Fuzzy Browsers

Description: After catching up with the week's security updates and news, Steve and Leo examine the use of "code fuzzing" to locate functional defects in the web browsers we use every day. Surprisingly, every browser in use today can be crashed with this technique.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-285.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-285-lq.mp3

Leo Laporte: This is Security Now! with Steve Gibson, Episode 285, recorded January 26, 2011: Fuzzy Browsers.

It's time for Security Now!, the show that covers your security, your privacy online. And who better to do that than the man at - I was going to say SGgrc. That's his Twitter handle. It's GRC.com. It's good to talk to you, Steve Gibson.

Steve Gibson: Leo, great to be with you again, as always. For Episode 285 we're going to talk about fuzzy browsers.

Leo: Are the browsers fuzzy? I mean, are they actually fuzzy?

Steve: Well, actually they're fuzzier than we wish they were. We wish they were really solid and sort of in one piece, and not something you would characterize as being fuzzy or furry or flaky. Unfortunately, they are, it turns out. And we're going to talk about the details of browser fuzzing, which is the practice of writing scripts for browsers specifically for the purpose of stressing them and demonstrating where there are unknown problems.

Leo: Very interesting.

Steve: Turns out it's frighteningly successful, that is, this whole process. So much so that Mozilla has incorporated browser fuzzing into their standard test suite now.

Leo: Wow. Well, we'll get to that in a second. But I imagine there's a few security updates on your plate today.

Steve: We've got some updates and some news. And speaking of Google, Chromium, or the Google Chrome browser, just got itself updated, like about a week ago. And they gave their so-called "Elite" Chromium Security Award to Sergey Glazunov. And he won, actually they call it a charity, so he received a charitable contribution for this particular problem that he found. He found a handful of them. But it was $3,133.7, which of course is eleet...

Leo: Eleet in leetspeak.

Steve: And they had announced, and we've talked about it before on the podcast, that they would reserve this amount for people who found really bad problems because they wanted - or, like, really difficult to reproduce, I mean, basically for a security researcher who really earned the discovery, which he apparently did. This is a critical problem, the Google Chrome people felt, with a problem with stale pointers in the speech handling for the Chrome browser. There was this one critical problem, 13 other problems rated high, and two that were rated medium. High security problems receive $1,000, and medium ones receive $500. And actually Sergey got a handful of them. So he pretty much cleaned up. This was a good month for him. And of course all of us who are using - all of the people using Chrome get the benefit of it having fewer problems now than it did before.

Leo: Very nice.

Steve: I received, speaking of SGgrc, through Twitter was the first channel that I saw the notice, then I saw it elsewhere, really nice news. Starting today, probably not this moment for people who are listening to this live, but I would imagine anyone hearing the podcast later this evening or, like, for example, Thursday the 27th, Facebook has just announced this morning in their blog, their security blog, that they are now offering 100 percent SSL and HTTPS option.

Leo: Oh, that is awesome.

Steve: Yes. The URL for that page, I've got a short URL: on.fb.me/ihQZiv.

Leo: Oh, that's easy.

Steve: Yeah. Well, I did tweet it. I tweeted it immediately when I saw the news and gave credit to the guy who'd sent it to me. And then immediately afterwards a few other of the main tech channels picked up on it.

Leo: That's fantastic. This is in response to Firesheep, obviously.

Steve: Exactly. Now, in fairness to Facebook, they responded so quickly after Firesheep that they were already working on this, that I think this was an issue that was already on their radar. We might imagine, though, that they gave it more acceleration, that they put some more effort into it to make it happen. Quoting from this page, they said: "Starting today we'll provide you with the ability to experience Facebook entirely over HTTPS. You should consider enabling this option if you frequently use Facebook from public Internet access points found at coffee shops, airports, libraries, or schools. The option will exist as part of our advanced security features which you can find in the 'Account Security' section of the Account Settings page." I did go there after having someone else respond to my tweet, saying, eh, I don't see it. And sure enough, because I've got a Facebook account that I maintain just for exactly this purpose, I don't actually - I'm not active there, anymore, Leo, probably, than you are. But...

Leo: Oh, I wish I could say that, but I am. I got sucked back in.

Steve: Once again. Anyway, so when I last looked it wasn't there. But presumably sometime today, on January 26, 2011, this feature will be added. And I don't have to tell any of our listeners who are Facebook users, check for it over the next few hours or days. Turn it on. And the good news is, wherever you are using Facebook, Facebook itself then will work to keep you over an HTTPS connection whenever possible, which will, as we know, with threats like Firesheep lurking around, will enhance your privacy and security.

Leo: Very good news. It also I think could have a one-time password, they say.

Steve: Well, they mention that they're going to. I got excited about that and read the rest of the blog and could not find it anywhere. So I think they're talking about getting there. I mean, even the fact that they're talking about that is a good sign. It means that they're beginning to be more serious about security. So being as big as they are, and as significant as they are, that's just good news.

I picked up a little bit of news relative to the privacy of employee email. You and I have talked several times, Leo, about how people need to remember that email that you use when you're at work really is not private. You need to consider it as you're using your employer's equipment, your employer's bandwidth and connectivity. So don't consider that private. There was a suit raised when a secretary, I don't know what the exact whole story was, but something about her becoming pregnant shortly after she began her employment. Her employer got upset because presumably, shortly after being hired, she would now need maternity leave or something. But anyway, the lawyers got involved.

And what happened was that it went up to an appeals court that ruled, quoting from their ruling, the way they phrased it: "The emails sent via company computer under the circumstances of this case were akin to consulting her lawyer in her employer's conference room, in a loud voice, with the door open, so that any reasonable person would expect that their discussion of her complaints about her employer would be overheard," the court wrote. So that is what the court said relative to the issue. And the

issue was attorney-client privilege. At some point they were trying to take the position that this could not be used as evidence in the suit because this was attorney-client privilege, protected under attorney-client privilege.

And what the appeals court said, and this actually does change some law or some prior decisions, which is why it got to the appeals court, previously there had been some instances where, for whatever reason, attorney-client privilege in a situation like this was held. And in this case the California Appeals Court said, nope, exactly as they expressed it.

Leo: Cool.

Steve: So be aware. If someone has to have a secure email dialogue at work, our users know, that is, our listeners know that you could use, for example, Gmail over HTTPS, which would prevent it from being eavesdropped on, so long as your browser has not received a security certificate from your employer that would specifically allow them to filter SSL connections. And again, the way to check that is, when you have established an HTTPS connection to Gmail, then check the certificate associated with the page, typically by right-clicking on it and checking properties and looking at the certificate, and see whether the chain of certificate authorities goes directly from Google to their root authority and not through something that looks like some third-party SSL eavesdropping monitoring system. In which case you really would not be able to have any kind of communication that would not be eavesdropped by the corporate network.

We have talked about Net Neutrality a lot, and I think that'll be not a huge topic for us because it's a little bit off-topic. But I wanted to mention that Verizon is challenging that ruling that we did talk about a few weeks ago where the FCC took the position that providers could not selectively throttle broadband traffic over their networks. And it's not been clear that the FCC has the authority provided by statutory law that's in place now to do that. And so Verizon is saying the FCC doesn't, which will probably force legislation to be created that gives the FCC or somebody that kind of regulatory power, which it may or may not have, depending on how Verizon's suit comes out. So that's unfortunate.

And speaking of unfortunate, this week a U.S. congressional panel is holding a hearing to consider reviving the impossible-to-implement ISP Data Retention Bill, which we have talked about, which would, if it were enacted into law, would require ISPs to retain their customers' Internet data for two years. Which, I mean, anytime hearings are held I'm sure the ISPs must sit in front of our representatives and senators or whomever is in these meetings, these hearings, and explain to them that it's just a physical impossibility. Of course, immediately, and as part of the story, they talked about, well, we want to be able to track down child pornographers. And so they march out child pornography as the gotcha. It's like, well, you're not *for* child pornography; are you? Although in fact what we know is that, as you and I were discussing, actually before we began recording this, Leo, electronic data and electronic privacy seems to be something we're losing very quickly.

Leo: Yeah, I hate it when they conflate child pornography with all of this because it's not - it has nothing to do with it. It's just an easy out.

Steve: Right, right. It's something that everyone is going to vote to...

Leo: Yeah, who's for that?

Steve: ...prevent, exactly.

Leo: Nobody's for that. But that's not what this is about.

Steve: Yeah. And the idea that an ISP would need to record two years, essentially we're talking about two years of all the traffic through their network. That is, because of course their network traffic is the sum of their customers' traffic. And what the people who are pushing the bill want is the ability to go back for a given customer, presumably under warrant, and rummage through everything that that person has done for the two prior years. So, I mean, if you were Google, sure, you've got a database that's large enough to hold two years of all your customers' traffic. But that's - nothing that's in place like that is happening now. They're not talking about a real-time tap. They're talking about recording everything for two years. So hopefully, much as somebody wants this to happen, saner heads will prevail and say, look, we can't do that.

Leo: This is an issue we were talking about before the show began. But at some point I'd love to talk more about these pen register search warrants and how they're being, I think, misused by law enforcement, and really unconstitutional ways we're being spied upon. And it's just getting worse. And it's because it's electronic, and so it's just deemed somehow less private. It's just appalling.

Steve: Yeah, and unfortunately it does make it easier. It's not like you have to - you don't have go install transmitters in someone's home and car. You just hopefully give them a subpoena. And as you're saying, in this case now, subpoenas are no longer even necessary.

Leo: Right.

Steve: Cisco's Annual 2010 Security Report indicated that cybercrime appears to be migrating from desktops, which as we know are becoming increasingly well secured, as are the desktop users, I think, to mobile devices which are inherently much more vulnerable. So it's probably not a surprise to anyone, but I thought it was interesting that Cisco is seeing this and noting that our mobile devices, which are becoming increasingly powerful, are becoming magnets for cybercrime. Not good, and certainly not surprising.

Now, the big news, and interesting news, I held for last. And that is that both Mozilla and Google have stirred on the issue of do-not-track behavior. We talked last week about Giorgio's adding this feature, the do-not-track, the "X-Do-Not-Track" header to NoScript. It had been there actually since a little bit before the beginning of the year, and someone brought it to my attention. I checked my outgoing headers from my browser when I'm using Firefox, and sure enough, those headers were there. Just in the last couple...

Leo: Of course. You have to pay attention to it; right? It doesn't mean anything if

somebody doesn't understand it.

Steve: Exactly. So, okay, there's two different approaches. Mozilla is introducing, unfortunately, a different do-not-track header. Yeah.

Leo: You know, why not, because it's always been good when there have been many, many standards that you have to adhere to. It always makes sense.

Steve: Exactly. So theirs is called X-Tracking-Choice. So X-Tracking-Choice. And then you add to it. Then it's colon and, like, don't track me or something.

Leo: Don't track me, bro.

Steve: Oh, I'm sorry, it's Tracking-Preference: do-not-track. So Mozilla's is Tracking-Preference: do-not-track. And it's Giorgio who replied to the Mozilla blog posting. And he said - I'm sure English is not his first language. He said: "Why inventing yet another header (X-Tracking-Choice) rather than reusing the 'X-Do-Not-Track' proposal," which of course is what he adopted with NoScript, "which is already implemented in NoScript and AdBlock Plus, and also endorsed by Mozilla?" So who knows why. Many cooks are involved.

So what Mozilla is doing, and they recognize that following this is optional, that is, is not something that they're mandating, they're just saying we're going to allow a user to turn this on and send this tracking preference header out with all queries. Now, I'm all for it. I like this solution. And as I've said many times, in fact months ago I've been talking about just adding a header like this. And this has all finally come to pass. It does require voluntary compliance until legislation, which does seem to be rumbling around in the U.S. Congress at this point, legislation would then make obeying and honoring a header like this mandatory.

Now, Google has taken a different approach with their Chrome browser. And I've studied what they've done, and I think I understand it, but it's not at all clear. They've got a new extension called "Keep My Opt-Outs," which is an optional extension that can be added to Chrome. And from what I can see, "Keep My Opt-Outs" must have the effect, although they didn't say so anywhere, so I can only guess this from the name, of preserving opt-out cookies when they exist.

Now, by looking at the blog and following some links, I discovered a site I had not seen before. There is a site called aboutads.info, where you can go, which sort of appears to be an advertising industry sort of communal site that's talking about the nature of tracking and online behavioral profiling and so forth. What's interesting is that there's another page, aboutads.info/choices. You need to have scripting enabled, that is, when you go there. So when I went I saw that something wasn't working. In fact, it told me that I had to turn on scripting. So using NoScript I enabled scripting for that page. And then what happened was it enumerated a large number of known advertisers. And this site allows, sort of provides a UI that allows people to create opt-out cookies for all of the advertisers that support opt-out cookies. So if we didn't have this, you'd have to go manually to each of these different advertisers and go wiggle around through their websites and figure out how to opt out of each individual advertiser's tracking.

What's nice about aboutads.info/choices is it pulls it all together into one place. So I've experimented with it, and it works. You're able to just say "select all," or choose the ones that you sort of - that don't have a good-looking taste for their cookies and then say, okay, I want opt-out cookies. So what happens is, since only the site which owns the cookie can set the cookie, this script doesn't have the ability to do so. So it must be going out to each of these different sites, using some sort of protocol that they have agreed upon, to have each of those sites plant a new opt-out cookie in your browser, which did happen, and I confirmed that it happened.

And so that's what aboutads.info/choices does. And if you're using Firefox you can do it. You can actually use it with any browser that supports scripting. So presumably, then, Chrome's new extension, Keep My Opt-Outs, does something to prevent them from being lost or deleted. So, for example, if a user flushes all their cookies, this would take priority, and it would keep these opt-out cookies in place, even if you otherwise cleared all your cookies.

Now, the one thing I did note was that DoubleClick.net is not listed among all of these advertisers at aboutads.info. There's something like, I don't know, 60 different advertisers. And the script that ran had some problem enumerating some of them so that I was only able to end with, like, eight or so that I was able to opt-out of, and with, like, 39-some were still, they were sort of there, but not working yet. Who knows what's going on. But anyway, so aboutads.info/choices is something interesting to look at.

Now, so Chrome is not adding a header. I would rather they added a header like Firefox is going to do, the Mozilla guys are doing, like NoScript and Giorgio have already done for us. But this is something. Microsoft and IE have an entirely different approach. They've got something called Tracking Protection Lists, or TPLs. They've just recently blogged about this. This will be a feature in IE9 which is - in some senses it's better. And, unfortunately, in some other senses it's not so good.

It's better in that these tracking protection lists are XML files, which are a moderately confusing syntax, unfortunately, but powerful, that allow both enable and disablement for IE to go to a third party. So you have the ability to block third parties, that is, to block IE9's following links to sites that you have blocked. And this doesn't, as I understand it, it doesn't even allow IE to go. It's not a matter of it, like, blocking tracking. It's blocking you going, which I think is fantastic. Websites may object to this. Something like this was going to be in IE8, but Microsoft removed it under pressure from advertisers, and so we didn't get that. So from IE's blog they say: "IE9 will offer consumers a new opt-in mechanism ('Tracking Protection') to identify and block many forms of undesired tracking. Tracking protection lists will enable consumers to control what third-party site content can track them when they're online."

So the reason this is troublesome is that it's not clear where these lists come from. For example, you might go to a site that is able to sense that you are not following its links to its ads, which scripting at the site could do, in the same way that a site is able to say, hey, you've got to turn JavaScript on if you want to use everything we're doing here. So a site could say you must load this TPL, and they would provide you with their own customized tracking protection list for them, which would get merged in, in some fashion, to provide overrides to your otherwise disabling this. Anyway, you get a sense for this, Leo. I mean, this just sounds like a disaster. I mean...

Leo: It is. It's worse than I thought, and I thought it was bad.

Steve: Yeah. So I don't know how this is going to settle out. I mean, again, it's why I just love the idea of a do-not-track header. Just, for users who know they don't want to be tracked, turn that on. I don't even care if it's off by default. Just turn it on once and have it be sticky, and so that way the browser is broadcasting, with every query it makes, it's saying do not track me. And then, yes, the argument is, well, but that requires compliance, voluntary compliance. And it's like, yes, until we have legislation in place, which it seems like we're going to have to have anyway. So the only alternative would be something like NoScript or cookie management or something like these TPLs, the IE9 tracking protection lists, which also seems like a big snarled nest of mess.

So anyway, this is - we're at the beginning of solving a problem which I'm glad the industry is looking at. I'm glad Congress is making noise. I'm so glad the browser vendors, I mean, everybody, this is like, it's not like no one's paying attention to this. Everybody is paying attention to this. Unfortunately, what that means is we've got 19 different approaches, and we haven't reached a consensus. But we've been there before. And in fact we'll be talking about the Document Object Model a little bit later when we talk about browser fuzzing. And boy, that was a disaster in the beginning. And it wasn't until the World Wide Web, the W3 Consortium stepped in and said, okay, here's how we're going to do it, that we finally got some sanity there.

And speaking of lack of sanity, Bruce Schneier blogged just the other day, I think it was yesterday, that a group of students at the Chinese University in Hong Kong have figured out how to store data in bacteria.

Leo: [Laughing] Okay.

Steve: They have an article, a web page that I have a link in our show notes, which talks about, they say, its "massively parallel bacterial data storage system, error tolerant data encoding and decoding system, recombination module for data encryption." And then, clearly, again, English is not their first language, not surprisingly. And then they said "Ready for the exciting future of Biocomputer." So in Bruce's blog he cites some other people that have looked at the article and scratched their head about what it is that they mean when they say "bioencryption," which is the term they've coined. Bruce's comment, predictably, was, "Well, I hope they haven't created some sort of new encryption, but rather they're using traditional encryption before they use their bacterial storage to store the encrypted data."

Leo: Yeah, because you can't encrypt bacteria. That's next.

Steve: Anyway, that's in the wacky - I guess I should have put this under errata.

Leo: Or just wacky. Wacky would be good.

Steve: Wacky, yeah, the wacky category, data storage in bacteria. And apparently, Leo, the bacteria can store a lot of data, so these students are quite excited about this.

What I do have in errata is a note about an interesting IPv4 countdown tweeter, or Twitter, or whatever that is, "@" sign. I guess it's...

Leo: It's a Twitter handle, yeah.

Steve: It's a Twitter handle. So it's @IPv4Countdown, just all run together. And this person is tracking the surprisingly rapid disappearance of IPv4 addresses. So I just thought our listeners, any listeners who are Twitter followers, and I know we've got about 16,000 because they're following me, might get a kick out of adding @IPv4Countdown to the handles that they follow. It's very low traffic. It only posts when there's been some loss of IPv4 address space.

Leo: It looks like it's every six hours, roughly.

Steve: Yeah. And it's dropping by millions.

Leo: Yeah, every six hours another million down. Wow.

Steve: I think we're, what, it's like 37 million, I think?

Leo: No, 29 million.

Steve: Ooh, wow, yeah. I guess it was yesterday that I looked.

Leo: Another one just came in, 28 million. That's really interesting. I mean, literally every - it's roughly every, well, it's every million, but that's roughly every six hours.

Steve: Yeah.

Leo: Wow, so somebody do the math. It ain't good.

Steve: Yeah, get your cell phone now. No, actually, it won't be a problem. Don't everybody - don't hyperventilate. Don't worry about it. In fact, when I posted this - I did tweet about this, and I said, don't get upset, I mean, this looks like we're running out of IPs very quickly. What this actually is, is this is the IANA allocating from their final batch. And we talked about this...

Leo: Is this, like, the last Class A?

Steve: Well, yes. This is, now, this is them giving away their blocks to the registries. This is not the registries giving those away. So this is - and we've expected that in the first couple months of 2011 the IANA would discharge the balance of the blocks that it had available to the major registries, ARIN and so forth, all the major NICs.

Leo: Is it IANA still, or is it ICANN now? I thought ICANN superseded IANA.

Steve: Okay, I think you're right, it's ICANN. I'm showing my age.

Leo: Jon Postel is no longer with us, I'm sorry to say. It used to be, it wasn't so long ago, there was one guy. Jon Postel at USC was the guy. That was it.

Steve: And all of those early RFCs had his name on them. I mean, when I was reading about, originally reading about the TCP/IP stack, that's where it came from, and email protocol and so forth.

Leo: By the way, Tom Johnson, who does our MailRoute stuff, worked for Jon, so he knows Jon - knew Jon.

Steve: No kidding.

Leo: Yeah. Which is a pretty good pedigree for somebody who does email spam fighting.

Steve: Yeah. And I do have a short note from another satisfied SpinRite customer who wrote to Greg/GRC, Greg of course being my tech support guy, who said: "I just wanted to send along a message of thanks for a great product. I was able to assist one of my customers in recovering data off a drive that I did not think was recoverable. When I got started, the drive was recognized by the BIOS, but that was about it. And setting the drive up as a slave drive in another machine caused Windows to claim that the drive was not formatted. Chkdsk failed to run completely."

He said: "SpinRite started out saying that it would take 4,500 hours to repair and recover the drive. So I basically gave up, but decided to let SpinRite run for a while. After 26 hours, SpinRite claimed that it was...." Well, SpinRite would have been 16 percent done, although I guess he's saying "claimed" because obviously the estimate is coming down rapidly. If it did 16 percent in 26 hours, then it's not going to be 4,500 hours for the whole drive. That's typically something that does happen when SpinRite...

Leo: Is slow at first.

Steve: Exactly, runs into trouble immediately. There's nothing SpinRite can do except assume that the drive's condition is going to be what it has seen so far. So it's continually reevaluating its estimate for completion based on the history of what it's seen so far. So at 26 hours it claimed to be 16 percent done. "On a whim," writes Peter, "I interrupted the process and tried the drive in a slave scenario again. At this point I was able to recover much of the data. I have since let SpinRite go back to work to finish the process. Even if it takes a few days, you have impressed me and made my customer extremely happy, as they had no backup of their data. Thanks again, Peter Elkins at Omnitech Computer Services."

Leo: Isn't that nice.

Steve: And Pete, thank you for the note.

Leo: We are going to get to the fuzzy browsers next. Steve Gibson, just a word of warning, you have about a half an hour. Is that enough time to get the fuzzy browsers unfuzzed?

Steve: I'm watching the clock, yeah.

Leo: Thank you.

Steve: That'll do it.

Leo: Steve Gibson today is talking fuzzy browsers, or browser fuzzing.

Steve: Yes.

Leo: What are we talking about?

Steve: We touched on this a couple times in recent weeks because Michal Zalewski, who is a Google security guy, released his what he called "Cross_Fuzz" for browsers. And our listeners will remember that Microsoft tried to - lobbied him to not release this at the beginning of 2011, as he had notified them six months before he was going to, because he provided them with his tool in June or July of 2010 with the plans to release this at the beginning of 2011, and telling them that his browser fuzzer found some fuzziness in their browser, that is, in Internet Explorer, and that they should take it seriously and see about fixing the problem. He never heard back from them until the middle of December, shortly before it was deadline time for him to release his browser fuzzer. And they said, hey, wait a minute, now we're able to reproduce the problems which we weren't able to reproduce before. So we haven't yet had a chance to fix them, so please don't let the world know about these problems, which would happen when you release the fuzzer.

However, it turns out that some malware authors, it is strongly believed, independently discovered the problem because, looking at Google search history, Michal was able to see that there had been some people, I don't remember if it was Russia or China, I think it was in China, that had been doing some search queries that clearly indicated that they knew where this problem was in IE. So it didn't come from him. It was already out in the wild. So he defended his decision to release, I mean, notwithstanding Microsoft's rather lame, well, you only gave us six months, and we didn't take you seriously until two weeks before you were going to release this approach.

Okay. So was is all this fuzzing? Fuzzing is a software-testing technique which essentially sends malformed junk, maybe invalid data, unexpected data, or just pseudorandom data, or all of the above, to the inputs of some sort of interpreter or parser, which is expecting

to receive normally good data that it's been designed to receive, specifically for the purpose of seeing whether that software can be destabilized by inputs that it was not designed to handle.

Wikipedia tells us that the term "fuzz" originated for the first time in the fall of 1988 from a class project topic in a Professor Barton Miller's graduate University of Wisconsin advanced operating system class. The assignment was titled "Operating System Utility Program Reliability - The Fuzz Generator," which is where the term first arose. And so his group of graduate students experimented with this concept of throwing junk at operating system utility programs to see if they could break them.

And, famously, we'll remember that Mark down at eEye in Aliso Viejo, and eEye themselves, had a lab set up where they were essentially doing fuzzing of Windows and finding Windows problems, problems in the Windows API, by calling functions in Windows, providing it with unexpected data, and then when one of - and they had a whole lab set up of machines running. And every so often one would crash. And when that would happen, they would look essentially at the audit trail of what had been sent to cause Windows to crash. And then that would give them clues for looking closely at what it was that their automated basically noise generator had done to cause the crash. And this is one of the ways that eEye has found and uncovered many security problems in the Windows operating system.

Now, this is a different approach to normal code testing than has traditionally been done. What normally happens is that there are essentially two parallel coding efforts. First you have a specification for some software that you want written. And the specification determines, specifies what it is that the software will do. So you have one team, normally the big team, of coders whose job it is to code that spec, that is, to turn that specification into functioning code.

You then have normally sort of like the also-ran group, and they don't get as much glamour. They're not normally held in as high regard. But they're the test suite coders. I would argue that, in this day and age, they need as much glory and as much attention as the main product coders. The test suite coders have the job of writing code to test the code that the main code writers write. So they're also working from the same specification. But their job is to essentially create an automated test suite which, once the product coders say, okay, we've read the spec, we've very carefully coded it, our code does what it's supposed to. Well, so how do you know that? Well, a user could play with it and click on things and see. What's much more efficient, though, is if you have, I mean, one for one, for every chunk of code that you have, you have a test suite.

This is very common, for example, in the crypto field, where you have a bunch of known test data which you encrypt under a given key, and then you know what you should get out if the code is correct. It's the way we verify that cryptographic code is working the way it should, is we run standard patterns through. So formal test suite development is something that has been done for years. It's sometimes called "regression testing." It's something that Microsoft does is they have a huge test suite, so they want to make sure, when they're in Windows, rummaging around and changing things, they haven't broken something that used to work.

Well, the only practical way to do that, you can't just have a ton of users sitting there trying to see if Windows is broken. You need something to exercise Windows. So these exercisers, these test suites have always been created. But notice that they're being created from the same specification that the coders use. And while the test suite guys can try to find problems, generally they have the same motivation that the main product coders do, that is, they're working to show that the code meets the specification, rather

than not.

Leo: There's a kind of a theory in coding these days, test-driven programming, where you start by writing a test, which fails immediately, and then you write the code to fulfill a test. But it doesn't test for these kinds of things. It tests exactly, as you say, I mean, it's a good way to code. You have fewer bugs, but it really only tests to see if the code does what it's supposed to do. Doesn't see if there's a buffer overflow or something like that.

Steve: Well, in fact in my own history, a little bit of GRC lore that no one knows, is after SpinRite was up and running and launched - and remember, Leo, I've talked about how I got - GRC grew to about 23 people, I think it was, at its largest. Well, I had an R&D department with four or five programmers. And we were working on something to do next. This was back in the days before caching was even part of the operating system. You had, like, Windows had SmartDrive, or DOS had SmartDrive that it came with later. And then there was SmartDrive in Windows. There were third-party caching utilities which dramatically sped up the operation of the system.

Well, GRC developed one called Propel. And it was, as you might imagine, better than all of the others. The programmer who wrote it was really gifted. I mean, this guy was a spectacular, world-class programmer. Unfortunately, Propel, to do what we wanted it to, was very complicated. Well, I was the guy who wrote the tester for it. And so I wrote sort of a generic program which read and wrote in a very pseudorandom, confusing, worst-case, data-pattern-checking kind of thing, to and from the disk.

The idea would be that, if the cache was not there, then the program would be writing actually to the hard drive, and it would be creating sectors with pseudorandom data and reading and writing and overlapping. And then when it read it, it would check, it would read back the data to verify that it was what it expected to have stored in that sector. The idea being that we needed to verify that inserting this cache in between my program and the hard drive didn't change anything. That is, everything still worked. The problem is, it would run for a while and crash. That is, there was some problem that introducing Propel, this never-shipped - and now we know why - never-shipped hard drive cache into the chain caused to fail.

Leo: Well, timing is so complicated in hard drives. I mean...

Steve: Oh, my goodness. And actually for all kinds of reasons Propel never saw the light of day. I wouldn't say that it was ready. It wasn't like this was the only thing that kept it from being a product. But I had my own experience with this kind of approach and can vouch for how good it is.

So, Cross_Fuzz. When Cross_Fuzz was created last summer, it found more than 100 problems in every single browser. No browser was unscathed. The largest number of bugs were identified in Firefox. Michal himself found 10, and then another 50 were found by the Mozilla people when they integrated Cross-Fuzz into their existing testing platform. The Firefox issues have since largely been addressed, so we're talking, like, 60 problems that have been found have largely been addressed. But some more obscure and hard-to-analyze crashes are still occurring. So even today, six-plus months later, if you run Cross_Fuzz on Firefox, you can kill Firefox.

Leo: Really, wow.

Steve: All WebKit-based browsers were brought to their knees, with approximately 24 crashes found. The developers were notified in July of last year, 2010. And since then all relevant patches have been released, yet some very subtle problems persist. And Opera was the other target of Cross_Fuzz. And Opera's authors were notified also at the same time in July of 2010. And all of the frequent crashes found by Cross-Fuzz were fixed in Opera 11.

So, what is Cross_Fuzz? How does it work? To understand that we need to dip a little into a topic we've never covered before, which is the so-called Document Object Model, or the DOM, of browsers. When scripting was first created, that is, JavaScript, which was originated by Netscape for the senior Netscape browser at the time, I think it might have been 2 or 3, one of the very early Netscape browsers, the idea was they wanted to create scripting on the client side so that a website could download, not only a static page, but also some scripting which the client, the browser, would execute, it would interpret, in order to make the experience more dynamic for the user. And of course we know that that's been a mixed blessing because scripting is a real security problem, also a tremendous benefit.

The problem is that pages at the time were described by HTML, the so-called Hyper Text Modeling Language. And HTML would describe paragraphs and frames and tables and forms and basically all of the content of the web page. But there was no means for a script running on that page to access the elements of the page, for example, to do fancy effects if you rolled your mouse over something; or to be able to notice, like, check an input field when you were asking for a credit card number, to quickly check to see whether it meets that rule of nines, the way credit cards sort of have a built-in checksum where you can instantly tell whether there might be a typo, a digit transposition, for example. So there were useful things that you would want to do, but no way for the script running to access the content of that page.

So the good news is we are many years forward from those bloody early days where Microsoft had their own approach, which was incompatible with Netscape's, and browsers that were emerging were inventing their own. It wasn't until we finally got some standards imposed and the browsers rather reluctantly, over the course of years, began to converge on a single unified standard. We have that now, called the Document Object Model.

And essentially what browser layout engines which receive HTML, they've basically become HTML parsers, which parse the incoming HTML into the Document Object Model description. That is, the way things have evolved is that browsers have become so complex; pages have become so complex; HTML, especially with the addition of CSS, cascading style sheets, have gotten so complex that there is now sort of a formal tree-structured container for a page. And when I say it's tree-structured, the root of that tree is sort of the anchoring document. And it goes by the term "document." And then the tree form is sort of described through dotted suffixes.

So, for example, you'd have document dot, and then the name of a form, and then dot, the name of an input field [document.formName.inputName]. Or you might have a table, and then rows and columns, essentially forming what's called a namespace where every single element of the page can be described through this dotted textual representation. And that's incredible powerful because that then allows scripting, which is running on the page, part of that page, a means for manipulating the page's content.

That is, so while the website loads down the initial HTML, the browser layout engine parses the HTML into this Document Object Model description of the page; and then scripting, which is also part of that HTML download, is itself able to access the entire contents of the page because the HTML describing the page has been translated into this DOM, this Document Object Model, which because there is this textual namespace representation, allows the scripting to sort of access itself.

So what Michal's Cross_Fuzz algorithm does is it opens two windows containing the same document. And this could be an HTML window, an XHTML, or an SVG, a Scalable Vector Graphic document, which itself is a distressingly complex format. It then - and I'm reading from his description. "It crawls the DOM hierarchy of the first document, collecting encountered object references for later reuse. Visited objects and collected references are tagged using an injected property to avoid infinite recursion; a secondary blacklist is used to prevent navigating away or descending into the master window." And then, "Critically, random shuffling and recursion fanout control are used to ensure good coverage.

So what that means is basically he's written a script which explores itself, that is, it explores the window that it's being targeted at to basically build a database of all the contents of the window." Then he says he "repeats the DOM crawl, randomly tweaking encountered object properties by setting them to one of the previously recorded references, (or, with some random probability, to one of a handful of hard-coded 'interesting' values)." Like a super-large value, super-small value, negative one, zero, positive one, so he's got a collection of things that generally cause problems. Then he repeats this crawl of the Document Object Model again, randomly calling any object methods which he has encountered. So he's also invoking JavaScript on the page that he's basically giving a real hard time to.

He calls parameters, or "Call parameters [to those object methods] are synthesized using collected references" and, again, "interesting values," as he noted above. "If a method returns an object, its output is subsequently crawled and tweaked in a similar manner." So if calling one of these methods, which is code on the page, if the method returns an object for the page, then he grabs that and crawls that, as well.

So essentially what he's done is he has built something to just give a seizure to, I mean, to just work the heck out of the code, which is hopefully always going to behave itself, never going to have a problem and, you know, just say, yeah, okay, fine, what you asked for was illegal. Or, yeah, that value doesn't make sense. We're going to ignore it. Instead, what often happens is, well, in fact there is not today a browser which can survive. One way or another, this thing is so horrendous to what it does, it just wrings the code out, and ultimately browsers collapse.

So there are problems with this, that is, and there have always been problems with the fuzzing approach because making something mysteriously break is very different from finding a bug. You haven't found a bug, you've found a symptom. And so...

Leo: Yeah, but you've got a foot in the door.

Steve: Yes. You're right. And what he's doing is he's auditing what he does. He uses pseudorandomness. And we know what that means. That means that, if you remember the seed, the pseudorandom seed that you used for driving your pseudorandom decisions, and if you start from a known good fixed condition, you can theoretically

exactly reproduce what happened because it should be deterministic. So still, though, this is the problem is that this thing might run for a day, and then the browser finally gives up in defeat. It says, okay, I just can't do this anymore.

Leo: I can't do it, I can't do it.

Steve: Now, the problem is, you haven't found the bug. You've just made something die. And, for example, we never found the problem with Propel. I mean, this drove Mike, the brilliant coder, to distraction. I mean, it would run and run and run, and then it would have a problem. It's like, oh, this sector didn't match. It's like, well, we don't know why. And we're sorry, but all we can tell you is it didn't work. But we don't know why.

So the good news is there is this insanely aggressive tool that now exists which the Mozilla guys, to their credit, have incorporated into their own test suite, which kills Firefox, kills Opera, kills Chrome, it kills anything you let loose. I mean, it's the destroyer. It has succeeded through the last six months of its use in rendering our browsers far more robust than before. That is to say, it takes it a lot longer to kill them than it did before, meaning that all of the problems that sort of were findable have been found. And the coders learned a lot from having this thing, turning this thing loose on their browsers. And they now turn it loose routinely, hoping to see - hoping it takes a long time before it brings the browser to its knees. Ultimately, it does. And that's browser fuzzing.

Leo: That's really interesting stuff because it is so closely related to test suites, but it's just going the extra mile, really banging on it.

Steve: Right. The test suites are - and this is what - the advantage of fuzzing is that, although...

Leo: You're testing it for something else, I guess.

Steve: Well, now, it's certainly not just throwing noise. I mean, that is to say, a huge amount of code had to be written to implement this technology which walks this Document Object Model hierarchy.

Leo: Is it safe to say it's a stress test?

Steve: Oh, it is really, yes, that's exactly what it is. It is really a stress test. Browsers should be able to pass it. None can. The reason is that this is, I mean, if nothing else it demonstrates how incredibly complex browsers have become in order to do everything that we ask them to do. I mean, browsers really are - there's a huge amount of code in them. I mean, in order to run JavaScript, they're doing just-in-time compilation of a language into the native machine's tokenized pseudo-language, which an interpreter then runs. I mean, we take it for granted. We go to a website, and it just works. But, I mean, there is a huge amount of stuff now under the covers. And we know for the sake of security that it has to be robust. The problem is, it's difficult to make it so. So this Cross_Fuzz system, I mean, I'm really happy that it exists because...

Leo: And it's not just, I guess the point is it's not just used by bad guys. It would be used by anybody who wanted to test their system; right?

Steve: Yes. And in fact that was the controversial aspect of it is that Microsoft didn't want it released because IE was being brought down. And if the bad guys - so everybody could run it. The bad guys could run on IE and use it potentially to find new problems. So now there's a race on, can Microsoft - I mean, and here's the problem, is when it brings the browser down, again, I mean, I can't emphasize this enough, you really don't know why. You just know that it died. And this was why Microsoft couldn't make it fail last July. They really - they didn't wait long enough, or...

Leo: Give it time.

Steve: ...give it time, exactly. And what they found was that it required several different documents to be fuzzed in a row before the final document fuzzing caused the problem. So that meant that you couldn't just run, you couldn't just fuzz the last document. You had to, like, reproduce the footsteps that Michal had used to get there, which meant that some state somewhere was, like, bleeding across between documents. The point is that, as we know, crashes beget malicious attacks. Crashes beget exploits. And so Microsoft was worried that a bad guy would run this and be better than they were at figuring out exactly what it was that the fuzzing found and then turn that into an exploit that they weren't ready for.

So anyway, this thing exists. Browsers today, six months after its release privately to the browser authors, are substantially more robust than they were. And that's why then Michal said, okay, it's 2011. I'm going to publish this, let it out for the world to play with. And browsers are ready for it, much more so than they were six months before, when it just knocked them all on their butts.

Leo: Very interesting stuff. Browser fuzzing. Steve Gibson is at GRC.com. That's the place to go for SpinRite, the world's finest hard drive maintenance and recovery utility, kind of does both. He also has a lot of free stuff there. But you'll also find show notes for this show, transcripts. You'll also find 16KB versions there for the bandwidth-impaired, and a whole lot more.

Somebody in the chatroom said we should someday get a tour of Steve's bookshelf. If you watch on video, you see all those books behind Steve. I bet there's some classics there. And the blinking lights, those are the PDP-8s.

So, Steve, thank you. We will see you next week. And we'll do a Q&A. So if you've got a question for Steve, maybe about this subject or anything else we've covered, just go to GRC.com/feedback. GRC.com/feedback. Steve, see you next time on Security Now!.