Browser Security

Description: Steve and Leo discuss the broad topic of web browser security. They examine the implications of running "client-side" code in the form of interpreted scripting languages such as Java, JavaScript, and VBScript, and also the native object code contained within browser "plug-ins" including Microsoft's ActiveX. Steve outlines the "zone-based" security model used by IE and explains how he surfs with high security under IE, only "lowering his shields" to a website after he's had the chance to look around and decide that the site is trustworthy.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-038.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-038-lq.mp3

Leo Laporte: Bandwidth for Security Now! is provided by AOL Radio at AOL.com/podcasting.

This is Security Now! with Steve Gibson, Episode 38 for May 4, 2006: Browser Security.

Security Now! is brought to you by Astaro, makers of the Astaro Security Gateway, on the web at www.astaro.com.

So finally the sun is shining in beautiful California. We have summer, or at least spring finally arrived. And, look, Steve Gibson's wearing a tank top and shorts. No, he's not.

Steve Gibson: Well, it's dry in Toronto, Leo. You were all chapped; I was all, like, chapped just after one day.

Leo: I am. And, you know, they complain in Toronto that it's too humid because they have the lake effect. But I guess it gets humid some other time because I – chapped is the word, exactly. My cheeks are bright red.

Steve: Yeah, my face was all, like...

Leo: Yeah.

Steve: ...tight feeling.

Leo: Yeah. Isn't that funny?

Steve: It's like, ah, what's going on, yeah. We're just wusses down here in Southern Cal, in California.

Leo: When you have a perfect climate. Oh, now, I shouldn't rub it in. We have listeners all over the world, many of whom actually live in perfect climates, so I won't bug them. And those of you who don't, you already know.

So our topic today is something, I think, near and dear to everybody's heart. No matter who you are, you use a browser. And it seems that browsers are one of two really prime vectors for security flaws, the other being email.

Steve: Yeah. I think probably maybe browsers more than anything because the design of browsing on the web, the way it's evolved, is part of the problem. Whereas you could argue that email viruses were sort of a side effect of email security problems, I mean, the idea, for example, that Microsoft's design for ActiveX was to allow a website that you visited to, I mean, on purpose, to download what is basically a DLL and run it, just by visiting the page, without any asking for permission or anything. I mean, things have tightened up since this was originally created and this original concept. But, you know, that was the idea.

So one of the things that I want to draw a very clear distinction between are problems due to policy versus problems due to mistakes. And, I mean, this has sort of been a long-running theme because, for example, over in a different era, you know, over in Windows with open ports, my argument with Microsoft was that they were running servers and services that were opening ports and windows by policy. I mean, they were deliberate- they deliberately had filesharing running. They deliberately had DCOM and RPC and all these ports open, not mistakenly, but on purpose. And it was not until they finally got the clue and had XP's firewall running by default in Service Pack 2 that those open ports were then hidden behind the firewall, so they were essentially closed to the outside world. So, you know, anyone can make a mistake, and of course Microsoft is famous for making mistakes. But it was the fact that they had this policy that essentially exposed their mistakes to the world. And this is very much the same problem we see over on the browser side.

Leo: So it really goes hand in hand. I guess you could claim that the policy was a mistake, but it was such a global mistake that we're still suffering years later, and it makes it difficult to fix. Whereas a security hole you can usually patch and fix and it goes away.

Steve: Well, let's – okay. Exactly. Let's stand back now a little bit and sort of, like, turn back the clock and remember that the original concept of the web were static, unchanging HTML text pages that browsers would download from the server, and those pages would then go fetch other images which would populate the page if it was, you know, a graphic web browser. And that was the – that was Tim Berners-Lee, you know, original concept for the web, was this distributed database called the Internet of servers that would serve pages, and then these pages would be linked to each other and so people could click on links to go to other pages. But the page itself was just static. It was a textual, read-only page that users would browse, and then they'd click on links to go to something else.

Leo: Actually, I think Tim in his original conception wanted them to be editable. But you're right, I mean, it very quickly became just these static pages. I think his original plan was

to have pages be collaborated on; and of course the immediate security issues became obvious, and everybody said, oh, no, maybe they'd better just be static.

Steve: Well, we didn't have collaboration technology. Maybe the idea was, like, to FTP to the server.

Leo: No, the original, believe it or not, the original web browser that he wrote on the NeXT box, that was the idea, he wanted other physicists to be able to modify pages and collaborate on documents. But you're right, I mean, that wasn't – that never left CERN, I don't think.

Steve: Right. Well, and in fact the original concept was from the server to the browser only, and the addition of forms that allowed web pages to send information back, you could tell by the nature of the way it was implemented, it was kind of a kluge that was stuck on afterwards.

Leo: You're talking about the Get and the Put and all of that?

Steve: Exactly, where for example you make a query to the server, and then you put a question mark in the URL and then show the parameters right there, which is, you know, how most search systems work today, where you're able to create a URL which contains the search parameters in the link text. This whole notion of getting data back to the server was sort of an afterthought and not something that was originally conceived.

So, moving forward, I'm not clear on the timing of where Netscape was with their scripting and where Microsoft was. You know, Microsoft tried to give us, you know, VBScript in the browsers. But then ultimately Microsoft came along and said, hey, let's not limit users to running JavaScript in browsers. We want this to be more powerful, more feature packed. So they took this whole notion of OLE, which is their Object Linking and Embedding technology which sort of had been limping along for a number of years, and somewhere along the way they renamed it. It's the same thing, they just renamed it ActiveX because that sounds fancy, you know, sort of like DirectX and Xbox and all that. And what ActiveX is, is it's just a DLL, you know, a compiled program which, in the case of a browser application, is able to download this code into the user's system and run it. And, I mean, it's very powerful. But of course it's horribly insecure.

Leo: I think you can blame Microsoft for this because the original scripting implementation by Netscape, which later became JavaScript, really wasn't very powerful. I mean, the browser could do things, but it couldn't actively access your computer, your hard drive. It wasn't really a full-blown programming language. It wasn't till Microsoft came along that you really had that.

Steve: Right, right. And that, of course, was by design. And in fact we know that the related language, Java itself, real Java, from day one it was created with the notion of a sandbox, that is, an enclosure. And so Java code was also – the idea was you would be able to download Java code and have a so-called "Java runtime," which would interpret the code so that you weren't actually running object code on the processor. You were running in an interpreted environment that could constrain what Java was able to do.

Leo: And that's still the case today. I mean, there are malicious Java applets, but nothing

on the order of the ActiveX applets.

Steve: Well, exactly. And so the distinction here to draw is that Sun and Netscape, who did Java and JavaScript, I mean, they always understood the need for constraint around what it was that a remote web server would be able to give the user. Microsoft didn't. I mean, Microsoft said, wow, this will be really powerful if we allow websites to download code. I mean, this is object code. This is unrestrained code that can do anything on the user's machine.

Leo: Anything any other program could do. And of course the success of Microsoft's approach was that everybody immediately started using ActiveX and said, forget Java, forget JavaScript, look how much more we can do with ActiveX.

Steve: Well, and in fact the classic example of the exploitation of this is the user experience of getting spyware installed on their machine, or various kinds of malware, just by visiting a website. You know, how many people were having problems with Gator and with weird sort of third-party search things that suddenly just appeared in their browser. They didn't do anything to install them. They didn't give permission. They just were out surfing around the web and were picking up this stuff because they were using IE, which was leveraging this ActiveX technology, which is, you know, very powerful. I mean, it's the way Flash is able to provide very sophisticated animations and substantial power is through this whole ActiveX plug-in system that Microsoft has. But the problem is, it was not designed with any sort of restraint built in.

Leo: Now, of course Flash works without ActiveX. But is Flash as dangerous as active scripting and ActiveX?

Steve: Well, Flash can do a lot with your system.

Leo: But it's more sandboxed, I think.

Steve: Yes.

Leo: Yeah. Can you describe a little bit about what Active Scripting, which was this former VBScript that you talked about, and ActiveX, the difference between the two, and the different capabilities of the two?

Steve: Well, the idea, in any kind of a scripting mode, whether it's VBScript that's Active Script or JavaScript, which was what Netscape's scripting language ended up being named, even though it really bears no relation to the Java language itself that...

Leo: Just a clever marketing ploy on Netscape's part.

Steve: Exactly. Java was sort of like the cool acronym about, what, eight or nine years ago. Any of these scripting things, the idea is that the textual content of the page is itself, within sort of some specified constraints on the page, is code which is read by your browser and executed. And so, you know, many things are made possible. I mean, the positive side benefit

of scripting is that pages can be intelligent themselves. For example, you could fill in a zip code, and the page would know what state that was in, so it would populate the state field for you if it knew – because it could, in the page, it could have code that knew the zip code ranges by state and just filled that in for you. And...

Leo: Although it would be good here to kind of make the distinction between server-side scripting and client-side scripting. So when you visit, for instance, ThisWeekInTech.com, or many pages, including your own page, Steve, because you use server-side scripts, programs are running on the server, and oftentimes there are zip code programs running on the server. But you can also have code that runs on my computer as a visitor, and that would be JavaScript or Active Script. Or Active X. Those are running on my side.

Steve: I think that's a really good distinction to make. For example, you know, my own ShieldsUP! facility is – it's code running on the server. And all I'm delivering to the user, despite the fact that they're getting a customized page showing which ports they have open and closed, all I'm delivering is absolutely generic HTML, you know, like dead code, or dead text that the browser renders. So that's..

Leo: That's really important because there's really no security implications on a server-side script, at least not to you. To me running the server there might be, but from a user's point of view it's safe.

Steve: Right. Server-side, as it sounds, is code running on the server, not on the client, not on the user's browser. So it's client-side code, or browser-side code, which is where we have problems because, in that model, the page that's delivered that contains textual content also contains interpreted script, meaning that it is literally – it's code that you can, if you view the source of the page and focus your eyes after looking at what some of this gobbledygook looks like, I mean, it is actual code which your browser reads and runs there on your computer. And again, that's where many of these cool features that we're used to seeing now on modern websites come from is – well, I mean, for example, many of the menu navigation systems where you float your mouse around and things pop up and open, and you're able to have a really nice navigational experience, that's done because you've got a bunch of code running in your browser right there without needing to go back and get updates and new stuff from the server all the time. So it's very real-time, it's very quick to operate. And even somebody over a slow modem, once they get that code loaded into their browser, then that page is able to do all this stuff locally by itself. And of course the disadvantage of that is security because we're constantly running across problems. Now, these are not policy problems. These are implementation problems.

And so that's the distinction I want to make is that, in general, it's scary to accept code from a remote server, whether it's script code or, even worse, an ActiveX object that could be – is literally running native code with no restraint at all. I mean, you really need to trust the source of ActiveX because it's, you know, you're literally running code that you've just received from the server. In the scripting side there's been an attempt to constrain what the scripts can do. But here we have the problem of mistakes being made.

In fact, you may have heard recently, Leo, there was a remote code execution vulnerability found in Firefox. You know, Firefox is a browser that I know you prefer because, due to its policies, it's substantially more secure than IE, for example, not running ActiveX controls that we've been talking about by default. But still, even so, these mistakes in the code still create vulnerabilities that people find from time to time and are able to exploit. And in fact there's exploit code on the 'Net now for this Firefox vulnerability. So far it's not running code, but it'll crash your browser if you just click this link because it takes advantage of a coding error in the scripting stuff.

Leo: Yeah, and that really is kind of a leak in the way that Firefox was designed, so that this stuff leaks into the actual application Firefox and then does things.

Steve: Well, and in fact, what we're going to talk about next week in detail is what are buffer overruns. Because anyone who's involved in security, I mean, even very peripherally as a user who's concerned, is hearing "buffer overrun, buffer overrun, buffer overrun" all the time. That's like the – it's the golden goose for the malicious hacker. And I want to talk in detail about what exactly a buffer overrun is, what is it that's overrun, why does it seem to be such a problem to control, and why is it, like, the continual vector for all the problems people are having.

Leo: But again, that's different. That's taking advantage of or exploiting a flaw in the programming. And the stuff we're talking about today is stuff that actually takes advantage of something that was intentionally done...

Steve: Right.

Leo: ...in the browser. This is, as you say, a policy mistake, not a programming mistake.

Steve: Right. So, I mean, in general the idea of accepting code, whether it's script code or it's an actual executable file, which is what ActiveX is, the idea of accepting code is dangerous. And so it's an unfortunate fact of modern life using the web that, in order to have the kind of experience that many websites want to give their visitors, code and the risk of code is what comes along.

Leo: Sure. Well, if you think about it, creating a web-based antivirus is pretty difficult to do unless the program can run on your system and can access your hard drive and examine your files.

Steve: You mean like a web-based AV scanner.

Leo: Like HouseCall, yeah, like HouseCall.

Steve: Right.

Leo: In fact, I understand they've been trying to write a Java version of HouseCall, but I haven't seen it yet, and I think it's because it's very, very difficult to do.

Steve: Well, it's probably hard to get out of the Java sandbox onto the machine.

Leo: Exactly.

Steve: And, you know, something like GoToMyPC, a browser-based remote access system, I mean, that's a very powerful concept. But there your browser is the client, which is running code on your machine in order to provide substantial features. So in every case we have this

tradeoff between the power to do what we want to do and the inherent lack of security that using that power brings along.

Now, Microsoft did something interesting in IE which, as you know, Leo, I take great advantage of, and that's this notion of zones. They have security setting controls, like all browsers do, for things you want to turn on and turn off. But what Microsoft's done, in order to try to create some sort of control around this otherwise arguably too powerful facility called ActiveX, is they've created this notion of security zones. If a web domain like GRC.com or Microsoft.com or TWiT or whatever is not otherwise defined as, like, good or bad, it's in the default Internet zone, meaning that it's treated just like any other site on the Internet. So you're able...

Leo: So your settings that you have set to general settings apply to all of those sites.

Steve: Exactly.

Leo: Whatever your security setting is for that.

Steve: Exactly. And so the way I actually surf the 'Net is I've got my Internet zone cranked to the highest security possible in IE. What that does is it basically just...

Leo: Breaks everything.

Steve: Yes, it does. It shuts down everything. I will not run scripting of any sort. I will not run ActiveX controls. Basically, you know, nothing is working.

Leo: Just out of curiosity, what percentage of sites that you go to with those settings work?

Steve: About 99 percent.

Leo: Really.

Steve: It's really the case, Leo, that a lot of sites, I mean, some of them aren't as fancy as they might be. You know, well-designed sites know when you're not running their script. And so they'll, like, fall back to something that still works. One advantage is that I don't have Flash ads, you know, jumping around, flying across the screen, you know, running through their animation cycle, because Flash doesn't run if security is cranked all the way up. So most sites will work. But there are, I mean, for example, I can use Amazon because Amazon doesn't use client-side technology. All of its stuff is server-side. So they're just providing static pages to browsers. I think eBay also works just fine.

You know, many things do work with security cranked all the way up. But every so often you'll run across a site that does need security. For example, many times when I'm doing ecommerce, if I have some ecommerce, they're using script to do something like populate forms for you, where, for example, you'll have a shipping address and a billing address. And so you'll fill the shipping in first, and then you'll click a button to say the shipping and the billing are the same. And when you do that, the form...

Leo: It fills in the form, yeah.

Steve: Exactly. So that's all happening on the client side.

Leo: That's usually JavaScript when that happens.

Steve: Well, exactly. And it won't work if, for example, in IE, if the security is set all the way to the max. So when I...

Leo: But then there are other things, like Windows Update, that won't work.

Steve: Absolutely.

Leo: Or that antivirus, they won't work either.

Steve: Absolutely.

Leo: So there's a broad range of stuff that doesn't work. What happens then?

Steve: So what I do is, IE has not only the default zone, which is the so-called "Internet Zone," but it's got this notion of a trusted zone. So you can have separate and completely different security settings for the trusted zone. And then you're able to, by URL, you're able to put any sites that you trust not to hurt you into your trusted zone. So, for example, I've got in there *.microsoft.com, *.ecommercerus.com, you know, whatever sites that I've decided, essentially, to lower my security for.

Leo: And that uses certificates, right, for identity. So you don't...

Steve: No no no.

Leo: ...have to worry about somebody spoofing that – well, what if you got a phishing email, and they spoofed Microsoft.com?

Steve: That's certainly a concern. The browser – I'm not sure...

Leo: It must be certificate based.

Steve: No, I'm sure it's not, Leo, because it's...

Leo: It's just the address, then.

Steve: It's just a URL. So the browser knows if it pulls the domain name and then has looked up, using DNS, looked up the IP and gone there.

Leo: I guess it'd be hard to fake that.

Steve: I think it would be difficult to fake that. I mean, I'm not going to say it's not possible. But in general I think it's done pretty securely. So essentially...

Leo: It would take some sort of man-in-the-middle thing to do that.

Steve: So essentially what this does is it allows you to have different security settings when you're just roaming about the Internet, which is where you want very high security; and then, for the sites that you visit from time to time, that you trust, you have security that has been lowered if it turns out that you need to. And so, I mean, it's really, frankly, it's the optimal way to be secure on the Internet is to be running nothing, I mean, no JavaScript, nothing. You know, many times when these alerts come out, their workaround until patches are available is turn off scripting. Well, I've already got mine turned off. I mean, it's off for all the sites that I don't already know I care about and I either need to or I want to enable those advanced features. So, I mean, I'm a real fan of this approach.

Leo: And if you do do it, you're very secure. I'm a little more cynical. I would say this is just Microsoft's way of deferring responsibility. Now they can say, well, if you get bit, it's only because you didn't use security properly. And I would say – I would bet 99 percent of Internet Explorer users don't even know they have this capability, let alone use it.

Steve: Right. It's built in to versions of IE for the last three or four major versions. It's certainly been in 5 and is in 6 now.

Leo: And in fact works better in Service Pack 2. They beefed up the local – they changed how the Local Zone is handled to be even more secure. So in theory it's a good idea. Do you think this solves the issue, this solves the security problem?

Steve: Well, certainly it transfers responsibility to the user, which is a burdensome thing to do.

Leo: My point exactly.

Steve: Yeah, I mean, it does. But certainly – okay. The best possible solution would be that you would never have a website that would try to give you a malicious ActiveX control, and that the scripting technology would have no workarounds or mistakes or glitches that malicious websites could take advantage of in order to get control of your browser. Unfortunately, neither of those are anything that a security-conscious user could rely on.

Leo: And in fact, if you've got code, you've got insecurity. So you're just going to have – this is never going to happen.

Steve: Well, yeah. And it's not really fair to blame the browser because, I mean, look at all the problems we have just with the OS itself.

Leo: Right.

Steve: So I guess my point is that the reason the browser is a focus point is it's out there on the frontline. I mean, you're literally, when you click a link, you are sending your browser off to somewhere it has never been before, maybe you've never been before. And it's like, good luck!

Leo: Cross your fingers. Hope you're okay.

Steve: Hold your breath. Yeah, I mean, in reality – that analogy is fun. But of course what you're really doing when you click the link is you are asking a server Lord knows where to send you a blob of stuff which your browser, running on your computer – I mean, maybe this is even more scary, it's the actual truth of what's happening, is you're saying, you know, let me have it. And so this blob of code comes down into your computer, and your browser says, okay, here we go. And with the best of intentions and in good faith it starts reading this page, displaying things and running scripts which may be included in that code. And who knows what's going to happen? And so it's not that the browser is worse code than any other code we have. It's just that it's the frontline. It's right out there as you surf the 'Net, getting pounded with the stuff from any website that you visit. So from my way of thinking, I want my shields up. I mean...

Leo: So to speak.

Steve: Yeah. I want my shields up while I'm up poking around the 'Net. And so nothing ever infects me. When I'm going to sites I haven't been before, mostly they work. If I decide that, okay, I want a better relationship with this site, you know, I'm on eVitamins.com or something, and I decide, okay, I want to buy some stuff. Well, then I will go over and I will put *.evitamins.com into my trusted sites list because this is, like, the real, legitimate site. But I have been able to sort of – I've been able to interview it first. I went there with my shields up; I poked around. If I decide I don't want to go there, then, you know, it's had no chance to affect me in any way because my browser won't do anything for this website.

Leo: That's a good way to do it.

Steve: Yeah, again, it's more burdensome for the user because your shields are up by default, and then you lower them only where you need to. So it's proven to be a fantastic way to work for me. And again, as you say, Leo, unfortunately there's this substantial power in IE, but most people don't understand it or know what it's about.

Leo: Right. Hey, I want to give you a little bit of credit. You won't get the credit probably elsewhere, but you deserve it. It was not so many years ago that you banged the gong about raw sockets in Windows XP. And of course about a year ago Microsoft finally disabled

that raw socket capability. And one of the reasons you didn't like raw sockets is it made that possible for anybody who co-opted an XP computer to spoof their attacks, to use the raw sockets capability to put a random, nonexistent IP address in the packet that they were sending out.

Steve: Right.

Leo: Well, there is a project going on, I just saw this on Slashdot, called the Spoofer Project, that's attempting to measure how many ISPs or providers of any kind allow spoofing to occur. As you've pointed out, all it takes is a little bit of due diligence on the part of the Internet Service Provider, and spoof packets aren't really a problem. And apparently, according to the Spoofer Project, they've only done a few thousand measurements, about 80 percent of the IP addresses they've measured don't support spoofing anymore. The ISPs have sat up, taken note, and prevented IP address spoofing.

Steve: Very nice.

Leo: That's really good news. And I think you deserve credit for it because I don't think anybody would have thought of this as a security issue, would have known about it, if you hadn't raised a ruckus about raw sockets those years ago.

Steve: Well, the big problem, of course, are not individual end-users who are doing this, but there are people whose computers are affected by these remote-control IRC bots.

Leo: Right. We're not worrying about Granny using raw sockets. But we're worried about her computer being used by a bad guy and raw sockets.

Steve: Exactly. And then attacks being launched from her computer. And having the advantage of raw sockets means that it's easier to produce a denial-of-service attack on other machines, which is virtually impossible, I mean, practically impossible to trace back. So I was really...

Leo: I think it's good news.

Steve: Yeah, I was really glad to see that Microsoft, you know, finally woke up with Service Pack 2 and removed raw sockets. And it's very cool that Slashdot's doing this. I had a plan that I never got back to. The idea was going to be a little gismo called Spoofarino, was a piece of freeware that I was going to do that would allow people to determine for themselves whether their ISP allowed spoofing outbound. That, of course, became much more difficult once XP Service Pack 2 happened because it would be much harder to generate such a packet.

Leo: So because I can run as root as my OS X system, and Linux users can, as well – in fact, you can even do this on Windows if you run this little software – it's actually not Slashdot. They have the story. But it's from MIT. It's called the Spoofer Project: spoofer.csail.mit.edu. I'll put a link in the show notes. But I think what's encouraging, whether you run it or not, is that so many ISPs have heard the call and are blocking spoofed packets.

Steve: Well, and I think they don't want to be hosts for these bots, either.

Leo: Yes, right.

Steve: So if they block – I mean, it's so easy for an ISP to know if the apparent source IP of a packet that's leaving their network could have originated in their network. I mean, is it one of their IPs or not? And if it's not, just drop it.

Leo: Right.

Steve: And so suddenly no machines in their network, within their entire network, can be used to generate spoofed attacks, and they stop being a useful source for those bots.

Leo: So one little small victory, anyway.

Steve: That's neat.

Leo: Many more battles to fight. Next week we'll talk about really a bigger battle. You can talk about securing your browser, and you can talk about better security policies from operating system companies. But one thing we'll never be rid of is bugs. And we'll talk about common bugs...

Steve: Buffer overruns.

Leo: Buffer overruns. This is the big one. How they happen and what they mean.

Steve: And why they're so darn hard to fix and to prevent, yup.

Leo: And as you know, security is important to us. That's why we are very happy to have Astaro Corporation as our partner on this podcast. They're our great sponsor at Astaro.com. Make sure you check out the Astaro Security Gateway, a free software for home users that you can run to protect yourself. And of course for businesses and people like me who want to be extra careful and extra protected for a very reasonable cost, the open source Astaro gateways are really the way to go. I love my 120. I'm really happy I've got it.

So, Steve, we'll be back next week, talking about browser and buffer overflows and all those nasty bugs. I hope you have a wonderful week and enjoy the sunshine.

Steve: Talk to you then, Leo.

Leo: Next Thursday, Security Now!.