SECURITY NOW!

Transcript of Episode #906

# The Rule of Two

**Description:** This week we're back to answering some questions that you didn't even know were burning. First, is the LastPass iteration count problem much less severe than we thought because they are doing additional PBKDF2 rounds at their end? What sort of breach has Norton LifeLock protected its users from, and have they really? What did Chrome just do which followed Microsoft and Firefox? And is the Chromium beginning to Rust? Will Microsoft ever actually protect us from exploitation by old known vulnerable kernel drivers? What does it mean that real words almost never appear in random character strings? And what is Google's Rule of Two, and why does our entire future depend upon it? The answers to those questions and more will be revealed during this next gripping episode of Security Now!.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-906.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-906-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. Lots to talk about. An update on the LastPass breach. Steve thought there was a saving grace. Well, I'll let him tell you the story. Norton LifeLock says it saved you from something. But what, really? And a look at Rust and how it's helping Google make Chrome safer. All of that and more coming up next on Security Now!.

**Leo Laporte:** This is Security Now! with Steve Gibson, Episode 906, recorded Tuesday, January 17th, 2023: The Rule of Two.

It's time for Security Now!. I know you've been waiting all week for this. Steve Gibson's back. And guess what? We're not going to talk about LastPass this week. Hi, Steve.

**Steve Gibson:** We actually are.

**Leo:** Oh, no. Well, I'm sure there's some follow-up. But we have other stories.

**Steve:** Yes, we do. We actually do. This is Security Now! #906 for the middle of, what is this, January. And this week, as you said, Leo, we are back to answering some questions that you didn't even know were burning. First, is the LastPass iteration count problem much less severe than we thought?

**Leo:** Oh, okay.

**Steve:** Because they are doing additional PBKDF2 rounds at their end.

**Leo:** Well, that would be nice to know.

**Steve:** Wouldn't that be nice? What sort of breach has Norton LifeLock protected its users from? And have they?

**Leo:** Oh, yeah, I saw that story. Hmmm.

**Steve:** Hmmm. What did Chrome just do which followed Microsoft and Firefox? And is the Chromium beginning to Rust? Will Microsoft ever actually protect us from exploitation by old known vulnerable kernel drivers? What does it mean that real words almost never appear in random character strings? And what is Google's Rule of Two, and why does our entire future depend upon it? The answers to those questions and more will be revealed during this next gripping episode of Security Now!.

**Leo:** Now, that's a tease, Steve. Congratulations. Golf clap. Very well done. I can't wait to find out all of the answers to those questions.

**Steve:** Ah, there are some goodies.

**Leo:** And a good Picture of the Week.

**Steve:** Ah, yes.

**Leo:** Picture time, Steve?

**Steve:** It is. So here we have yet another puzzle for our listeners.

**Leo:** Oh, it's not so puzzling.

**Steve:** We have a sidewalk path running from the right foreground to the left background of this photo. To one side is sort of a knoll covered with dried grassness, weedy sort of stuff. Over on the left is a not-very-well maintained typical green grass, which is now dried out. Okay. So that's sort of the setting. Now, right in the middle, I mean across this sidewalk, is a permanently installed Sidewalk Closed fence. This is not a gate. If you look at it, there's a steel pole on either side of the concrete sidewalk, and then there's a well-built wire gate which is strapped between these poles.

**Leo:** Oh, it's good. They didn't stint on this. They put in the best.

**Steve:** No, honey, this is meant to stand the test of time. And there's a sign. In case you were curious about, like, okay, why is my path blocked on this sidewalk, across the front of this wire it says "Sidewalk Closed." Sorry. You're SOL, as they say. So you're confronted with this gate. And it's like, oh, well, I guess I can't proceed. I had my heart set on going into the distance here and around the corner. But the sidewalk is closed. Except that, like, it's not. Because you could choose the direction you wanted to go around this small hurdle in your way. And there's, like, sidewalk behind it. The sidewalk looks just fine.

**Leo:** Well, that was before it was closed.

**Steve:** I mean, if they wanted to close it, why not remove it? I mean, like, you know, get out the jackhammer and break this concrete up and haul it off. But no. It's there.

**Leo:** It's a puzzle.

**Steve:** A perfectly functioning sidewalk which no one can use unless they walk around this barricade. And, look, oh, it works. The sidewalk still works, Leo. It's like once I remember, this is a long time ago, I let my car insurance lapse.

**Leo:** Yes.

**Steve:** And I thought, oh, I can't drive. But I got in the car and turned the key, and it still worked.

**Leo:** Still works.

**Steve:** It was amazing.

**Leo:** It's a miracle.

**Steve:** Anyway, yeah. So, wow. Okay. I originally was going to title this podcast "A Brief Glimmer of Hope" over a pursuit that took me, I think it was like six or seven hours until I realized how it turned out, and how brief the glimmer was. But it's an interesting issue, and it could come up. It already had come up. So, and I figure, you know, since all LastPass users could use a bit of good news about now, I became excited when it appeared that things were not as bad as we thought last week. As we know, many users discovered that LastPass had never increased their clients' local PBKDF2 iteration count from its earlier setting of 5,000, or maybe 500, or in many cases, and they keep mounting, I've seen a lot more reports in the last week, as few as one iteration, which of course results in a trivial-to-bypass encryption of LastPass backups.

Well, one of our listeners with a sharp memory, by the name of Parker Stacey, wrote with an important quote from a page on LastPass's website. The page Parker quoted is titled "About Password Iterations," and most of this we already know. But there's an important line on this page from their website that was news to me. Okay. So in order for that important line to be understood in context, which is somewhat unclear, I'm going to share this short piece in its entirety.

So this "About Password Iterations" on the LastPass site says: "To increase the security of your master password, LastPass utilizes a stronger-than-typical version of Password-Based Key Derivation Function (PBKDF2). At its most basic, PBKDF2 is a password-strengthening algorithm that makes it difficult for a computer to check if any one password is the correct master password during a compromising attack." Right.

They say: "LastPass utilizes the PBKDF2 function implemented within SHA-256 to turn your master password into your encryption key." All this we know. "LastPass performs a customizable number of rounds of the function to create the encryption key, before," they said, "before a single additional round of PBKDF2 is done to create your login hash." Then they said: "The entire process is conducted client-side. The resulting login hash is what is communicated with LastPass. LastPass uses the hash to verify that you are entering the correct master password when logging into your account."

Okay. This is the next line is what, like, I go, what? They said: "LastPass also performs a large number of rounds of PBKDF2 server-side. This implementation of PBKDF2 client-side and server-side ensures that the two pieces of your data - the part that's stored locally and the part that's stored online on LastPass servers - are thoroughly protected." Then it said: "By default, the number of password iterations that LastPass uses is 100,100 rounds. LastPass allows you to customize the number of rounds performed during the client-side encryption process in your Account Settings."

Okay. So, what? It's like: "LastPass also performs a large number of rounds of PBKDF2 server-side." So, okay. I thought, that's not anything we've talked about or looked about. So it also wasn't anything that I recalled Joe Siegrist ever mentioning to me. And since it perfectly responded to the worries we talked about last week, I mean, that's what named last week's episode "1," and since LastPass has now been revealed to be, if nothing else, a bit klutzy, I was actually just a bit suspicious about exactly when this convenient page first appeared on LastPass's website. So I dipped into the Internet Archive's Wayback Machine to see when it had first indexed this page. And what I found did little to assuage my suspicion.

In the show notes I have a picture of the Wayback Machine's calendar showing that this page didn't exist before 2022, and in fact it was first indexed by the Wayback Machine on July 2nd of 2022, about six weeks before we learned about this particular problem. And so, okay. I was a little suspicious about that. It's like, oh, isn't that interesting that some good news for this problem was first indexed then. So, you know, it did precede public disclosure of the trouble. It's not as if, however, that page had been around for years.

I shared my curiosity over this with Parker, the guy who first brought this to my attention, and I had the intention to do some more digging before today's podcast. But Parker's curiosity was also piqued, and he tracked down a LastPass PDF document titled "LastPass Technical Whitepaper," which the Internet Archive had indexed and stored on December 18th of 2019, so way longer ago, four years ago. And looking at the same titled paper today, because there's also something by the same name, "LastPass Technical Whitepaper," if you Google that, it pops right up.

None of the PDFs that were found contain an edition date, but both papers contain exactly the same paragraph, which is more clear than the watered-down help page that I just read. Both papers, both PDFs, the old one from 2019 and today's, say: "LastPass also performs 100,100 rounds of PBKDF2 server-side. This implementation of PBKDF2 client-side and server-side ensures that the two pieces of the user's data - the part that's stored offline locally and the part that's stored online on LastPass servers - are thoroughly protected."

Okay. So I didn't want to give anyone the wrong impression from last week, so I had to pursue this. I was curious about anything additional that that particular PDF's metadata might reveal. I was, again, still like, okay. I learned that it was created by Microsoft Word 365 by Erin Styles on June 6th of 2022. And as a quick sanity check, I noted that Erin's Twitter photo shows her proudly sporting a bright red LastPass sweatshirt.

Okay. So what would the flow of this system look like? What does it mean for LastPass to be performing apparently another 100,100 iterations of PBKDF2 server-side? So as we know, to LastPass's client, the user provides their email address, which serves as the salt, and their LastPass master password, which is the input, along with their account's default iteration count to an SHA-256-based PBKDF2 function. That produces a 256-bit encryption/decryption key which is never sent to LastPass. Right? I mean, that's the whole point is LastPass never has that key. It is only ever used by the local client to encrypt and decrypt the master vault blob of data which LastPass stores for them and shares among the user's various LastPass clients in order to keep them synchronized.

Okay. So how, then, does LastPass verify that they've logged on using their proper credentials, if that key never goes to LastPass? Well, after running that local PBKDF2 iterations, one additional round of PBKDF2 is used to produce a 256-bit user logon verification token which is what LastPass calls the "login hash." That token is sent to LastPass to store and use to subsequently verify the user's proper logon credentials. LastPass uses the user's ability to provide that login hash token to log them into their online vault and before sending their encrypted vault blob to them. That's how they avoid sending the vault that does contain a bunch of private unencrypted information to other people. You've got to be able to provide this login hash.

So the problem that occurred to me was that only a single round of PBKDF2 was separating the user's secret, like super secret, vault decryption key from this login hash, which is what the client provides to LastPass as its authentication. That meant that in theory it could be easily brute forced by reversing that hash function. But this apparently also occurred to LastPass. Although there's language about this in several places in this whitepaper, the clearest description appears near the bottom of page 19 under the title "Login Hash Storage," where they write:

"LastPass receives the login hash from the user," and they said, "(following the default 100,100 iterations on the user's master password using PBKDF2 SHA-256)," meaning on the client-side. "The login hash is additionally salted with a random 256-bit salt, and an additional 100,000 rounds of PBKDF2 SHA-256 are performed. That output is then hashed using Scrypt to increase the memory requirements of brute force attacks. The resulting hash stored by LastPass is the output of 200,101 rounds of SHA-256 PBKDF2 plus Scrypt."

Okay. So what they're saying is they recognized the reversibility of the login hash only being one round away from the user's secret key creation. So when they receive the login hash from the user, that's what they run an additional 100,000, a salted 100,000 rounds of PBKDF2 SHA-256 on in order - it's because they're going to store that permanently, and they don't want there to be any chance of it going backwards to the secret key. So I needed to pursue this all the way out to the end to understand whether, as I was hoping, LastPass might have used all of this additional, and again, as I said, they talk about it all over this whitepaper, server-side PBKDF2'ing - even going so far as to deploy Scrypt - to create a super-strength encryption key for use when saving their users' vault data. But there was no sign of that anywhere in the whitepaper.

**Leo:** So for my understanding, they're protecting it from a bad guy trying to log in as you. But if the bad guy's got the vault, it's independent of that. Is that right?

**Steve:** Correct. That's exactly right.

**Leo:** And you mentioned Scrypt, but I want to make it clear because of the way you said it, there's no evidence they used Scrypt. They used PBKDF2; right? They didn't mention...

**Steve:** Actually, they're saying they used both.

**Leo:** Oh, good. But not on the right part.

**Steve:** Well, correct. Exactly. Exactly. So what they get from the client is they don't get the secret key ever. But they do one more round of PBKDF2 on the client, and the client sends them that, which they call the "login hash." So that's their token is that the client uses to verify the login. And since they're going to be storing that for the long term, they realized, oh, this is not safe for us to store because it's only one PBKDF2 iteration away from the key.

**Leo:** Right.

**Steve:** So let's hash the crap out of this, and that's what we'll store. So anyway, my hopes were dashed. For a while I was thinking that they were doing something really cool, that they were going to take what they got from the user, hash the crap out of it, and then reencrypt the vault under that next-generation key, in which case people with an iteration of one on their client...

**Leo:** Would still be okay.

**Steve:** ...would have been protected, yes.

**Leo:** But they're not protecting that vault. They're just protecting the login.

**Steve:** Correct. And in fact there is a place where they specifically say, under a section of their page "User Data Storage," they say: "Sensitive vault data is encrypted client-side, then received and stored by LastPass as encrypted data." So I was thinking for a minute that I might have steered everybody wrong a week ago, that one iteration meant one iteration. Well, unfortunately, it does mean one iteration. And things are as bad as we thought last week.

**Leo:** As good as we thought. This is part of the problem with all of this is the marketing.

**Steve:** Yes.

**Leo:** And there's a lot of hand-waving. Different password managers have different amounts of hand-waving. But the marketing department, which probably doesn't understand what it's saying, usually takes that the technical people have given them, does some magic hand-waving, and what we get is often not very useful. Look at the digging you had to do for this.

**Steve:** When we talk about - we're going to be talking about Norton LifeLock problems in a minute.

**Leo:** Oh, this one, yeah.

**Steve:** Oh, and this is exactly to the point you're just making. At one point they said: "We have secured 925,000 inactive and active accounts that may have been targeted by credential stuffing attacks." Uh, you secured them? What does that mean?

**Leo:** No, they sent you an email.

**Steve:** What? What?

**Leo:** Anyway, we'll get to that.

**Steve:** Yeah, okay.

**Leo:** There's a lot of that. And in fact...

**Steve:** I feel so much better.

**Leo:** I'm trying to do the same thing. I'm not as knowledgeable or as adept as you. So I'm trying to filter through some of this marketing material, too, from other password managers, to see what's going on. I do notice that when I rekeyed my Bitwarden vault, I gave it a new password, they said you can rekey it, you can regenerate a secret, but that's going to mean we have to do a bunch of stuff. It's going to take a little while. And make sure you log out everywhere because once we do that we could corrupt your data. You could corrupt your data if you haven't logged out. And we'll log you out, but just make sure we did successfully. What is that?

**Steve:** Okay. That's a really good point. And it's a subtlety that I was deliberately skipping over, just because I didn't want to really just boggle people's brains.

**Leo:** Oh, boggle us, Steve, boggle us.

**Steve:** There is the key that you use for decryption, and there's the key that you use for decrypting the key.

**Leo:** Okay.

**Steve:** So all of this that we've been talking about in LastPass, for example, where you're doing this PBKDF2 to get a key, that key that you get isn't the actual decryption key for the vault. It's the key that decrypts the key. So there's a level of indirection there. And the reason that's done is that you're able to change your encryption key without having to rekey the vault. And thanks to that level of indirection. It's very much like if you had a password protecting a hard drive, and you wanted to change the password. Well, if you actually change the key, you have to decrypt...

**Leo:** You have to reencrypt everything.

**Steve:** ...the whole hard drive and then reencrypt the whole hard drive. So instead, nobody does that. You assume that the actual key was achieved through a very high-entropy, pseudorandom, hopefully actually random, bit generator, and then that's the thing that's encrypted by the key that you use.

**Leo:** Right.

**Steve:** And then decrypted when you want to use it. And so that's also what you were seeing with Bitwarden where they said, well, we could actually rekey your vault.

**Leo:** I did that. I did both, yeah. Why, I don't know. I mean, I just thought I would just to see what happened.

**Steve:** Sounds good.

**Leo:** Then the other thing a lot of people - no harm done. No harm done. Then something a lot of people mentioned is that Last Password has an additional key that they generate that is stored, as I understand it, in the hardware or...

**Steve:** I think you mean 1Password?

**Leo:** Did I say Last Password? I meant 1Password, yeah.

**Steve:** Yes.

**Leo:** In their Agile Keychain. And it's actually a file, you get it as a file that you store somewhere, and then you use as a secondary encryption. And, now, correct me if I'm wrong because I was talking because I was talking about this on our Ask the Tech Guys show, the new show we do to replace the radio show, Mikah and I, and I said, that's great if you don't have a good master password because then there'll be a backstop for a bad master password. But if you have a very, very good master password, I mean, something that's impossible to break, isn't more impossible to break with a second master password; right? I mean, one would be good enough. It does add some complexity and confusion because people have to save this key as a file, and they have to put it on - you know. So it's great. And I said, if you're going to have naive users use a password manager, and you think they might use monkey123 as their master password, this would be good.

**Steve:** Yes. Because what it means is that the actual password is the composite of some true high-entropy file and monkey123.

**Leo:** Yes.

**Steve:** So that if the vault ever escaped, then there's just no point in trying to do any sort of, you know.

**Leo:** It's another wall.

**Steve:** Yes.

**Leo:** But my position is, well, that's fine, and if you're worried you could do that. But if you have a good master password, which you should, long, random master password, and you're going to talk about what random means in a bit, that's probably belt and suspenders. It's not needed.

**Steve:** I agree. So, again, convenience versus security. It is technically more secure, but a lot less convenient. And the point is at what point does more security not actually buy you anything? Right? You know, where all you're doing is making things way less convenient; but you already, even without that, you already had enough security. Your point is, if you really use a good random master password, you're done. There just isn't any need for anything more.

**Leo:** Okay.

**Steve:** Especially at the cost of significant inconvenience.

**Leo:** Yeah. So that secret key is nice, but better even - and it cost, by the way, it cost an additional amount, which is I think the real reason they offer it. You have to pay a subscription fee to get it.

**Steve:** Oh, my god. Okay.

**Leo:** But if somebody is going to use monkey123, then they should be using 1Password to do this. Or train them in how to make a good password. Which Steve will do. I'm sorry. On with the show.

**Steve:** Okay. So the other - no, this was all good. The other question is, ECB or CBC? That was the other question to be answered this week, thanks to the past week's worth of listener feedback from all the people who used Rob Woodruff's PowerShell script to peer into their vaults. What we wanted to find out was about the prevalence of the less desirable ECB encryption mode.

Well, one of our listeners provided a unique insight which simultaneously answered a question I had. Mark Jones tweeted. He said: "You requested updates on LastPass." He says: "Regret as a loyal listener that I was stuck at 35,000 iterations, not 100,100." Still, 35,000's not bad. He said: "I found 28 ECB items, mostly secure notes." He said: "I now certainly regret putting so much in notes. A couple of comments," he said. "Incrementing iterations" - and he actually meant changing iterations, but okay - "changed all to CBC." He said: "It appears you're correct in asserting any changes get rid of ECB."

**Leo:** That's good to know.

**Steve:** Well, I didn't assert that, so he's giving me credit for something I didn't say. But thanks. But this explains the mystery of why my own vault had no instances of ECB. I know that my vault contained a number of very old account credentials from the beginning of my use of LastPass which would have never been updated, and which I therefore expected to find still encrypted in the original ECB mode. But what Mark observed was that simply changing his vault's iteration count, as a lot of us did, many of us did five years ago when we were covering this issue on the podcast, that gave our clients the opportunity to reencrypt all of those older ECB mode entries in CBC mode.

**Leo:** Good news. Good news.

**Steve:** So Mark, another mystery solved. Thank you.

**Leo:** Yeah, because I've heard a lot of our listeners, you've heard many more, but I've heard a lot of them say, "I don't have any ECB stuff." But that would make sense. That's why, yeah, because a lot of them were long-time, like you and me, long-timer users.

**Steve:** Yup. Okay. So while we're on the topic of password manager troubles, we should touch on Norton LifeLock and more doublespeak being produced by corporations that have grown too big to need to care. Before we get to this, we should note that what was formerly Symantec Corporation and Norton LifeLock are now renamed Gen Digital. And they just refer to themselves as Gen. So now they're both living at GenDigital.com. And since Symantec had been acquiring companies over the past few years, this new Gen Digital is now operating the brands Norton, Avast, LifeLock, Avira, AVG, Reputation Defender, and unfortunately CCleaner. Which, you know, was a beloved tool for a long time.

Okay. So what happened this time? Around 6,450 Norton LifeLock customers were recently notified that their LifeLock accounts had been compromised. And by compromised, we mean that unknown malicious parties have somehow arranged to log into those 6,450 accounts, giving them full access to their users' password managers' stored data. Whoops. That's not good. In a notice to customers, Gen Digital, as I said, the recently renamed parent of this collection of companies, said that the likely culprit was a credential stuffing attack as opposed to a compromise of its systems.

Now, this seems very odd since Gen Digital explained that by this they meant that previously exposed or breached credentials were used to break into accounts on different sites and services that share the same passwords. But that wouldn't explain why there was apparently what appears to be a quite successful targeted attack against 6,450 of their users. If these were username and password credentials leaked and/or somehow obtained from other unrelated site breaches elsewhere, how is it that they just happened to all be useful against 6,450 of LifeLock's account holders? You know, that doesn't smell right.

What I suspect actually happened is that LifeLock's web portal is, or was, lacking in brute-force password guessing protection. In this day and age, it is no longer okay for a website to allow a fleet of bots to pound away on its login page at high speed, hoping to get lucky. BleepingComputer also covered this news, and they posted a statement from Gen Digital spokesperson, which is the one I quoted, saying: "We have secured 925,000 inactive and active accounts that may have been targeted by credential-stuffing attacks."

Okay. So wait. First of all, 925,000 accounts. And, quote, "We have secured them." Does anyone know what that means? What does it mean, "We have secured 925,000 inactive and active accounts that may have been targeted by credential-stuffing attacks."

**Leo:** Maybe they reset the password. Could it be?

**Steve:** Oh, my god, they could not have done a password reset on a million.

**Leo:** A million, no.

**Steve:** I mean, it would be the end of life as we know it.

**Leo:** So they probably just sent an email out.

**Steve:** Oh, my god. Yeah, maybe that's, oh, we had to let everybody know that they might be hacked. So we were going to call that "securing the accounts." Oh, lord. Anyway, you know, I guess whatever they did, it's much better than if they had not secured them. So since we're not actually offered any information, Leo, to your point, we don't get any these days, we just get corporate PR speak, it's necessary to read between the lines.

So my guess would be that Gen Digital currently has 925,000 Norton LifeLock customers, or accounts, at least, some inactive. They wandered off. And due to completely absent web portal security and a lack of monitoring for some length of time, once bad guys realized that there was no security to stop them, a fleet of bots was programmed to assault Norton LifeLock's login page, guessing account credentials at high speed without limit. And as a result of this lack of security, that fleet of bots was able to successfully login and compromise the accounts of 6,450 Norton LifeLock users.

Gen Digital did admit that it had discovered that these intruders had compromised LifeLock accounts beginning on December 1st, 11 days before they're saying its systems, that is, Gen Digital's systems, finally detected a "large volume" of failed logins to customer accounts. Well, that large volume would have been going on for weeks; right? But somehow they didn't see that. The red flag finally went up on December 12th when LifeLock became aware of it and presumably brought this attack to a halt. So I suppose that's what they meant when they said that they had "secured" those 925,000 active and inactive accounts. They basically halted an ongoing login attack after 6,450 successful logins and full account compromise of those customers.

**Leo:** When I read this, I thought, oh, it's a credential-stuffing attack. I didn't realize that they were being able to brute force attack it. I just thought they were copying passwords from other breaches and trying them on LifeLock.

**Steve:** When they say that they, on the 12th, a large volume of failed login attempts...

**Leo:** But that could also be credential stuffing because you don't know who's reused passwords. So you might have a database of 10 million...

**Steve:** Oh, oh, I completely agree. By "brute force" I don't mean start at 0000000.

**Leo:** Oh, okay, yeah, yeah.

**Steve:** I just mean let's try all the passwords.

**Leo:** Let's see if this password that we have works.

**Steve:** Yeah.

**Leo:** Yeah, yeah.

**Steve:** Right, yeah. Let's ask Troy Hunt for his master list. We'll check them all.

**Leo:** That's right, yeah. So it's not, in a way, that would - I don't - I have no reason to defend them. But this is why you don't reuse passwords, because of these credential things.

**Steve:** It's absolutely why you don't. But also it took them at least two weeks before they saw - before something happened. So okay. So here's my theory. They're not saying how much earlier the attack was underway. So why 11 days from first successful compromise to first detection of an attack that had been ongoing for some time? If I had to guess, I would suggest that the bot fleet's attack was probably carefully throttled so as not to trip any alarms. And that after some successful undetected logins, the bot fleet's operators may have started creeping its attack rate upward slowly.

**Leo:** They got greedy.

**Steve:** Yeah, exactly, to see how much faster they could go. And remember that before they were shut down they'd successfully scored against 6,450 accounts. I mean, that's a lot of accounts. So things had been going well for the attacking fleet for like 11 days at least from the first known compromise. And truth be told, we don't even know that they actually ever were detected. We don't know that's what tripped the alarm and raised the red flag.

Given that the bots had been stomping around within 6,450 of LifeLock's user accounts, I would be surprised if some user out of 6,450 didn't notice that something was amiss and contact Norton to report suspicious account activity. So it may have just been, you know, the fact that they were tipped off by a victim, and they thought, oh, what? Maybe we ought to go over and look at that web server. And it's like, oh, my god. And then they, you know, did whatever they said they did to secure all the accounts.

Anyway, this does highlight another good reason for choosing an iteration count that takes the web browser a few seconds to obtain a go/no go login decision because it also serves as very good brute force protection against login attempts to your provider's portal; right? In order to log into LastPass.com, to use that example, some script has to be run on the browser in order to churn away at a guest password. I mean, that length of time has to be spent in order to create a token to hand to LastPass to say please log me in. So again, high iteration counts are just all-around protection from many types of problems.

Okay. One last thing, then we'll take our second break. Chrome has followed Microsoft and Firefox. Remember the certificate authority TrustCor? Last December we spent some time looking at the bizarre set of corporate shells and various shenanigans being played by that very shady certificate authority, TrustCor. And at the time it was difficult to understand why anyone would trust those clowns, based on the history that was revealed. Remember that one of the corporate users of their certificates was a deep packet forensics entity that was found to be selling their TLS middleboxes to agencies of the U.S. federal government. And there was also that individual and private company that was also affiliated whom no one had ever heard of before, who had for some reason received that huge block of previously unused U.S. Department of Defense IPv4 allocation. You know, there was something really fishy about this whole business.

Anyway, finally, in response to a long dialogue with a company representative that convinced no one, Mozilla and Microsoft both marked TrustCor's root certs as untrusted, thus immediately revoking trust from any certificates that had been signed by those certificate authority certs. We're talking about this again because, as I said, Google and Chrome have now followed suit by removing TrustCor's CA certs from Android and Chrome's root stores. So it's pretty much game over for that group. And good riddance. As I've said, you know, the responsibility - I've always been impressed that the browsers are so reticent and like really reluctant to pull trust from a CA. You've got to really be bad in order to have that happen.

**Leo:** It's going to break a lot of things. That's why they don't want to do it.

**Steve:** Oh, it's going to break everything that that CA ever signed.

**Leo:** Right. So they don't want the calls. It's expensive.

**Steve:** Right.

**Leo:** All those tech support calls are expensive. And we were talking last week, and I made a completely new, very good, long - and I used Password Haystacks to pad it out, long password. And I rekeyed it. And I turned the PBKDF2 iterations up to two million, and I just - I didn't need to do any of that, but I just - it makes me feel a little bit better. Now, the one thing I still have to do is go through all my passwords and update them because who knows how many were in my LastPass.

**Steve:** Boy, and that's one thing I heard from so many of our listeners is what a pain in the butt it is to change passwords.

**Leo:** Well, and as you said last week, there should be a way to do this. There should be an API. There should be something; right? But there isn't. There's no standard way to do it.

**Steve:** Okay. So Chromium is beginning to Rust. Google's announcement and blog posting last Thursday is titled "Supporting the Use of Rust in the Chromium Project." They wrote: "We are pleased to announce that, moving forward, the Chromium project is going to support the use of third-party Rust libraries from C++ in Chromium." That is, libraries called from C++. "To do so, we are now actively pursuing adding a production Rust tool chain to our build system. This will enable us to include Rust code in the Chrome binary within the next year. We're starting slow and setting clear expectations on what libraries we will consider once we're ready. Our goal in bringing Rust into Chromium is to provide a simpler and safer way to satisfy the Rule of Two."

And I actually skipped over the fact at the top of the show that that is today's podcast title, "The Rule of Two," which we'll be talking about here in a minute. So they said: "Our goal in bringing Rust into Chromium is to provide a simpler and safer way to satisfy the Rule of Two," whatever that is, "in order to speed up development - less code to write, less design docs, less security review - and improve the security, meaning increasing the number of lines of code without memory-safety bugs, decreasing the bug density of code, of Chrome." And they said: "And we believe that we can use third-party Rust libraries to work toward this goal."

And they finished: "Rust was developed by Mozilla specifically for use in writing a browser, so it's very fitting that Chromium would finally begin to rely on this technology, too. Thank you, Mozilla, for your huge contribution to the systems software industry. Rust has been an incredible proof that we should be able to expect a language to provide safety while also being performant." And god, I hate that word "performant."

**Leo:** I know.

**Steve:** It just seems - it's like, is it Apple who talks about the learnings?

**Leo:** No, Microsoft is learnings, yeah.

**Steve:** Oh, Microsoft is learnings.

**Leo:** The whole tech industry has its own vocabulary, and it's the business vocabulary.

**Steve:** Yeah. Anyway, everyone listening to this podcast has heard me lament that we're never going to get ahead of this beast of software flaws if we don't start doing things differently. What was that definition of insanity? Anyway, it's great news that the Chromium project is taking this step. It will be a slow and very evolutionary move to have an increasing percentage of the Chromium codebase written in Rust; but this is the way that effort, and this eventuality, gets started. You know, you've got to start somewhere.

And you may have noted, as I said, that Google's announcement mentioned this Rule of Two, which we'll be taking an in-depth look at here in a minute. But first we have another instance of BYOVD, Bring Your Own Vulnerable Driver. It was just in the news this past week. "CrowdStrike documented their observation and interception of an eCrime" - I hadn't seen that term before, now we have eCrime - "eCrime adversary known variously as Scattered Spider, Roasted 0ktapus, and UNC3944," for those who are not very imaginative.

"This leverages a combination of credential phishing and social engineering attacks to capture one-time password codes or to overwhelm their targets using that multifactor authentication notification fatigue" that we were talking about before, where they just finally say, okay, fine, I don't know why I'm being asked, but fine. And then, you know, that lets the bad guys in. Once the bad guys have obtained access, the adversary avoids using unique malware which might trip alarms. Instead, they favor the use of a wide range of legitimate remote management tools which allows them to maintain persistent access inside their victims' networks.

CrowdStrike's instrumentation detected a novel attempt by this adversary to deploy a malicious kernel driver through an old and very well-known eight-year-old vulnerability dating from 2015 which exists in the Intel Ethernet diagnostics driver for Windows. That is, a legitimate driver published in 2015 by Intel for performing Ethernet diagnostics. The file is "iqvw64.sys."

The distressing factor is that the technique of using known-vulnerable kernel drivers has been in use by adversaries for several years and has been enabled by a persistent gap in Windows security. And we've talked about this before. Starting with the 64-bit edition of Windows Vista, and ever since, Windows does not allow unsigned kernel-mode drivers to run by default. That was easy to do; right? It just shut down one path of exploitation. But clever attackers started bringing their own drivers that were signed, like legitimately signed, like Intel signing this Ethernet diagnostics driver. They would bring that along and cause Windows to install it, and then exploit the known vulnerabilities in this driver.

Okay. So what do we do about that? Well, in 2021, about two years ago, Microsoft stated that - this is Microsoft: "Increasingly, adversaries are leveraging legitimate drivers in the ecosystem and their security vulnerabilities to run malware," and that "drivers with confirmed security vulnerabilities will be blocked on Windows 10 devices in the ecosystem using Microsoft Defender for Endpoint Attack Surface Reduction (ASR) and Microsoft Windows Defender Application Control (WDAC) technologies to protect devices against exploits involving vulnerable drivers to gain access to the kernel."

Well, that was the plan. How did that work out? As we discussed some time ago, multiple security researchers through the years, the past two years have repeatedly and loudly noted pounding on Microsoft that this was all apparently just feel-good nonsense spewed by Microsoft, and that the issue continues to persist as Microsoft continually fails to actually block vulnerable drivers by default.

The crux of the problem appears to be that any such proactive blocking requires proaction from Microsoft. And any fair weighting of the evidence, many examples of which we've looked at during the past two or three years here, would conclude that Microsoft has long since abandoned any commitment to true proactive security. They no longer find most of their own problems in Windows. They increasingly rely upon the good graces of outside security researchers, and even then they drag their feet over implementing the required updates, which have been handed to them by the security community.

Okay. That said, though, being fair to Microsoft, there is a flipside to this. We know that the last thing Microsoft ever wants to do is deliberately break anything. They have enough trouble with inadvertently breaking things, let alone doing so deliberately. So proactively blacklisting anything, especially something like a network driver that could potentially cut a machine off from its own network, well, that's the last thing that Microsoft would choose to do, assuming that a choice was being made in the first place. But as I was thinking about this, it occurred to me that this is something that a third party could do on behalf of users who subscribe.

I'm bringing this back up again because this was not supposed to happen. The last time we spoke about this last year, the presumption was, based upon clear statements made by Microsoft at the time, that this had all been a big oversight mistake for several years, and that all that was going to be better now. But this latest news is from last week, and this was not actually fixed.

Okay. So there's that. But the bigger point I want to make is that this all needs to be made proactive. Somebody needs to be proactive. The need for Microsoft to be proactively blocking known vulnerable drivers seems like something we are never going to be able to get from Microsoft. For the foreseeable future, at least, this appears to no longer be in their DNA. That's just not who they are any longer. And we've seen cycles; right? They've, like, swung from one side to the other. Maybe they'll swing back. We can hope. But that's not where they are today.

But, you know, it occurred to me that this presents a huge and significant opportunity for some third-party security company. Solve this problem. Be proactive in a critical area where Microsoft refuses to be. It's a simple thing to do, like get the list of all the previously known vulnerable drivers. Create an app that looks to see if any of them are in use. If so, tell your subscriber to update the driver to a new one so that it can then be blocked from malicious future use. Anyway, my sense is there's a big opportunity here for someone to make themselves a lot of money by closing what is a persistent and gaping hole in Windows security. I hope somebody will do it.

Okay. A piece of closing-the-loop feedback. My Twitter feed was so overwhelmed with feedback from last week's call for feedback that I was unable to reply to tweets as I usually try to do. So to everyone who tweeted, please accept my thanks for the feedback and my apology that I probably didn't reply. I know that replying is not required, but courtesy and chivalry are not dead here. As many of my regular Twitter correspondents know, I often reply when I can.

But one public tweet caught my eye for its cleverness, which I wanted to share. Somebody tweeting as @mammalalien tweeted: "@SGgrc Sometimes it is hard to picture just how many more random strings there are than English words. So try this." Sort of a thought experiment. "How often have you ever seen real words coincidentally appear in randomly generated passwords?" He said: "I noticed it today for I think the second time ever."

So anyway, I think that's a very clever and worthwhile observation. It helps us to truly appreciate how much less entropy exists in non-random text, which we recognize as words in our language, such that it almost never appears by chance. I don't think I've ever noticed any significant word - maybe "is" or something - in truly random strings of characters. So anyway, I just thought that was a cool observation.

Okay. The Rule of Two. I'm going to get into this, then we'll take our third break, and then we will continue. The Rule of Two. In their posting about the adoption of Rust by Chrome, Google mentioned something they called "The Rule of Two." In Google's official Chromium Docs they explain what this so-called "Rule of Two" is. They said: "When you write code to parse, evaluate, or otherwise handle untrustworthy inputs from the Internet, which," they said, "is almost everything we do in a web browser, we like to follow a simple rule to make sure it's safe enough to do so."

They said: "The Rule of Two is pick no more than any two of untrustworthy inputs, unsafe implementation language, and high privilege." And just to highlight the point on the page, under the Chromium Master Docs where they show this, they give us a Venn diagram where one circle is "Code which processes untrustworthy inputs" is one circle. "Code written in an unsafe language," and they show C and C++ as examples, that's another Venn diagram circle. And then the third one is "Code which runs with no sandbox," for example, in the main browser process. And so they've got all three circles with overlapping regions. And in the center, where all three circles overlap, they've got in big red all caps "DOOM! Don't do this." That is, do not operate in that place where all three of these things are true: untrustworthy, unsafe language, unsandboxed execution.

Now, of course this is reminiscent of that great old saying that in a project, you can choose any two, but only two of the following three outcomes: good, fast, or cheap. You can't have all three. You'll need to sacrifice the thing that's least important to you among them. You can have something good and fast, but then it won't be cheap to get it. Or you can have something fast and cheap, but then it won't also be good. Or you can have something good and cheap, but then you won't be able to get it fast. So similarly, for Google's Rule of Two, untrustworthy inputs, unsafe implementation language, and high privilege.

**Leo:** Can we just not have any of the above? I would like to, I mean, why do we have to pick two?

**Steve:** Well, yeah, wouldn't that be nice.

**Leo:** None of the above.

**Steve:** On the other hand, our browser is going out in the Wild West of the Internet; right? I mean, you might go to a site.ru, and god help you. Who knows what your browser is going to pick up? Okay. So let's take a look at these. Leo, let's tell us about our last sponsor, then we can dig into...

**Leo:** The Venn diagrams.

**Steve:** The Venn diagram.

**Leo:** Seems to me if you're going in the Wild West, you don't want any of those, let alone two of them. Okay. But I guess it's hard to do in real-world environments.

**Steve:** We do have people who have a separate machine running only a browser, and it's like it's off their network, and that's what they use. It's like isolated in order to be...

**Leo:** A lot of unsophisticated people, because I've recommended on the radio show for years get a Chromebook for your banking and just do that. And a lot of people like that idea. Like, good, because then I can use my Windows machine as I wish, but I'll know that I'm pretty safe when I'm doing my banking.

**Steve:** Right.

**Leo:** You know, I don't think that's an unusual idea.

**Steve:** Not a bad idea. And the Chromebook has the wash button, too.

**Leo:** The power wash, yeah. And it of course checks for signed firmware so you can't - it's harder to do bad things to it. All right. Back to Steve and two things.

**Steve:** Yes. So we're going to walk through this piece where Google is explaining the Rule of Two. And this was actually written as advice and guidance for would-be Chrome developers. So Google explains: "When code that handles untrustworthy inputs at high privilege has bugs" - so untrustworthy inputs at high privilege and bugs - "the resulting vulnerabilities are typically of critical or high severity." They said: "We'd love to reduce the severity of such bugs by reducing the amount of damage they can do by lowering their privilege, avoiding the various types of memory corruption bugs by using a safe language, or reducing the likelihood that the input is malicious in the first place by asserting the trustworthiness of the source."

So they said: "For the purposes of this document, our main concern is reducing, and hopefully, ultimately eliminating bugs that arise due to memory unsafety." They said: "A recent study by Matt Miller from Microsoft Security states that 'around 70% of the vulnerabilities addressed through a security update each year continue to be memory-safety issues.' A trip through Chromium's bug tracker will show many, many vulnerabilities whose root cause is memory unsafety." They said: "As of March 2019, only about five out of 130 public critical-severity bugs are not obviously due to memory corruption." Only five out of 130 were not memory corruption.

They said: "Security engineers in general, very much including the Chrome Security Team, would like to advance the state of engineering to where memory-safety issues are much more rare. Then we could focus more attention on the application-semantic vulnerabilities. That would be a big improvement."

Okay. So it's clear that the historic use of C and C++ has been the source of a great many past security vulnerabilities despite the coders of those languages doing the very best jobs they can. There is no date on this document, but it feels a few years old since they were citing Matt Miller's well-known research, which we've cited before here, from March of 2019. From what was written above, it's also clear that they were wishing and hoping then for the move that Google announced just this past week with the formal incorporation of Rust as a first-class Chromium implementation language.

Okay. So let's flesh out each of these three factors in the context of browser implementation. First, untrustworthy inputs. They explain that untrustworthy inputs are inputs that, A, have non-trivial grammars, meaning a complex language to figure out; and/or come from untrustworthy sources. So Google explains: "If there were an input type so simple that it was straightforward to write a memory-safe handler for it, we wouldn't need to worry much about where it came from for the purposes of memory safety because we'd be sure we could handle it. We would still need to treat the input as untrustworthy after parsing it, of course." But they're saying, you know, if the grammar is simple, parsing is not a problem.

So they said: "Unfortunately, it is very rare to find a grammar trivial enough that we can trust ourselves to parse it successfully or fail safely. Therefore, we do need to concern ourselves with the provenance of such inputs." So in other words, the stuff a browser confronts is both typically a highly complex language such as HTML, CSS, or JavaScript, all of which have unfortunately evolved to be incredibly complex. And we don't know about their source, such as some third-party advertising server serving ads submitted by unknown entities. None of this can be trusted. If the language used to express these things was simple, it would be easier to trust that our parsing of the language would be enough protection for us. But unfortunately, these are not simple languages.

Okay. So Google said: "Any arbitrary peer on the Internet is an untrustworthy source, unless we get some evidence of its trustworthiness," they said, "which includes at least a strong assertion of the source's identity." They said: "When we can know with certainty that an input is coming from the same source as the application itself, for example, Google in the case of Chrome, or Mozilla in the case of Firefox, and that the transport is integrity-protected over HTTPS, then it can be acceptable to parse even complex inputs from that source." They said: "It's still ideal, where feasible, to reduce our degree of trust in the source, such as by parsing the input in a sandbox." And we'll be talking about that in a second.

Okay. So that was untrustworthy inputs. What about implementation language? Google explains: "Unsafe implementation languages are languages that lack memory safety, including at least C, C++, and assembly language. Memory-safe languages include Go, Rust, Python, Java, JavaScript, Kotlin, and Swift." And then they said: "Note that the safe subsets of these languages are safe by design, but of course implementation quality is a different story."

Okay. So what about unsafe code in safe languages, which there are often provisions for? Google said: "Some memory-safe languages provide a backdoor to unsafety, such as the unsafe keyword in Rust. This functions as a separate unsafe language subset inside the memory-safe language. The presence of unsafe code does not negate the memory-safety properties of the memory-safe language around it as a whole, but how unsafe code is used is critical. Poor use of an unsafe language subset is not meaningfully different from any other unsafe implementation language."

So they said: "In order for a library with unsafe code to be safe for the purposes of the Rule of Two, all unsafe usage must be able to be reviewed and verified by humans with simple local reasoning. To achieve this, we expect all unsafe usage to be three things: Small, so the minimal possible amount of code to perform the required task; second, encapsulated. All access to the unsafe code is through a safe and understood API. And third, documented. All preconditions of an unsafe block, meaning a block of code, for example, a call to an unsafe function, are spelled out in comments, along with explanations of how they're satisfied." So in other words, they said, where a safe language such as Rust provides facilities for breaking out of safety in order to address some need, that region of unsafety must be small, contained, completely understood, and well documented.

So they continue: "Because unsafe code reaches outside the normal expectations of a memory-safe language, it must follow strict rules to avoid undefined behavior and memory-safety violations, and these are not always easy to verify. A careful review by one or more experts in the unsafe language subset is required. It should be safe to use any code in a memory-safe language in a high-privilege context." Okay. So there's two of the rules.

"It should be safe to use any code in a memory-safe language in a high-privilege context. As such, the requirements on a memory-safe language implementation are higher. All code in a memory-safe language must be capable of satisfying the Rule of Two in a high-privilege context, including any unsafe code that it encapsulates in order to be used or admitted anywhere in this project." Okay, so that was interesting. I mean, they're like saying we're following these rules. If you can't satisfy the Rule of Two, the code is not coming into Chromium.

Okay. So finally, code privilege. That's the third item from which we can only have two. So Google explains: "High privilege is a relative term. The very highest privilege programs are the computer's firmware, the bootloader, the kernel, any hypervisor or virtual machine monitor, and so on." They said: "Below that" - you know, below those very top ones - "are processes that run as an OS-level account representing a person. This includes the Chrome browser process." They said: "We consider such processes to have a high privilege. After all, they can do anything the person can do, with any and all of the person's valuable data and accounts.

"Processes with slightly reduced privilege include, as of March 2019, the GPU process and, hopefully soon, the network process." They said: "These are still pretty high-privilege processes. We are always looking for ways to reduce their privilege without breaking them. Low-privilege processes include sandboxed utility processes and renderer processes with site isolation, which is very good, or origin isolation, which is even better."

Okay. So Google then talks about two topics that we've discussed through the years as we've observed over and over how difficult they appear to get right, parsing and deserialization. Remember that deserializing is essentially an interpretation job, and interpretation is notoriously difficult to get correct because it appears to be nearly impossible for the coder of the interpreter, who inherently expects the input to be sane, to adequately handle inputs that are malicious.

So Google says: "Turning a stream of bytes into a structured object" - that's the deserialization - "into a structured object is hard to do correctly and safely. For example, turning a stream of bytes into a sequence of Unicode code points, and from there into an HTML Document Object Model tree, with all of its elements, attributes, and metadata, is very error prone. The same is true of QUIC packets, video frames, and so on. Whenever the code branches on the byte values it's processing, the risk increases that an attacker can influence control flow and exploit bugs in the implementation. Although we are all human, and mistakes are always possible, a function that does not branch on input values has a better chance of being free of vulnerabilities." And then they say: "Consider an arithmetic function, such as SHA-256, for example."

And I thought that was a really interesting observation. We made SHA-256 branch-free so that differing code paths would not leak timing information and would not leave sniffable hints in our processor's branch prediction history. But a side effect of that also increased the algorithm's robustness against deliberate code path manipulation because there is none. So anyway, what surprised me a bit is that the Chromium security team, as I said, is extremely literal about the application of this Rule of Two. They're not joking around. They actually apply the rule when evaluating new submissions.

And they wrote some advice to those who would submit code to the Chromium project. They said: "Chrome Security Team will generally not approve landing a new feature that involves all three of untrustworthy inputs, unsafe language, and high privilege. To solve this problem, you need to get rid of at least one of those three things. Here are some ways to do that."

Okay. Privilege reduction, obviously one of the three things. They said: "Also known as sandboxing, privilege reduction means running the code in a process that has had some or many of its privileges revoked. When appropriate, try to handle the inputs in a renderer process that is isolated to the same site as the inputs came from. Take care to validate the parsed processed inputs in the browser, since only the browser can trust itself to validate and act on the meaning of an object. Or you can launch a sandboxed utility process to handle the data, and return a well-formed response back to the caller in an Inter-Process Communications message."

So, okay. These ideas are structural means for creating an arm's-length, essentially a client-server relationship where a low-privilege worker process does the unsafe work and simply returns the results to the higher privilege client. That way, if something does go sideways, there's containment within the process that cannot do much with its malicious freedom because it doesn't actually have much freedom to be malicious with. As for verifying the trustworthiness of a source, they say that if the developer can be sure that the input comes from a trustworthy source - so not overtly attempting to be malicious - it can be okay to parse and evaluate it at high privilege in an unsafe language, even though that seems scary. In this instance, they say, a "trustworthy source" means that Chromium can cryptographically prove that the data comes from a business entity that can be or is trusted, for example, in the case of Chrome, coming from one of the Alphabet companies.

Google then talks about ways to make code safer to execute under the title of "Normalization." Writing to would-be Chromium coders, they explain. They said: "You can 'defang'" - literally, their term - "'defang a potentially malicious input by transforming it into a normal or minimal form, usually by first transforming it into a format with a simpler grammar." They said: "We say that all data, file, and on-the-wire formats are defined by a grammar, even if that grammar is implicit or only partially specified, as is so often the case." They said: "For example, a data format with a particularly simple grammar is" - and they have an internal data structure, SkPixmap, which basically is a simple image pixel map.

They said: "This 'grammar' is represented by the private data fields: a region of raw pixel data, the size of that region, and simple metadata which directs how to interpret the pixels." They said: "Unfortunately, it's rare to find such a simple grammar for input formats. For example, consider the PNG image format, which is complex and whose C implementation has suffered from memory corruption bugs in the past. An attacker could craft a malicious PNG to trigger such a bug. But if you first transform the image into a format that doesn't have PNG's complexity - in a low-privilege process, of course - the malicious nature of the PNG should be eliminated" - basically defanged, as they said, or purified - "and then be safe for parsing at a higher privilege level.

"Even if the attacker manages to compromise the low-privilege process with a malicious PNG, the high-privilege process will only parse the compromised process's output with a simple, plausibly safe parser. If that parse is successful, the higher privilege process can then optionally further transform it into normalized, minimal form, such as to save space. Otherwise, the parse can fail safely, without memory corruption."

So they said: "For example, it should be safe enough to convert a PNG into this SkBitmap in a sandboxed process, and then send the SkBitmap to a higher privileged process via an interprocess communication. Although there may be bugs in the interprocess communication message deserialization code and/or this SkBitmap handling code," they said, "we consider that safe enough."

So I think that the interesting message here for those of us who are not writing code for a browser is, first of all, to be thankful that we're not.

**Leo:** Yeah, no kidding.

**Steve:** That is, boy, that is not an easy job; and, secondly, to more deeply appreciate just how truly hostile this territory is. It is a true battlefield. On this podcast I've often noted that the browser is that part of our systems that we blindly thrust out into the world and hope that it doesn't return encrusted with any plagues. When this environment is coupled with the insane complexity of today's browsers, it's truly a miracle that they protect us as well as they do.

And all of this is up against the crushing backdrop of the imperative for performance. Google notes that, they said: "We have to accept the risk of memory-safety bugs in deserialization because C++'s high performance is crucial in such a throughput- and latency-sensitive area. If we could change this code to be both in a safer language and still have such high performance, that would be ideal. But that's unlikely to happen soon." So this was written several years ago. Rust was noted earlier, so at least it was on their radar. I wonder whether that might be the right compromise, if it would be possible to move this most risky aspect which they have had to keep up in C++ because in this - this is the pinch point in the performance pipeline for getting the page on the screen. Could they actually reimplement this in a memory-safe language? I guess we're going to see.

But for now, it's clear that the reason Task Manager shows us 30 processes spawned when Chrome or Edge launch is for containment. Low-privilege processes are being created and are given the more dangerous and time-critical, performance-critical tasks to perform. They're time-critical, so they're written in an unsafe language to go as fast as they possibly can; and they're performance critical. But they're created in a low-privilege separate process, even though there's some overhead talking back and forth between processes because, if something explodes, you want it to be in a low-privilege container. They cannot be written in a safe but slow language. They need C++. So they're held at arm's length and with their interprocess communications strictly limited to receiving a task to perform and returning the result of that work, like rendering processes, which are notoriously error prone.

These notes for Chromium developers talk a bit more, kind of wistfully, about the idea of using safe languages. Google writes at the end, they said: "Where possible, it's great to use a memory-safe language. Of the currently approved set of implementation languages in Chromium, the most likely candidates are Java on Android only, Swift on iOS only, and JavaScript or WebAssembly," they said, "although we don't currently use them in high-privilege processes like the browser," okay, so meaning that JavaScript and WebAssembly are currently only being used in browser web pages and in browser extensions. They said: "One can imagine Kotlin on Android, too, although it is not currently used in Chromium. Some of us on the Security Team aspire to get more of Chromium in safer languages, and you may be able to help with our experiments." And of course we know from their announcement last week, Rust has made that move.

Okay. So at this point no mention of Rust's adoption, and this was a couple years ago. But we now know that's changing. This interesting discussion and guidance for would-be Chromium developers concludes by noting that all of this is aspirational and that, unfortunately, even this does not reflect Chromium's current state. Under the final heading of "Existing Code That Violates the Rule," they write: "We still have code that violates this rule. For example, Chrome's Omnibox" - the single URL and search box up at the top of the UI - "still parses JSON in the browser process. Additionally, the networking process on Windows is at present unsandboxed by default, though there is ongoing work to change that default."

Okay. So we're seeing an evolution across our industry. Web browsers have become so capable that they're now able to host full applications. That has enabled the relocation of those applications from our local desktop to the cloud, and a redefinition of the customer from an owner to a tenant. I, for one, hate this change. But those of us who feel this way are dying off, and we're clearly irrelevant anyway. But neither are my hands completely clean since I'm composing these show notes in Google Docs, which is a stunning example of how well this new system can work.

But the gating requirement for any of this to work and for any of this future to unfold is for our web browsers to survive on the front line of an astonishingly hostile Internet. No one who's been following this podcast for the past few years could have any doubt of the open hostility that today's web browsers face every time they suck down another page loaded with unknown code of unknown provenance containing pointers to other pages of code with the need to go get, load, and run that code, too. I mean, it just makes you shudder.

So I'm very glad that Google's Security Team is thinking about the problems they're facing, that they take mitigations such as this Rule of Two as seriously as they do, and that they're finally beginning to migrate to the use of safer languages, which, I'll note, was made possible by Mozilla pioneering this wonderful Rust development language.

**Leo:** Is that, do you think, the only one they could use? Or it's the best? Or have you even thought about that at all? I mean, I know you use assembly.

**Steve:** It may be Mozilla's influence.

**Leo:** Yeah.

**Steve:** Even though they seem like competitors, there is a lot of cross-pollination...

**Leo:** Sure, of course there is, yeah.

**Steve:** ...between Chromium and Firefox.

**Leo:** Yeah, yeah. Very cool, yeah.

**Steve:** And everything we hear about Rust says that it is a serious implementation language. I mean, like a systems-level systems implementation language.

**Leo:** Yeah, yeah. It's a really interesting language, yeah. Steve Gibson, once again, has done it. He's put together two hours of fascinating conversation about the things we care about the most. Thank you, Steve. You did it again.

**Steve:** Well, we also found out that LastPass turned out not to be willing to help us.

**Leo:** Disappointing. Disappointing.

**Steve:** I was hoping, I hoped for several hours that that was the case.

**Leo:** Oh, well. You'll find Steve at GRC.com. That's his website, the Gibson Research Corporation. There's some good stuff there. Of course SpinRite, the world's finest mass storage maintenance and recovery tool. It's available now in version 6.0, proven bug-free over the last 18 years, but soon to be 6.1, also bug free for another 18 years. You'll find that in process. But if you buy 6.0 now, you will get 6.1 as soon as it comes out, which should be fairly soon.

**Steve:** I did release Alpha 9.

**Leo:** Nice.

**Steve:** Which had a huge slew of new features. And so we're in the process of getting that tested. We're getting closer.

**Leo:** Good. Good. That's a good reason to get it right now. While you're there you can also get a copy of this show. Steve has two copies we don't have, a 16Kb audio version for the bandwidth impaired, and transcriptions, handcrafted transcriptions by Elaine Farris, so you can read along as you listen or use them for search. That's all at GRC.com. Lots of other great stuff there. So browse around.

There is a feedback form, GRC.com/feedback, but he also is on the Twitter, as you heard him mention, @SGgrc, which means you can also - his DMs are open. You also can DM him there. But he has been swamped lately, so don't expect a personal reply. He'll do his best.

**Steve:** I like to when I can.

**Leo:** I know. I know you do. That's why you're not joining us on Mastodon. It's all right, I understand.