SECURITY NOW!

Transcript of Episode #221

The Oxymoron of "JavaScript Security"

Description: This week Steve and Leo are joined by author and software developer John Graham-Cumming to discuss many specific concerns about the inherent, designed-in, insecurity of our browser's JavaScript scripting language. Now 14 years old, JavaScript was never meant for today's high-demand Internet environment - and it's having problems.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-221.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-221-lq.mp3

Leo Laporte: It's time for Security Now!, the show that covers all things secure and private and important, like that. With us right now, the king of security, Mr. Steve Gibson from GRC.com. Hello, Steve.

Steve Gibson: Hey, Leo. Great to be back with you again for our 221st episode of Security Now!.

Leo: Unbelievable. We have a good show this week, too.

Steve: We've got a great show this week, absolutely. We have a - we don't often do guests on the show. But a friend of both of ours, John Graham-Cumming, whom we've mentioned a number of times, and of course he is the author of "The Geek Atlas," which we've talked about and both like very much. He's going to be on because he did a - he gave a presentation to a recent virus conference. The title was "JavaScript Must Die."

Leo: Hmm. This is right after your own heart, isn't it.

Steve: Believe me, this is - yes, exactly. I've understood in broad strokes what the problems are. John is going to tell us in painful detail why 14-year-old JavaScript just really doesn't cut it anymore. And his thesis, that we'll discuss in a minute, is that there's probably no way to fix it.

Leo: Wow.

Steve: Yeah.

Leo: Well, we're going to get him on in just a minute. I'm sure there's some security news in the hopper. So, Steve, what's been happening in the week since we talked last?

Steve: Well, we've got a little bit of news. It's been relatively quiet. One thing that I wanted to do, though, which sort of sets us up for today's topic, is we talked about the most recent set of security patches for Firefox, and also for SeaMonkey since they use a bunch of common code. And Firefox 3.0 and 3.5 we talked about last week. And I just sort of quickly glossed over those things. But I came back and looked more closely at what went on. And I thought, you know, it's one thing just to say, oh, update Firefox. But it's interesting, I think, every so often to just say, okay, wait a minute. Let's just take a look at what it was that happened in this one, just one of a continual procession of security fixes because it's very educational relative to today's topic of JavaScript. And also you and I were not really sparring a little bit, but you were saying, wait a minute, you're still on 3.0? You haven't gone to 3.5?

Leo: Right.

Steve: So that sort of folds into this, too. So there were 10 different things, major things that were fixed. There was a form history vulnerability that would allow form history to be stolen. A security researcher, Paul Stone, reported that a user's form history, both from web content as well as the smart location bar, was vulnerable to theft, so that a malicious web page could synthesize events such as mouse focus and key presses on behalf of the victim and trick the browser into auto-filling-in forms which it would then send to the server. So without your interaction. And that's one of the things that John will be talking about today, is that JavaScript provides no differentiation between things that it does and things the user does. So JavaScript is able to perform on your behalf, which is convenient, but also creates vulnerabilities like this one.

The second problem was a crash with recursive web-worker calls. Security research Orlando Berrera of Security Theory reported that recursive creation of JavaScript web-workers can be used to create a set of objects whose memory could be freed prior to their use. Okay, so you're creating a bunch of things, then you're freeing their memory before you use them. Well, that's not good because then you've got a memory problem. These conditions often result in a crash, which could potentially be used by an attacker to run arbitrary code on a victim's computer. And so this text is coming from the Mozilla site. And so their site said, "Note: Web-workers were introduced in Firefox 3.5." So this vulnerability did not affect earlier releases such as Firefox 3.0.

The third problem, a crash in Proxy Auto-configuration regular expression - "regex" - parsing. Security researcher Marco C. reported a flaw in the parsing of regular expressions used in Proxy Auto-configuration files. In certain cases this flaw could be used by an attacker to crash a victim's browser and run arbitrary code on their computer. Since this vulnerability requires the victim to have a PAC, a Proxy Auto-configuration file, configured in their environment with specific regular expressions which can trigger the crash, the severity of this issue was determined only to be moderate. And the workaround was to disable JavaScript. Number four...

Leo: Of course, what else.

Steve: Of course. There's a heap buffer overflow, remember we talked about it, in the GIF color image map parser. In this case the security research firm iDefense reported a heap-based buffer overflow in Mozilla's GIF image parser. This vulnerability could potentially be used by an attacker to crash a victim's browser and run arbitrary code on their computer.

Number five, a Chrome privilege escalation. A Mozilla researcher reported that the XPCOM utility unwrapped doubly wrapped objects before returning them to Chrome callers, Chrome being not like Google's Chrome browser, but internal technology that Mozilla also happened to coincidentally call Chrome. This could result in Chrome privileged code calling methods on an object which had previously been created or modified by web content, potentially executing malicious JavaScript code with full Chrome privileges. The workaround for Mozilla? Disable JavaScript.

Number six, a heap buffer overflow in string-to-number conversion that we've seen a couple times and talked about. In this case a security researcher, Alan Rad Pop of Secunia Research, reported a heap-based buffer overflow in Mozilla's string-to-floating point number conversion routines. Using this vulnerability, an attacker could craft some malicious JavaScript code containing a very long string to be then converted to a floating point number by the user's browser, which would result in improper memory allocation and the execution of an arbitrary memory location. This vulnerability could thus be leveraged by the attacker to run arbitrary code on a victim's computer. Mozilla says, workaround? Disable JavaScript.

Number seven, cross-origin data theft through document.getselection. And John will be talking about cross-origin problems in JavaScript. Security researcher Gregory Fleischer reported that text within a selection on a web page can be read by JavaScript in a different domain using document.getselection function. Now, these are supposed to - the origins of scripting is supposed to be separate. But there's all kinds of problems with that not being done right that creates cross-origin, sort of like cross-server, leakage. This violates the same origin policy. Since this vulnerability requires user interaction to exploit, its severity was determined to be moderate, not severe.

Three to go. Download filename spoofing with RTL, that's right-to-left override. Mozilla security researchers Jesse Ruderman and Sid Stamm reported that when downloading a file containing a right-to-left override character in the filename, the name displayed in the dialogue title bar conflicts with the name of the file shown in the dialogue body. An attacker could use this vulnerability to obfuscate the name and file extension of a file to be downloaded and opened, potentially causing a user to run an executable file when they expected to open a non-executable file. So that could kind of catch out people who think they're more savvy and professional and know what's going on, like checking the filenames. Except whoops, due to the way this is handled, it's possible to obfuscate the actual name of the file that you're downloading.

Number nine, we talked about the media libraries being upgraded. It was called an upgrade, the media libraries, to fix memory safety bugs. Mozilla upgraded several third-party libraries used in media rendering to address multiple memory safety and stability bugs identified by members of the Mozilla community. Some of the bugs discovered could potentially be used by an attacker to crash a victim's browser and execute arbitrary code on their computer. The liboggz, libvorbis, and liboggplay libraries were all upgraded to address these issues. And then on the Mozilla site it said, "Note: Audio and video capabilities were added in Firefox 3.5." So prior releases of Firefox, such as 3.0, were not affected.

And finally, crashes with evidence of memory corruption. I remember when we talked about this last week it's like, well, that seems sort of ambiguous. But specifically it said that Mozilla developers and community members identified and fixed several stability bugs in the browser engine used in Firefox and other Mozilla-based products. Some of these crashes showed evidence of memory corruption under certain circumstances. And we presume that with enough effort at least some of these could be exploited to run arbitrary code. Mozilla's workaround? Disable JavaScript. So...

Leo: Now, how much of that is a problem with JavaScript, and how much of it is just a workaround that prevents people from accessing these other problems which aren't really JavaScript problems but can be addressed through JavaScript?

Steve: That's a very good point. These are typically - these are problems in the browser which JavaScript is used to get access to. So JavaScript is the means to using these mistakes. It's the mistakes themselves that are the problem. Now, what John is going to be talking about is different than this. But what I wanted to highlight here is that there are new things that were added to 3.5 which are the sources of new problems. So we've talked again, as we often do, the idea that anything new has a problem because it hasn't been proven. It's just it's so difficult to design code securely that anything that you do, there's just going to be a certain percentage of the code that's going to have a problem. So...

Leo: Although as we've seen, everything old has problems, too. It's not like, I mean, it's not - they're just a new set of problems.

Steve: Yes. Well, okay, yes.

Leo: I mean, it's not like old code is somehow magically better code.

Steve: I would disagree, Leo. I think older code is better code. Older code has had this kind of stuff pounded out of it. But what we need is...

Leo: Well, that's the question, is does it get pounded out, or does it just get pounded? I mean, you can't say XP is safer because it's had many, many, many, many patches. Because we keep finding holes in it.

Steve: Well, it's because people - no one leaves it alone. We're still - it's not a static operating system. Microsoft keeps messing with it. 98 is old, but solid, and isn't prone to being affected now. I really do believe older code is better code than new code. However, it's got to be old code that you leave alone, that you don't keep messing with. And so XP continues to be brought forward because Microsoft's supporting it, and features that are being added in Vista and Windows 7 are still being backported into XP, which is destabilizing it.

Leo: True. True, true, true.

Steve: So it is significant, though. Relative to Firefox 3.0 and 3.5, I wanted to mention to people that Mozilla has formally stated they are not going to be continuing to support Firefox 3.0, the Firefox 3.0 series, after January of 2010. So only November and December, two more months of support for Firefox 3.0.

Leo: Really. That's all?

Steve: Yes.

Leo: Wow. That's pretty quick.

Steve: Yeah. That does strike me as being quick. I wanted to mention to people using the Opera browser to check for updates. They're now at version 10.01 with a bunch of vulnerabilities. I won't go into them in detail. But there's a set of vulnerabilities. They're publicly known and available. So if you're an Opera user you definitely want to make sure you stay current there.

And then one bizarre bit of news. I picked up on this on the Discovery Channel news. Their security editor, Eric Bland, on the 28th posted an article that just captured my imagination. The title was "[Digital 'Ants' Take on Computer Worms]." "Digital ants could soon be crawling through your computer's hard drive, but don't worry, they are there to help." And, okay, this is just too fun.

"Scientists from Wake Forest University and the Pacific Northwest National Laboratory have created an army of digital ants and their superior officers, digital sergeants and sentinels, to search out viruses, worms and other malware. The new antivirus software could provide better protection while freeing up valuable hardware.

"'We are using the ants to sense something very basic, like a connection rate,' said Errin Fulp, a professor of computer science at Wake Forest University who helped develop the digital ants. 'Then we collect that evidence which points us to a particular infection or security threat,' said Fulp.

"Like their biological counterparts, each individual ant is not very bright. A connection rate, CPU utilization, or one of about 60 other technical details is all they can sense. When an ant detects something unusual, it leaves a digital pheromone, a tiny digital sense that says something" - oh, this is wacky.

Leo: This analogy's gone a little too far.

Steve: "...a tiny digital sense that says something unusual is going on here, and that other ants should check it out. The digital ants report any suspicious activity to a digital sentinel, a program designed to watch over a set of computers in a network. The sentinel sorts through all the information the ants gather and, if it's suspicious, passes the information on to a digital sergeant. The sergeant then alerts the human supervisor, who can deal with the problem. The sentinels and sergeants reward the ants for finding problems. If an ant doesn't find enough problems, it 'dies' off, although a minimum number is always maintained.

"If a particular kind of ant finds lots of problems, then more of them are created to monitor the problem. The entire system is modeled off of a normal ant colony and uses 'swarm intelligence' to find and diagnose problems. The beauty of using digital ants, instead of a traditional antivirus program, is their flexibility. Traditional antivirus software usually scans constantly or on a set time schedule. Constantly scanning for threats is effective, but uses a lot of computer resources, resources that could be better spent doing something else.

"Scanning at certain times, usually at night, optimizes computer usage, but it leaves a computer more vulnerable [in the interim]. Since the number of ants rises and falls with number of problems being detected, it can free up computer hardware to perform calculations when an attack isn't happening. If an attack is happening, more ants can quickly be created to help deal with it." So there you go, Leo.

Leo: Digital ants.

Steve: We're going to have ants crawling around in our computers.

Leo: Well, it's funny because John Graham-Cumming calls JavaScript the "elephant in your browser." So this is really kind of a menagerie of problems here.

Steve: And I have to say, all of this starts making my PDP-8 computer look...

Leo: Look better and better.

Steve: ...pretty good.

Leo: Hey, if you never get on the Internet, you'll never have a problem.

Steve: That is, that is the case.

Leo: Well, maybe never. But you'll have fewer problems. What else we got before we - should I get John on, or do you want to cover some errata first?

Steve: I'm not going to hold him up for long. I have a brief errata and a short little SpinRite story.

Leo: Okey-dokey.

Steve: My errata is titled "I hate Adobe."

Leo: That's not an errata. That's a facta.

Steve: Okay. Get this, Leo. I haven't had it happen a second time because I haven't used another machine. But Adobe's Flash updater, with no apparent ability for me to stop it, installed a demo of NaturalReader.

Leo: Oh, no, that's not right.

Steve: Now, there was a checkbox. It also wanted to give me a free MacAfee security scan.

Leo: No.

Steve: And I said no, thank you. But then it says it's - I'm watching it update itself, and it says it's installing a demo of NaturalReader. Well, first of all, I own NaturalReader.

Leo: Oh, great.

Steve: I use the voice for various things. So, and sure enough, on my desktop now - or was, I deleted it - but there didn't seem to be a way of uninstalling the demo. It didn't leave...

Leo: Oh, that's inexcusable.

Steve: Didn't put something in the Add/Remove Programs. There was an icon it added to my desktop. When I clicked it, and I made sure this was all legit, it popped open. It was a web page that was running NaturalReader from my system. And it's like, okay, wait a minute. I mean, so this is nothing to do with Adobe Flash. And this thing is installing demoware that I couldn't see any way to avoid. I mean, this is really wrong. So...

Leo: What was the updater? Was it your Adobe Reader updater? I mean, what was...

Steve: It was the...

Leo: The Flash updater?

Steve: It was the Adobe Flash updater.

Leo: God.

Steve: And so I just, I wanted to put out a call. I'm sure if other listeners had this happen to them, drop me a note at GRC.com/feedback. Let me know if you saw the same thing. This is just - at this point it happened to me. I haven't seen anything more about it. But it's like, oh, if Adobe starts doing this, then this is really wrong.

Leo: Companies do tend to like to do stuff like that, just kind of auto-install software.

Steve: I know.

Leo: Lately I've just made sure, you know, like very carefully when I'm installing software, especially free software, watching each window, say oh, no, I don't want the Yahoo! toolbar. No, I don't want the - uncheck, uncheck, uncheck.

Steve: I know. And by default they're checked, and you've got to go in and manually say, no, thank you, I don't need another copy of Google Toolbar installed.

Leo: Golly.

Steve: Anyway, we did have a nice note from Martin. He said, "Hello, Steve and Leo." So he addressed both of us. He says, "Not a great SpinRite story, but just a successful one." He said, "My primary data print server started acting funny. So I thought a reboot was in order. Once the server turned back on, it sat and sat at the 'applying computer settings' screen. Uh-oh. A need for SpinRite. So I ran SpinRite at Level 2. It found two bad, unrecoverable sectors in the course of 30 hours. Following the completion of the scan, a successful reboot was performed, and the server works perfectly now. I've since reconfigured a new server for my data and printing. But without SpinRite, I would have had a really tough time pulling my data off the old machine. Like I said, just a plain SpinRite success story, a successful one. Thank you for a great product."

Leo: That's not so plain. That's nice.

Steve: Yeah.

Leo: It's a good feeling. Let's get John on right now because he's been, poor guy, he's been waiting in the wings for half an hour now, and I want to call him. I think he's in Great Britain, which means it's also getting later and later at night.

Steve: Yup.

Leo: So let's get him on. He's the author of "The Geek Atlas," which I love. In fact, I will go get my copy of it to hold up as we talk to John. It's really a must-see. And he has a good website, too, which you can go to. It's, let's see, JGC.org. I'm just calling him up right now.

JOHN GRAHAM-CUMMING: All right.

Leo: There he is. Hey, John.

JOHN: Hello.

Leo: Welcome to the show, John.

JOHN: Now, do you see me all right?

Leo: I don't see you yet. But if you turn on your camera...

JOHN: Maybe if I press this video button...

Leo: It's a magic button there. There he is.

Steve: And Leo, this is an authentic UK accent.

Leo: As opposed to my crappy fake one? Yes, it is. John, it's great to see you again. Welcome to the show.

JOHN: You actually sound like an evil villain in an American movie, trying to be British, when you do it.

Leo: Yeah, yeah. Not good, I know.

JOHN: Although somebody in the chatroom accuses me of sounding like an evil villain. And I do actually have a white Persian cat.

Leo: Uh-oh.

JOHN: Which I could bring onto...

Steve: Uh-on. Long as it's not on your lap, stroking it.

JOHN: Not right this minute.

Leo: Ernst Stavro Graham-Cumming. So, Dr. Evil, welcome to the show. This all started with a blog post. And we put a little SnipURL, or Steve has, together, snipurl.com/javascriptsecurity, if people want to read it.

Steve: Yeah, John sent me a little heads-up to a blog posting of his, probably a subject that he knew already was near and dear to my heart, on his blog posting, where it's titled "JavaScript Must Die." He said, "My thesis is that the security situation with JavaScript is so poor that the only solution is to kill it. End-users have very little in the way of protection against malicious JavaScript. Major websites suffer from cross-site scripting and cross-site request forgery flaws," both of which we've covered in this podcast in the past. "The language itself allows appalling security holes. And as data moves to the cloud, the 14-year-old JavaScript security sandbox becomes more and more irrelevant."

So, you know, I've spoken over and over and over about just the idea, the concept that a user browsing the Internet can go to a server, and that server can give their browser code, any code, of any kind, I don't care about the details. The idea that you could go to a site you've never been before, and something remotely, who you don't know that you trust, can install code on the fly which your machine will run, in any fashion, creeps me out. I mean, from a security standpoint there's just nothing good about that. What I love about what John has done is he understands and knows JavaScript inside and out and can put meat on this fundamental concern I have, I mean, the idea that that's basically a bad idea. And John can fill in the details.

JOHN: Okay. What an introduction. So perhaps I should tell you just a little bit about this talk that I gave and the conference it was at because it sets the scene a little bit for what I was talking about, which is that this was given at the Virus Bulletin Conference in Geneva in September. And Virus Bulletin is typically about viruses, malware, worms, all the sorts of things that come out of the virus industry. It's a very hardcore conference. There is a commercial track in the conference where you get some sort of commercial discussions, but it's really about technology.

And I had wanted to talk, I've talked at that conference quite a lot of times. And I wanted to talk about something a bit different this year if they would let me, which was JavaScript security, because I thought that what was needed was a bit of a wakeup for people. And this was a welcoming forum where we could talk about it. And so in the presentation I pushed as hard as I could in, you know, claiming that JavaScript had to be completely destroyed. And in fact I even, right at the end of the presentation quoted from, I think it's "Aliens," where she says we should take to the air and nuke the whole thing, to try and sort of get the point over that it is a very serious situation. And what I did in the presentation was I went through some very serious examples of problems with JavaScript. Now...

Leo: That's something you don't have to do here because that's pretty much what this show is all about.

JOHN: Yeah. And of course as a listener I'm well aware that...

Leo: It's about all we talk about.

JOHN: You just have to say "JavaScript" to Steve, and he gets a sort of scared look on his face, as well. But...

Leo: Understandably. Me, too, now, I have to say.

JOHN: Well, you know, and to get things off to sort of a good start, the way in which I deal with it, just so everyone knows, is I use NoScript...

Leo: Yes.

JOHN: ...in Firefox.

Leo: As does Steve.

JOHN: So I whitelist the sites I really trust. And then when I come to something I don't trust, obviously it's off. And then if I need it, I'll do the "temporarily turn it on," just so I can take a look around at what's needed on that particular site.

Leo: Just to provide complete balance, I should say I don't do that. I use JavaScript on every platform, all the time. I just boldly go where no elephant has gone before. And you probably think I'm crazy.

JOHN: No, I don't think you're crazy.

Leo: Oh.

Steve: So give us a sense, John. There are a number of things that you've touched on. One was I loved this notion that you picked up on that the security model, which is now 14 years old, as is JavaScript, is no longer really protecting what we care about.

JOHN: Right. So if you go back in history, JavaScript security, the sort of defining areas of JavaScript security date to 1995 with early Netscape versions. And at that time there were two big concerns. There was a worry that a malicious website might attack your computer. You know, the big worry was, okay, there's this JavaScript thing running in your browser. Somehow it's going to get access to your documents folder and steal the letter you wrote to the bank or something like that. That was one big concern. And that's why JavaScript doesn't have an easy way of getting at files on the disk. And in fact that's why when we do uploads of files we have to go on this browse thing and select the file and actually get it because JavaScript specifically prevents the code from actually going and looking at files on the hard disk. So it was that sort of "protect the computer" side of things.

And then the other key thing was stop a malicious website interacting with another one. So you can't be on your bank's website, and then some other website suddenly is able to play with the bank's website and do a transfer or something. That one's still very, very important. That's a very important part. That's the cross-domain security which we can talk about.

But the first one, about stop a malicious website from attacking your computer, I think is less and less relevant. And the reason for that is essentially we're moving a lot of our stuff out into the cloud. Just look at, you know, Google Mail, Google Docs. In my company, in my day job, we use Google apps for everything. So we've got the calendar, the mail, documents, it's all up in the cloud. So that notion of attacking the PC is becoming a little bit irrelevant. You know, my work computer, I've got barely anything that's actually on it. You could steal it from me and, you know, you're welcome to it. Don't actually do that, but what I mean is, you know, it's not full of those documents. So that part of the JavaScript sandbox is important because you don't want random code attacking your machine. But it's not as important as the cross-domain attacks that can exist.

Steve: Well, I know that in your slide presentation you also talk about the fact that JavaScript is inherently a global language. And you give an example of the TechCrunch website, which loads 18 - it looked like the home page - 18 different third-party JavaScripts from, for example, Mediaplex.com, ScorecardResearch.com, Quantserve.com, IXNP.com, DoubleClick.net, GoogleSyndication.com, CrunchBoard.com, Snap.com, TweetMeme.com and Google Analytics. So there is script coming from all of those different domains onto the same page. What's the consequence of that?

JOHN: Okay. So, first of all, I chose TechCrunch just because they load a lot of JavaScript. But they are by no means an outlier. There's plenty of other websites that load lots of different sorts of JavaScripts. And I actually have stats on that which I'm going to blog about in the next few days because I've got a spider that's been looking at this. And there's an important thing to distinguish when we talk about this, when I talk about the consequence, which is, yes, JavaScript has very many global variables, global functions, global objects, which I can talk about separately.

But specifically in the instance of TechCrunch, what's dangerous is something that's slightly outside of the language itself, but the way in which browsers use the language. So that is that when a web page like TechCrunch is loaded, and it contains a whole load of these script tags, and script tags have a thing with the source attribute, and the source attribute says go get this piece of JavaScript from over there. So it could be from DoubleClick, it could be from Google Analytics, could be from all over the place.

What happens then is that the browser puts them all together, if you like, in the web page you're running, so in that TechCrunch web page. And they're able to interact with each other as if they all came from the same place. So there's no sense that they are in any way separated. So they can all talk to each other. They can all call each other's functions. They can all look at each other's variables. So they have this equal access. And what I liken this to is the way in which on Windows you run as administrator all the time.

It's a bit like the administrator thing. You know, you're on the TechCrunch website, everybody gets to do everything to everybody else. And that means that, if you were able to compromise one of those JavaScripts, any one of them, maybe by breaking into the website that hosts it, by changing a DNS setting, any way in which you could compromise it, then you get access to everything else.

So just to give an idea, Google Analytics is present on around 40 percent of the top websites by traffic. Imagine if you could insert into Google Analytics's JavaScript. You'd instantly be running your code on 40 percent of those websites when people went to get it. Because there's no way of protecting it. There's no way of knowing that it's genuine. And there's no way that JavaScript is protected from the other bits of code in the page, or the page itself. It gets to do whatever it wants.

So what scares me is, I mean, I went ahead and said we've got to kill JavaScript. But what scares me actually more than that is the way in which the browser uses JavaScript, which says there's no containment between these things, and there's no way of proving that the JavaScript is correct. For example, we've seen - I think, Steve, you've talked about this, these DNS attacks where you're able to modify DNS. Imagine if you did that to an ISP, where you just decided to change the entry of Google Analytics. Suddenly you'd be able to attack an enormous number of websites that people were loading through that ISP. And there's no way that you can actually verify that that happened.

Steve: Right.

JOHN: The big problem is, I thought, well, wait a minute. Maybe if we loaded those scripts over HTTPS, we'd get an invalid certificate, and the browser would warn us and say, wait a minute, this has been modified. But it turns out, I did tests on this, that the most popular browser out there, IE, gives you a warning of the classic sort of, you know, there's something funny here, do you want me to carry on, okay. And then goes ahead, if you hit okay, goes ahead and runs that particular piece of JavaScript. And of course we know that most users are educated to hit okay.

Steve: Right.

JOHN: So actually doing this HTTPS protection doesn't help you, unless you're running Firefox or Safari. And what happens in that case is, if the certificate fails because you're loading the JavaScript through HTTPS, and the certificate's bad, it just simply doesn't load it. Silently throws it away. So, you know, once again Firefox, and in this case Safari, do the right thing. They protect you. So if you're going to do anything to sort of save this, you could do HTTPS to protect it. But the big problem, when you go back to this global thing, is that the browser shoves all of the script tags, essentially, at the same, what I call the "administrator level" in the browser. And that's what causes your major problem. And that's what I find the most scary thing. There are issues...

Steve: Well, and it's also...

JOHN: Yeah, go ahead.

Steve: As I understand, it's also possible for a JavaScript to redefine the language intrinsics; to, like, redefine fundamental intrinsic functions in the language.

JOHN: Yeah. So that's a rather - there's a really great example of this which happened with Twitter, which is that the - there was a way of getting a list of, I believe it was the people you were following, through using a thing called JSON. Now, JSON - JavaScript

Object Notation - is, essentially, it's an object which is written in JavaScript. And so it could be an array, or it could be an associative array, but written in JavaScript. And what happens is the browser goes and downloads it, normally, so that some other piece of JavaScript can use it to display something on the screen, like the list of people I'm following. If you're logged into, say, Twitter or any other site, then of course you have this problem, which is that the cookies can be sent by anybody. So if anybody says, requests something, it looks like you're logged in. And so you have that problem which happens with, you know, cross-site request forgery where you're logged into your bank; and, you know, it tries to do a transfer even though you don't see that happening.

Well, so the really nasty example with this Twitter thing was that what it was doing was there was this JSON object. Now, the JSON object isn't JavaScript code itself that you could actually look at. So nobody should have been able to get it if they were doing one of these cross-domain things. But when it got loaded by our script tag, because it was actually JavaScript, it got loaded into the same, if you like, context as any other JavaScript that was loaded. Now it had no name, so you couldn't in the language go and poke at it and say give me the list. So it looked like it was safe to do this. But it turns out in JavaScript, because of its incredible flexibility they built in, it's possible to actually redefine the object constructor.

Now, if we go to the way in which objects work, when you have, in an object-oriented system, you have something which says I'm making a new object, and at that point sets up memory and things like that. Well, it turns out that JavaScript has this special thing, there's nothing wrong with it, it's just a special JavaScript thing called a "prototype." And you can go into this prototype thing and actually redefine things which to many people they would think are inherently not changeable. And one of the things you could redefine was what we call the "setter," which is the thing that actually sets the values that go into this object. And what it was possible to do, if you redefined the setter for the global object, then when this Twitter status thing got loaded, even though it had no name and was essentially anonymous, it had to get constructed and set. And in that moment you could grab its contents. And so that's an example of something in the language that's kind of scary. Again...

Steve: And so it...

JOHN: Yes.

Steve: So if this was actually done and exploited, which I guess is what you're saying happened...

JOHN: Yes.

Steve: ...it's clear that this is not a fault of the language. It's the fact that the language is very powerful. But it also means that somebody somewhere who really understood this stuff went out of their way to create this exploit. I mean, wrote code that would leverage this functionality of JavaScript in a way that ended up being malicious and allowed all this information to be stolen from people.

JOHN: Yeah, absolutely. I mean, this was done by some security researchers. And I'd have to find the exact reference to it. But it's pretty easy to define, though, because I know that Bruce Schneier talked about it. It's called "JSON hijacking." And JSON is spelled J-S-O-N.

Steve: Right.

JOHN: And it's an example of something very, very powerful in the language which is rather scary because it lets you get at things that are very, you know, it's almost like you're dropping down into an assembly language level and fiddling with stuff on the processor. This is at such a high, deep level in the language that it's scary. It also allows you to do amazing things, which has made JavaScript very, very powerful.

But I want to emphasize one thing. You could leave this in the language if you fixed the problem of all the scripts in a page having exactly access to each other. Because if you separated them into little, you know, silos, then the script that was actually redefining the object constructor could use that for legitimate purposes, and it wouldn't ever be able to touch this other script that was coming from Twitter. So, you know, in the presentation I said, well, you've got to destroy all of JavaScript. The biggest issue for me is this issue of all the scripts being able to run at the same level of priority. And actually that's number one. And then number two is there's no way of proving that a script hasn't been modified. So that's another worrying thing.

Leo: Let me ask you about that first point. Could a browser with attention to security like, say, Chrome do that kind of siloization of the JavaScript modules?

JOHN: Yeah, actually there are, there are a couple of proposals around. Mozilla has this thing called the Content Security Proposal, CSP, which proposes a way in which basically we restrict the way in which JavaScript is used in the browser. Douglas Crockford, who's one of the real experts on JavaScript - I'm definitely not at his level - has proposed a system of sort of silos for the different scripts.

Steve: Would you call them separate name spaces?

JOHN: Well, I think the idea is they are sort of separate name spaces; although, you know, in JavaScript, because things are so sort of global, it's a bit dangerous to talk about name spaces. But the idea would be that scripts could get the sort of tag associated with them that says, you know, we're the same. We come from the same site. We can work together. And then, you know, someone building a website, if they didn't do that, then it'll be like, well, you're siloed off there. You go do your thing, in track with the web page, and that's fine. Because you want to be able to do this stuff for things like Google Analytics. People want to be able to do that, you know, bring in external scripts and run them. But the danger is they can interact with each other.

So, you know, there are proposals. I made a different proposal which is around signing scripts - and we can talk about that a little bit further on - which is to do with cross-site scripting, which is another big area. But this idea that all the scripts are running in the same context is scary and I think does need to get addressed. And I think there are some thinking - there is thinking around addressing it.

Steve: And do scripts not stomp on each other then by mistake? Like, I mean...

JOHN: They can do.

Steve: Yeah.

JOHN: Yeah, they can do. Because you could, for example, redefine a function, you know, you give some function. And so often when you look at JavaScript that's actually out there, they'll use really weird names, you know, start things with underscore and just go xxxxx, you know, just because they don't want to stamp on each other.

Steve: Right.

JOHN: The other thing is, there is a way in JavaScript to completely contain your functions, your variables, which is to use a closure. A closure is a special sort of function, basically sort of anonymous sort of function where it turns out that you can define variables to be local in JavaScript if you specifically do define them that way. And if you wrap it all up in the global function, you know, it's this anonymous closure thing, and nobody else can actually get at it. So I think it's obviously very, very powerful. It has all these sort of facilities. You just have to be pretty good at programming it to be really safe.

Steve: And is there any evidence at this point that this single-context environment is currently being used by some scripts to talk across to other scripts so that creating some separation would then break sites that had become dependent upon it?

JOHN: Yeah, it probably would. There are some examples in the web analytics world, particularly when you get in a situation where you upgrade to a new version where they make use of the fact they can read what the old version was up to and grab data out of it. So, you know, you've got the old tag and the new tag. And the new tag can say, oh, by the way, I can go grab that from this tag over there, because it knows it can get access to it. So, yeah, there is an upgrade issue for that. And it's certainly the case that inline scripts, i.e., ones that haven't been sourced from somewhere else, if you go and look at, you know, any large website, they are expecting to be in the same context. So you've got to, you know, you've got to be careful about how you do this. But I think it is, from a security perspective, something that needs to be worried about. Because it is, that combination is scary.

Steve: What's your sense of how worried people are? I mean, are we the only people worrying about it? Or are, like, important people worrying about it?

JOHN: I think...

Leo: You guys are important, now, come on.

JOHN: No, I know, self-important.

Steve: People who could actually do something.

Leo: Ah. That's different.

Steve: Yeah.

JOHN: Yeah, people actually do something. Well, there is this Mozilla CSP thing, which I think is very important. Douglas Crockford, of course, is very involved with that kind of script, has been making a lot of noise about this issue, and he has some good presentations about it. And there are a couple of moves to try and make JavaScript safe. People may remember quite recently there was a situation with The New York Times where a malicious ad was inserted in The New York Times. This is exactly an example of, yeah, here's a piece of JavaScript, it could do what the hell it wanted.

There is a thing called ADsafe. And ADsafe is a way of statically examining a piece of JavaScript which you're going to use in an ad, typically, and enforcing certain security so it can't do lots of malicious things. And in fact what it does is it uses this thing I talked about where you encapsulate the entire thing so it can't get out, around. And it's only allowed access to a thing called the ADsafe Object, which lets it, well, this is a proxy for some of the more dangerous activities. So that is a very good thing because it allows you to take an ad and test it.

And there's another thing called Caja, or Caha, C-a-j-a, which is again a safe subscript, subset of JavaScript. So this is definitely being thought about. But I think at the current stage of things it's a bit scary that there are all these different issues going on. We've seen all these cross-site scripting and many other things I talked about in my presentation, which are rather terrifying.

Steve: What scripting does an ad need to run? I don't want ads running scripts.

JOHN: No, I realize you don't. You probably just want there to be...

Steve: I mean, it's enough to put up with the ad itself.

JOHN: You know, to be honest with you, I actually would feel happier if we got JavaScript under control and dealt with some of these issues, and that they were used for ads rather than Flash. Because at least the JavaScript implementations are, for the most part, open, and we know what the hell is going on inside them.

Leo: A lot of times a JavaScript - it's the worst of both worlds because JavaScript is used to enable Flash.

Steve: To invoke the Flash, right.

Leo: We use that on our website. We have a Flash player, and the JavaScript checks to see if you have that capability and tries to do something intelligent if you don't.

Steve: Well, in my case, I was briefly using Google Analytics until I looked closely at what it was doing, and I realized that every time one of GRC's pages was being presented to anyone who visited GRC, the code which I had been asked by Google to tack onto the end of my page was going out and fetching a block of code on the fly that I had no chance to look at or understand. But it was - so basically my server was causing anything Google wanted to append to all of my pages to be included. And I just - I thought, well, I just can't have this. I mean, that's ridiculous.

JOHN: Well, and if you look at that TechCrunch example I gave, I mean, there's tens of them. And TechCrunch is basically in a trust situation. They say, well, we trust this code isn't doing anything malicious, and it probably isn't. And we also trust that nothing's ever going to go wrong with it. And you've just got no way of verifying it.

[Talking simultaneously]

JOHN: ...a proposal to do signing of JavaScript. This was back in Netscape 3.0, maybe, I mean, a long time ago. You can still find information about it. And it's just gone nowhere. And I would be much happier if they were signed so we'd know when things changed.

Steve: Yeah, one of the things that all security experts know is, as bad as external security problems are, the great majority of actual exploitation or mistakes or, I mean, either by mistake or deliberate, occur internally. So all it would take would be somebody with some malicious intent at any one of these sources, for example, in the TechCrunch example, not to pick on them specifically because, as you said, many sites are pulling many scripts from many locations. But, I mean, it creates this multiplication of potential for problem. It would just take one insider job exploiter to change the script in some subtle way that would have a huge effect on the Internet, not just one site, but every site that pulled script from them.

JOHN: Right, exactly. Exactly. So that is a real worry, yes, definitely, that, you know - and I think in a way it can be quite easily dealt with by siloing. So if something bad happens, you know, it's contained, at least. Now, that would make an enormous difference. I think if we could then sign scripts, then that would make a big difference because then you, you know, you'd enter into some agreement and say, hey, you know, these guys signed it, and so that gives you another level of, you know, assurance that it's the right thing, the thing you were actually asking for.

Steve: And are you suggesting that it be digitally signed so that it would have a certificate that would be checked before it was run? Or that then it would be agreed to, a certain script, and then they would not be able to change it without going through some sort of authentication process or authorization process.

JOHN: Well, I suspect that last one is a bit too complicated. I mean, I think if you got to the point of just saying, look, Google Analytics always comes over HTTPS. It's signed by Google. And, you know, we've signed this piece of JavaScript. That would give you quite a lot of information. It would save you from a lot of potential problems to be able to do that. And the only thing I'd be...

Steve: Yup, and then that - go ahead.

JOHN: No, the other thing I'd be proposing is that the other problem is that cross-site scripting is a problem because, if you can - so cross-site scripting, which is a really weird name, basically it's the problem of somebody manages to inject JavaScript into a web page via, say, in a chatroom, for some reason, there was an interesting example on reddit not very long ago where someone manipulated a markdown, you know, the markdown format they used for marking things up to actually inject a piece of JavaScript into the page. That got stored in their database. Anybody who read the comments on that particular story, that JavaScript executed in their browser.

Now, if you think back to the fact that all the JavaScript is executing at the same level of priority, that piece of JavaScript executing in there now has got access to all the other JavaScript in the entire page. Now, you can actually fix that problem. And this is partly what Mozilla CSP is trying to do. They do it in a slightly different way. But one way to do it would be to sign the script tags you put into a page. So this page author would say, I put this in, and here's my signature. You can check. Now, I mean, it wouldn't matter if someone managed to insert a piece of this JavaScript because it wouldn't have the signature.

Steve: Right.

JOHN: And you would - that would end cross-site scripting if you did it. So, again, that's not inherently bad in JavaScript. It could be a different language; right? It could be VBScript. But the issue is you don't know where it came from, and it gets nevertheless sort of administrator, if you like, within the contents of the page, access.

Steve: Right. So it really, in the case of cross-site scripting, it really is a strange fluke of sort of the Web 3.0 approach where users are submitting content, that malicious users can submit script content which will be displayed and run by anyone who then views that content.

JOHN: Yeah. And the thing is, there's loads of, you know, examples of these cross-site scripting things. And the inherent problem is that what happens is you've got these layers of software in, you know, a common website. So suppose that, you know, you're using JavaScript and HTML in the browser. It gets submitted back over HTTP. It goes into a Ruby on Rails app which gets stuffed into a MySQL database. There's loads of potential for weird stuff to happen to that stuff you entered along that route because you've got all these different languages and escaping, well, in MySQL it's like this, and in Ruby it's like that, and et cetera.

And there've been, you know, classic examples of this. The reddit one was interesting because it was a bug in their markdown implementation. There was a Ruby on Rails one to do with Unicode decoding, where they had this thing which tried to deal with Unicode, and they'd handwritten it themselves, and it turned out there were some bugs in it, and it was possible to create essentially bad Unicode that got decoded into ASCII with a script tag in it. So that, you know. And there's another great example which is to do with UTF-7, you know, Unicode type thing, which is if your website wasn't specifying that it was in, you know, UTF-8, UTF-16 or whatever, you could use UTF-7 characters and then stick in their metatags - and by the way, this is UTF-7, which would then promptly get decoded by the browser into a script tag, and off you go.

So there's loads of potential. I think Douglas Crockford calls this the turducken problem. You know that thing where they put a chicken inside a duck inside a turkey? You've got just layers and layers of different stuff. So, sorry, I'm getting too excited. Now I'm going to have to start coughing.

Leo: Stop talking about turduckens.

Steve: Well, just in terms of traditional security, how secure has JavaScript's actual sandbox turned out to be in practice?

JOHN: Actually, to be honest with you, I think it's pretty good. There have been lots of bugs. But, you know, hey, what a surprise. It's certainly no Windows, let's put it that way. I mean, it's, you know, there are bugs, and you do see examples, if you look in the SANS database and things like that, of such and such a bug in the sandbox where we could get through and do cross-domain things or cross-security-domain things in IE. But, yeah, there are bugs. What a surprise. There was a nasty one in Google Chrome which the Mozilla guys found in, funnily enough, by actually inspecting their code. That was just a few weeks ago.

Steve: Right.

JOHN: But to be honest, the sandbox has been pretty good. The bigger problem is not the sandbox because, as I say, it's not breaking out of the sandbox is the problem. You can do plenty of damage within the sandbox.

Steve: Right. And so problems there are resulting, or are the result of, bugs in JavaScript's implementation of the sandbox. But everything else we've been talking about are sort of - are fundamental to what happens when you have scripting power in a browser in this very complex environment where users can provide code, code is being pulled from multiple domains into a single page, code from all these multiple domains is able to see each other and interact with each other. And again, all of this is hugely complex, and we know that complexity is the enemy of security. So bad guys are able to look at all this and just rub their hands together and think, wow, look at all this opportunity we have for exploiting all of this complexity, fundamentally enabled by the fact that we've got a scripting engine running in the user's machine that will do what we tell it to.

JOHN: Yeah. And by the way, I've got nothing against us having a scripting engine in the machine. I think that's actually a very important part of where we're going with, you know, mobile code being able to be downloaded. And, you know, I'm very happy to use things like Gmail. I think it's fantastic.

Leo: Yeah, absolutely. You use Google Docs in your company. You can't do it without scripting.

JOHN: No. And it's fantastic. And I think the issue is we need to look at the way in which JavaScript is being used in browsers today and deal with some of these problems because, you know, there are lots of nasty examples of cross-site scripting problems. You know, stealing someone's Twitter friends is probably, you know, the timeline is not that serious. But you can imagine these things do get more and more serious as the systems get more and more complex. And so fixing is important. So that was sort of the reason for my presentation was to say, wake up everybody. This is not, you know, a fun situation. Let's not just let it run and run like this.

Leo: Steve, I don't want to preempt, if you have some other questions before we get to John's recommendations.

Steve: No, I think this is perfect. I'd love, I mean, I know my solution.

Leo: Well, it sounds like John uses the same solution. You both prefer to use NoScript. Is that right?

Steve: Right. Right.

Leo: John, how do you use NoScript? You said you turn on full protection and then allow sites.

JOHN: Yeah. Yeah. It's funny actually because I didn't use it for a long time. And I used to listen to Steve talking about JavaScript security and go, what a nutcase he is. Of course there's nothing wrong.

Leo: Nothing wrong with that.

JOHN: What a crazy idiot, going on about how you shouldn't run JavaScript in your browser. And then I started - I had done a few things with JavaScript. Then I started working with some really high-end JavaScript developers, and we started looking at some things. And I just got more and more and more appalled. So, yes. I use NoScript, and I basically have it, you know, it's like those old application firewalls. I have it yelling at me all the time. And I find it infuriating. But I like it...

Leo: That's why I don't use it.

JOHN: Yeah, no, I understand. I mean, so...

Leo: You need it, though, it sounds like.

JOHN: Of course the sites I'm using all the time are whitelisted, you know, they're allowed to do whatever they want. So I just go ahead and decide to trust them. And, you know, for other sites I have it essentially switched off, you know, so that basically, you know, you cannot get any JavaScript running unless I specifically say so. And then I'll go in and do it when I need to.

Leo: What about on cell phones? I mean, a lot of cell phones use JavaScript.

JOHN: Yeah, I mean, I have an iPhone. And to be honest with you, I use it to access not very many different sites, so...

Leo: Stay off the web.

JOHN: No, I use it to stay on the web, of course, particularly for email and Twitter. But I don't find myself browsing around a lot on my iPhone.

Leo: For that reason.

Steve: Well, the idea that in the long term JavaScript is with us, I think no one contests.

Leo: Yes.

Steve: We agree that a scriptable browser is vastly more useful than a dead browser that just lays there and can only display static pages that come from servers. I mean, that was, you know, the Web 0.5 version of the world, and not something we're ever going to go back to. The idea that there are, as John has presented, there are definitive ways to fix some of the aspects of JavaScript, I mean, I'm still uncomfortable by the idea that anything I, you know, random browsing is going to be running code on my machine. That just isn't cool with me.

But the idea that these more appalling security holes, I mean, the things that are in there really by design are being looked at and can be tightened up I think is tremendous good news. And I will tell everybody that, I mean, using NoScript is a burden. You go to a site and something doesn't seem right. It's not quite working. And so then it's like, okay, let's see if turning scripting on will make the form work correctly. And it always does.

JOHN: Yeah.

Steve: I don't have NoScript set to tell me when it's blocking something. That's really obnoxious because scripting is everywhere now. So all I do is, if I see that a site doesn't seem to be functioning right, I look down at the bottom, and there'll be like a little red slash through the "S," saying I'm blocking stuff for you. And it's like, okay, fine. And so normally I temporarily allow scripting. You're able to do it just like for the session or to permanently whitelist, which is very nice. That way I don't have my whitelist growing forever. Because most sites that I'm randomly going to, I'm not coming back to. So I find that it really works for me. And I think for today it's the right balance between, you know, we have to have some scripting for sites that need it; yet you don't really just have to be running around completely naked on the 'Net all the time and allowing anyone to poke at your browser.

JOHN: The problem is with NoScript is that, you know, you have to be a pretty high-end user to be able to use it. And that's why, you know, my presentation, you mentioned this at the beginning, Steve, which is I said there's no viable way for my mother to control this. And that's why this is something that's got to be fixed technically to protect people from what's happening. It's good for people like us. But it's not a solution for the general web user at all. It drives me insane. And, you know.

Leo: Well, that's why I don't use it. But now I'm terrified.

JOHN: There's one thing I didn't talk about, which is kind of interesting, which is just to leave you with this thought, which is that there's no way for a website to tell the difference between a click made in JavaScript and a click made by a human.

Steve: Ah, I meant to bring that up. Yes, yes, yes, I meant to bring that up. Yeah, talk about that, John.

Leo: This is that click fraud thing; right?

JOHN: Yes. The thing is...

Steve: We actually talked about it already earlier in this episode, the idea that - well, go ahead - that JavaScript and user clicks are seen as the same thing.

Leo: Right.

JOHN: Yeah, there's just no way. Because, I mean, what happens is you've got this system in the browser of events, which are things like, you know, you move the mouse over something, and [indiscernible] changes color or whatever JavaScript is used for. Or you click on something. And because of the way in which the web is put together with these essentially different layers which are essentially independent from each other, what happens in JavaScript is you can fire an event. You can say cause this click to happen, which turns out to be an immensely useful thing to do because you might want to, within your JavaScript, you know, click on something as if the user had done it. You know, you might have two buttons, and you want to actually click the other one, or some other thing that has to happen. But actually inside, and I guess going back to the website, there's definitely no way to tell whether a machine did that or a human did it. So you get this problem of you can't tell whether a person actually initiated that action or not. And so that's another one of those things where you think about it and say, wow, it would be really good if you could actually tell if that was actually the mouse was moved and someone actually clicked on it, or if a piece of code said, hey, click it.

Steve: Well, yes. And for all kinds of, I mean, we talk about authentication a lot. And so it's really important to be able to, like for a server remotely to be able to know for sure that when it challenged the user, the user himself physically moved the cursor over a button and clicked on it, rather than the page happened to load some script from somewhere that did that on the user's behalf, for something really important where you really want to assure you actually have user focus and awareness and interaction. And with the model as it is now, you don't have that.

JOHN: Yeah. Yeah. So that's, you know, one of those things in, you can argue whether that's in the language or in the implementation of the language in the browser, but is a worrying thing if you're trying to understand, you know, what did a machine do and what did a person do.

Leo: Well, I hope that people are paying attention to this, and I hope that something gets done about it. We need it. It's a very powerful language.

Steve: The important people. We hope the important people are paying attention.

Leo: Well, all it takes is for Google, for instance, although given the problems you talk about with Google Analytics, maybe there's no hope. But for Google or somebody, maybe Google could say, well, we're going to make Chrome enforce this. We're going to silo the data.

JOHN: Well, Google are the people behind the Caja - or Caha, depending on how you say that "j" - project. So there's definitely thinking in Google going on there about it. And I talk about Google Analytics just because of its incredible popularity. There's nothing inherently wrong with Google Analytics. I mean, it scares me that it's on so many sites. That's what worries me about it. I don't think Google's up to anything naughty.

Leo: Well, but loading code without any, I mean, kind of - it's inherently risky. But it's doing what everybody does with JavaScript, I guess, so...

JOHN: Yeah, it sure is. And it's just that's the biggest worry is that it lets us slap things together, and they're all running as the same priority, and that's a scary situation.

Leo: Is Caja as good a solution, do you think, as siloing or signing?

JOHN: No. I think if we had proper silo, you know, we could actually break things up into separate spaces and then...

Leo: We don't need to strip the language down, then.

JOHN: I think there's some merit in working, going down to a subset of the language which is known to be safe and sort of building back up again from there to try and avoid some of the scarier parts. I think that's a valid thing to do. But I think that, you know, just being able to sign things and know they are what you're expecting to get is very important.

Steve: Well, and we're talking about a language which is 14 years old. Remember the world 14 years ago when JavaScript was designed. It wasn't facing anything, any of the challenges that it's facing today.

Leo: Well, part of the problem was that JavaScript wasn't in fact designed. I mean, it was very ad hoc and was implemented in a variety of different ways.

Steve: Well, and the fact that it's even called JavaScript is a misnomer.

Leo: Yes.

Steve: I mean, it's got nothing to do with Java at all.

JOHN: No, no, absolutely. I mean, I guess Netscape were hoping that Sun would buy them, and so they called it JavaScript.

Leo: Well, John, I'm so glad we could get you back on again. I do want to give a plug for your "Geek Atlas" because it's a great book from O'Reilly that covers 200, what 256 of the great geek...

JOHN: 128.

Leo: 128, okay. Two to the...

JOHN: Yeah, here it is.

Leo: Here it is. Take a look at it. He's got it.

Steve: 2^7.

Leo: 2^7, not 2^8, yeah.

JOHN: If I can get it back in front of the camera. Here we go. And now, here's the big surprise. I just received this today.

Leo: Oh, but the book is upside down now. What, it's an updated...

JOHN: It's a German edition.

Leo: [Speaking German]

JOHN: Yeah, so he can speak German. So English or German, depending on what you like.

Leo: Congratulations. That's really great. It's a wonderful book. I have it, and I hope maybe someday to retire and make the trek to all of those sites. Be fun. Be really fun. JGC.org is the place to go for the blog, John Graham-Cumming's blog. And Steve's SnipURL for the article that started this all and the slide stack that started this all is snipurl.com/javascriptsecurity. Did I get that right, Steve?

Steve: Yup.

Leo: Well, that's great. Thank you, John. We really appreciate it.

Steve: Thank you so much, John. It's great spending the time with you.

Leo: I've got to quickly go instead NoScript on all my machines.

JOHN: It's too late. I already infected them.

Leo: How many times have I said that before? But this time I'm going to really do it. You did finally scare me into it. And we'll see you next time on Security Now!.

JOHN: Right.

Leo: Bye bye.

Steve: Bye bye.