI. You're just protected. I mean, and that's - the idea is you don't want users to unintentionally bring this security down. The goal is for there to be no downside to it which would cause a problem for users.

Leo: So, cool. So I look forward to this. But really, though, it should not be built into a plug-in, but built into all browsers and then all sites.

Steve: Well, it's a plug-in only now because Firefox 4 is not out. It is in Firefox 4. It is already in Chrome.

Leo: Oh. Oh, that's neat.

Steve: So Google's Chrome browser has it already and is obeying it. And we users of NoScript, we have the benefit of it now since last September when Giorgio put it in. And clearly I have not been to a lot of sites that support it, or my little NoScriptSTS.db file would have a much longer list of domains. It only had two. PayPal is there, but PayPal is one of the early promoters of this. I think, as the word spreads - now, the fact that EFF, for example, could not get all of Amazon to be secure, it means that Amazon has some work to do in order to get the rest of their assets able to be served over SSL. And when they do that, it would certainly be in their best interest to add this policy to their outbound headers.

Leo: You bet, you bet.

Steve: And suddenly browsers, emerging browsers will be supporting this.

Leo: Is the primary reason that people don't do it, somebody like Amazon doesn't do it is because some of their assets come from non-SSL browsers?

Steve: Well, non-SSL servers.

Leo: I mean, not browsers, servers. Like it's mixed, they have mixed servers, and some can't handle it.

Steve: Yeah, now, what's interesting is that, for example, I learned while I was researching this something that I didn't realize, and that is that CSS files, which provide formatting for web pages, and Shockwave Flash files are not - the security of them is not enforced, such that, if you serve an unsecure Shockwave Flash or CSS file on a secure page, the user is not notified of mixed content. Which is interesting because it must have been that there was - the problem was that people were wanting to embed Shockwave Flash things, probably ads that were Shockwave Flash, and they were not coming over secure connections, and so too many errors were being generated by browsers. And so

unfortunately the browsers backed off and decided, oh, well, we'll make an exception to our mixed content warning because otherwise people are going to be getting these things all over the place.

Leo: Right, right.

Steve: So, yeah, so the problem is that sites are sometimes deliberately mixing content, but also one of - and this is in the spec. One of the things that the spec writers for this recognized is sometimes webmasters, web designers, mess up. A web can have its entire surface served over SSL. All it takes is one little mistake somewhere, for example, one mistake somewhere on a large site like PayPal, making a page insecure, would allow an attacker to gain a foothold because that would allow them to run a script, to inject a script onto that page and then get up to who knows what kind of monkey business.

So one of the other benefits of the Strict Transport Security system is it declares the website's intent to be 100 percent secure. And so it essentially solves the problem of mistakes in the delivery of insecure content on that site. No Strict Transport Security token will be obeyed over non-SSL. So what that does is that prevents a bad guy from, like, setting it to zero seconds of life, or one second of life. It prevents that. It also prevents a denial of service attack because imagine if the Strict Transport Security tag could be honored over just a regular non-encrypted connection, then a bad guy could add a tag for a site that doesn't support SSL. The browser would honor it and suddenly switch everything to SSL and refuse to do otherwise, and the user could never get to that site.

So this has been carefully thought out so that it simply works in the background. I mean, anyone who's been using PayPal with NoScript and Chrome, or NoScript on Firefox or Chrome, has already had this protecting them and didn't even know it. It causes no problems. It just really does a beautiful job of enforcing security in a very important way. As I was saying, there are so many ways that HTTP can let us down if we're not really, really careful about it. And this thing says, I mean, I see this as a major step forward in making security really work to a much greater degree than it has before.

Leo: As long as we're talking about browser security, I just want to mention - because I know we're all LastPass fans, and I was logging into LastPass today, and a serious security alert came up from LastPass. If you use Chrome, and you use one of the beta versions of Chrome, Chrome bug 52096 breaks the LastPass hashing code. So...

Steve: Oooh.

Leo: So I don't use the beta of Chrome, although I have in the past. There is, if you go...

Steve: Oh, so they've got a JavaScript glitch.

Leo: Yes.

Steve: Okay, yup.

Leo: And they even have, at rodan.lastpass.com, which is obviously a LastPass site, he has a demonstration that you can run to see if the Chrome Nightly bug will bite you. It gives you a hash. And if you get matching hash codes at the bottom, then you're okay. But if you get a zero in the incorrect result, then you are running a version of Chrome that will be a security flaw in LastPass. So I don't know, I guess that means that you're hashing - I don't know what the impact would be, but...

Steve: What it would mean is that, as we know, LastPass uses some technology, it uses hashes in order to authenticate its users. So they must have, somehow in their add-on or in their - it must be a bug in their add-on - in a library that the LastPass add-on is using, probably generating SHA-256 because that's what LastPass uses.

Leo: That's it. That's the one, yup.

Steve: Yup. And so Chrome must have introduced a bug in the hashing algorithm such that LastPass's use of it would generate a hash that no longer matches - remember that the way LastPass authenticates is that it stores hashes in its server, never the user's password or anything else. But that allows it to perform one-way authentication. So if suddenly it's getting different hashes from the browser, even though you've logged in correctly to it on your browser, the LastPass server would say, no, you've got the wrong login credentials. Try again. So it's not your fault, it's the browser's fault in this case.

Leo: I use the release version of Chrome, and it does not affect the release version of Chrome, just if you're using a nightly build, or a beta version. Thought I'd pass that along. Steve, a great subject. I look forward to STS being available everywhere. For those of us who use Chrome, we're ready. Firefox 4, you'll be ready, soon.

Steve: And we have NoScript doing it right now for us.

Leo: That's really awesome. Really awesome. If you want to get 16KB versions of this show, transcripts, show notes, the best place to go, Steve's site, GRC.com. You can watch us do this live on the TWiT network. It's live.twit.tv. We record Security Now! at 2:00 p.m. Eastern, 11:00 a.m. Pacific every Wednesday. That's 1800 UTC. Again, at live.TWiT.tv. You can also subscribe to the show, audio and video, at TWiT.tv/sn. And if you're looking for SpinRite, or any of the great free applications that Steve offers, GRC.com is the best place to go. Next week we're going to answer questions, so GRC.com/feedback will be the place to go if you have a question about this or any of the topics we cover on Security Now!. And Steve, I hope you have a great week.

Steve: Will do.

Leo: And we'll see you next time.

Steve: Talk to you for Episode 263 next week, Leo.

Leo: Well into our sixth year. Thanks, Steve.

Steve: Thanks, Leo.