Flame On!

Description: This week, after catching up with a large amount of the week's news, Leo and Steve carefully examine two major new discoveries about the Windows Flame worm.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-357.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-357-lq.mp3

SHOW TEASE: It's time for Security Now!. Steve Gibson's here. Boy, he's got a lot of security news. There was a big Microsoft update, a new Apple update of Java, and Oracle, too. But the big story is Flame, two amazing revelations that might give us some idea about where Flame actually came from. Steve Gibson, next, on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 357, recorded June 13th, 2012: Flame On.

It's time for Security Now!, the show that protects you online. And here's the Protector in Chief, Mr. Steve Gibson of GRC.com, the Gibson Research Corporation, creator of SpinRite, world's best hard drive maintenance and recovery utility. It looks like you're wearing a plain black shirt, Steve. Surely there's something written - "Born to Code." I knew it. I knew it. Is that a Think Geek T-shirt?

Steve Gibson: No, I had it custom made.

Leo: "Born to Code."

Steve: Yeah, that one and the one that says "Future Centenarian," both of those.

Leo: I do like that.

Steve: Yeah, I've got some great comments. I was walking down in Laguna Beach about a couple of weeks ago with Jenny, and some guy said, "Oh, I love your T-shirt."

Leo: "Future Centenarian?"

Steve: Most people just sort of stumble over it. It's like, what the hell does that say?

Leo: How old are you now?

Steve: Got a lot of syllables.

Leo: "Future Centenarian." I'm making one of them. Can we sell that in our TWiT Store? Or something like, "I'm Ketogenic, Are You?" Something like that.

Steve: That'd be good.

Leo: Got your ketones rising.

Steve: Do you have a TWiT store?

Leo: We do. We just opened one.

Steve: With the hoodies and the fezzes and the...

Leo: Yeah, TWiT - not fezzes yet. Unfortunately, some of the fine TWiT merchandise like this mug and the fezzes are made by third parties.

Steve: Fine TWiT merchandise.

Leo: And so for fezzes you have to go to Fez-o-rama. It'd be nice if we could get it. But we have a Spreadshirt store, twit.spreadshirt.com, that you can get hoodies with the logo and the T-shirts and stuff. We could have a couple of fun ones like "Born to Code," "Future Centenarian," "I'm in Ketosis."

Steve: "I'm in Ketosis, Why Aren't You?"

Leo: Yeah, I like that. We've got to spread the word.

Steve: Yeah, we've actually - I've had so much great feedback. People, I mean, our listeners are losing multiple tens of pounds, I mean, decades.

Leo: You know, we blew it. If we'd been smart - but, see, we're not really marketing anything. But if we'd been smart, we would have had, like, a page that says how many pounds, and a thermometer, and a giant scale with how much our audience has lost, things like that. That would be awesome. Or gained. We could put a "gained." I doubt anybody's gained, but we could put a "gained" just to be fair. Today we have - the title of the show is "Flame On!" In fact, I'm going to add an exclamation mark.

Steve: I've got one here, yeah.

Leo: I think you're talking about Flamer.

Steve: We are. Two major revelations have come out this week, as I was sure would happen. And as I've said the last couple weeks, we will be learning more about this over time. This just - it takes time to peel this apart, to reverse engineer it, for these guys to look inside. They don't have source code. They've got the executable code that they have captured. And so there's just no substitute. Somewhere, certainly with Kaspersky in Russia and scattered around the globe, there are engineers who are burning the midnight oil, figuring out what this thing does, how it works, and where it came from. And we have two, as I was writing the notes for this, Leo, this morning, I was getting goose bumps because of some of what has been discovered this week. It was also a huge week for security stuff, so we've got a whole bunch of stuff to talk about at the top of the show. And then two major, bone-chilling...

Leo: Oh, boy.

Steve: ...really interesting revelations about Flame.

Leo: Holy cow.

Steve: Yeah.

Leo: Well, it's a jam-packed episode today of Security Now!. I'll tell you what. Hey, I forgot to ask you, did you see "Prometheus"?

Steve: I did not, only because I want to wait a little bit longer for the theaters to cool off. So I'm going to go Monday, and we can talk about it next Wednesday.

Leo: Good. I won't say anything. I did see it. There was not - it was not a crowd. There was no crowd.

Steve: Interesting.

Leo: Yeah, that surprised me. I was ready. I bought tickets two hours early and went 45 minutes early, sat in the front row. The theater didn't even fill up. On a Saturday night.

Steve: Whoa, on a Saturday night?

Leo: I don't know if that's Petaluma or what.

Steve: Did not see it. I'm looking forward to it.

Leo: I know you were really excited about it. So, Patch Tuesday. Did you see that? Playing in theaters worldwide.

Steve: I saw that coming.

Leo: Yes.

Steve: So Microsoft Patch Tuesday was yesterday. And the most-used operating system in Iran and the Middle East got patches for 26 known security flaws.

Leo: Say again? 26?

Steve: 26 known security flaws. Brian Krebs refers to them as "patch bundles," which I like because the way Microsoft does them is there'll be, like, they'll fix one file, some DLL, and it had three problems in it. So it's three vulnerabilities, but one patch.

Leo: Got it.

Steve: So there were in this case seven patch bundles covering 26 known flaws. Half of them, 13, were in IE. So those were important. Microsoft announced and fixed, all at once, a critical flaw in their RDP protocol, the Remote Desktop Protocol. And also they've got a problem that they acknowledged which they have not fixed, but I just tweeted it because it's critical. So, let's see. I don't see a link in front of me. Oh, yeah, there's an MS Fix it, one of the quick fix-it buttons, and that's support.microsoft.com/kb/2719615.

So the story here is - again, this is not fixed with yesterday's Patch Tuesday because Google found this being exploited in the wild, so it was a zero-day exploit. It is under active exploitation now. It involves Microsoft's XML Core Services, all versions, 3.0, 4.0, and 5.0, across all Windows platforms and Office 2003 and 2007. Again, I love - actually I turned this into an acronym. Brian Krebs refers to it as "Browse and Get Owned." I thought, ooh, BAGO.

Leo: BAGO.

Steve: Browse And Get Owned.

Leo: Holy cow.

Steve: And this is worse, though, because Office is involved. This is also being used in targeted attacks simply by mailing people Office documents. So it's an uninitialized memory vulnerability. Microsoft wrote: "The vulnerability exists when MSXML attempts to access an object in memory that has not been initialized, which may corrupt memory in such a way that an attacker could execute arbitrary code in the context of the logged-on user." Which is Microsoft's traditional backpedaling way of saying BAGO: Browse And Get Owned. It's not like it could and it might and if the moon is in the correct phase. No, this happens.

Leo: BAGO.

Steve: So BAGO. You browse somewhere, and that's it.

Leo: That's terrible.

Steve: It's very bad. So thus I tweeted it. You can go to just Twitter.com/SGgrc. You'll see a link that I sent out this morning to this fix. Everybody who's listening who's using Windows should do this because this will fix it. In the meantime, I'm sure Microsoft will nail this by next month. But this is not something you want hanging out there. Google detected that it was being used and locked it down.

Leo: That's interesting.

Steve: So now doubt they've got some - their web crawlers crawled some infected pages and said, what's this? So it's important. Also on Microsoft, Microsoft's been very busy because there's been a lot of upshot from what's happened with the Flame worm. I mean, they got really caught off guard. And last week I said, eh, it wasn't clear to me whether there might have been some sort of plausible deniability here. It just sort of seemed, from what we knew at the time, a little suspicious to me that there could be this crossover of certificate signing.

Well, I no longer believe that, due to what I will be talking about later in this podcast today, because we now understand much more about how the fraudulent certificate was created that was used to sign this malicious Flame code. And this is why I got chills. So this wasn't Microsoft's sort of wink, wink, nod, nod, to the NSA. Microsoft I'm sure is very unhappy.

So they have updated immediately a bunch of infrastructure things under Windows Vista and 7. They are updating their OS's certificate update tool to automate the process that

used to be manual-only of disallowing certificates, that is, putting certificates in the untrusted status list. They said: "The new certificate update tool will rely on a 'Disallowed Certificate Trust List' maintained by Microsoft. The tool will check the list daily, moving certificates found on the list into the 'untrusted' store. In the past, moving certificates to untrusted status required manually updating them."

So this is Microsoft being proactive, saying we're going to have every Windows Vista and 7 box on the planet check in daily so that we're able to, if this ever happens again, we're able to be much more proactive...

Leo: Interesting.

Steve: ...in instantly distrusting those certificates. So that's a cool, nice step forward. They also said that they are giving advance warning - and this is something I'm surprised but very pleased about - they're giving advance warning of an update to how Windows manages certificates that will blanket-invalidate certificates that don't have adequate security. They said certificates with RSA encryption keys shorter than 1024 bits will automatically be marked invalid. Once this key length update is released...

Leo: Wow.

Steve: ...all such certificates will be treated as invalid, even if they are currently valid and signed by a trusted certificate authority.

Leo: That's one way to do it.

Steve: So that's big. There are 768 bits, it's not really strong, but it's pretty strong. 512, not so much.

Leo: But this isn't so much about the strength of the bits as an opportunity to kind of invalidate a bunch of certificates.

Steve: Yeah. And...

Leo: Yeah? Older ones.

Steve: Exactly. So really, since certificates are always expiring, we know that they have a year or two or three life - although it's possible for individuals to create longer living certificates if they want. I'm using certs for my VPN that I used really ridiculously long bit counts, but I gave them a long life, so I'm not, like, traveling and then having my cert expire on me, which would be a problem. So this is a good thing that they're doing.

Still, it's surprising because this is the kind of thing that will catch some people unawares who aren't paying attention, and suddenly something will break that was working before they did this. But it's all - this is the nature of staying ahead of the moving target of

security, which we'll be talking about here later because this MD5, which is increasingly insecure as problems have been found with it, and it was MD5 that Microsoft only stopped using last year, and that was the vulnerability that allowed the certs to get made for Flame.

But also, still in reacting to Flame, Microsoft is updating Windows Update. They said they have a hardened Windows Update infrastructure so that the Windows Update client, that is, that part running in our machines, will only now trust files signed by a new certificate that is used solely to protect updates to the Windows Update client, which, again, this is them reacting to the fact that the Windows Update client had been unnecessarily permissive. As we know from what I said last week, it was essentially accepting code that was signed with a sort of a generic - it was in the trust chain with heredity from Microsoft Terminal Services. And there was no reason for Windows Update client to be so permissive.

Well, they've locked that down. They're saying, nope, we're just - there's no reason it was that way. Essentially, this is Microsoft sort of maybe overly trusting the whole public key infrastructure, the PKI system, just saying, well, our root is going to be secure, so no one's going to get bad certificates. Well, someone did. And had this been in place before, this would have excluded the particular solution that was found that allowed Flame to get these certs. Furthermore - get this, Leo.

Leo: Oh, boy.

Steve: They are strengthening their communications channel. No more proxying of SSL traffic. They were allowing an enterprise or, for example, educational institutions we've talked about often that have their own SSL certs and proxy traffic on behalf of the Intranet, the idea being that a Windows client running inside a corporate environment or an educational institution would, in initiating an SSL or HTTPS connection, would have a cert from a proxy on the network border, which it would negotiate with. That would then decrypt at that stage, and then that proxy would establish a connection, in this case to Windows Update. Microsoft is saying we're not going to let that happen any longer. We need to have a non-inspected passthrough. So they're saying that they're going to no longer allow proxying of SSL traffic. Enterprises and educational institutions who connect through a network proxy will need to create passthrough exceptions so that Windows Update traffic is tunneled without inspection. And of course again they're doing this to strengthen it. They're saying you're connecting with us, you don't get to look inside of our tunnel. Like, okay. Because of course the...

Leo: What side effects is that going have? What if people in business probably rely on that; right?

Steve: Yeah, although businesses are normally pulling their updates separately already. That is, the clients are not directly connecting to Microsoft. They're using an enterprise update facility so that IT is able to look at everything and make sure it doesn't break things within the enterprise. And then an enterprise deploys these updates into their Intranet, inside of their border. So but for, like, schools, schools are not doing that, and this will break Windows Update until the schools add this proxying exception. So again, these are serious infrastructure changes. But as I said, all this indicates to me, Microsoft is not happy. They are not happy that their system got abused in this fashion. One of the...

Leo: Yeah, I wouldn't be. I mean, that's really about as bad as you can get.

Steve: Yeah, one of the further indications that this was no wink and a nod.

Leo: Right, right, they didn't agree to this one.

Steve: Unh-unh.

Leo: And that makes sense. That would be such a mistake to agree to that.

Steve: It would, it would.

Leo: You can see the horror.

Steve: So we have an amazing flaw that came to light in MySQL, the My Sequel Server. HD Moore, who's our famous originator of the Metasploit framework, explained in his posting, he said: "On Saturday afternoon Sergei Golubchik posted to the oss-sec mailing list about a recently patched security flaw" - and it has a CVE number of 2012, of course this year, dash 2122 - "in the MySQL and MariaDB database servers. This flaw was rooted in an" - this is a really interesting mistake, which I think our listeners are going to get a kick out of. You will, too, Leo. "This flaw was rooted in an assumption that the memcmp() function would always return a value within the range..."

Leo: Oh, always a mistake.

Steve: Uh-huh.

Leo: "Always" is a bad word.

Steve: Now, HD wrote "-127 to 127," and he says "signed character." But actually assembly language programmers know it's -128 to 127. You've got to be careful with those rounding errors. But still, so what he's saying is that the mem compare function would always return a value within that range. "On some platforms and with certain optimizations enabled, this routine can return values outside of this range, eventually causing the code that compares a hashed password to sometimes return true, even when the wrong password is specified."

Leo: [Laughing]

Steve: Uh-huh. "Since the authentication protocol generates a different hash each time this comparison is done, there is a one-in-256 chance that ANY," he has in all caps,

"password would be accepted for authentication."

Leo: Any all-caps password, regardless, would be accepted. Geez Louise.

Steve: So, he says: "In short, if you try to authenticate to a MySQL server affected by this flaw, there is a chance it will accept your password even if the wrong one was supplied. The following one-liner in bash" - and he has a little one-line bash script - "will provide access to an affected MySQL server as the root user account, without actually knowing the password."

Leo: Wow. That's not good.

Steve: "This evening Jonathan Cran, CTO of Pwn Express and Metasploit contributor" - yes, it's the Pwnie - actually, I'm sorry, it's Pwnie, P-w-n-i-e. The "Pwnie Express and Metasploit contributor committed a threaded brute-force module that abuses the [they're calling it an] authentication bypass flaw to automatically dump the MySQL password database." Okay, so get this. It exploits the flaw to dump the database.

Leo: Now you got everything.

Steve: "This ensures that, even after..."

Leo: Oh, they can't fix it.

Steve: Right. "Even after the authentication bypass vulnerability has been fixed, you should still be able to access the database remotely using the cracked password hashes obtained before the fix was applied. Pulling from resources of a personal side project of mine," says HD, "I was able to derive some statistics about the real-world impact of this vulnerability. This project managed to find and gather the initial handshake for approximately 1.74 million MySQL servers across the Internet at large."

Leo: Oh, horrendous.

Steve: "This statistic only includes MySQL instances that were on hosts publicly exposed to the Internet and not bound to localhost." So these are, I mean, again, the idea that any clown would have their SQL Server answering random queries from the Internet is like, okay, just lock them up now.

Anyway, so let's dig into this a little deeper because this is interesting. The memcmp() function is a standard C function. I just put "memcmp" into Google. The first link was www.cplusplus.com/reference/clibrary/cstring/memcmp, which is the standard C string memory comparison function. Under description it says: "Compare two blocks of memory: Compares the num bytes of the block of memory pointed to by ptr1 to the first num bytes pointed to by ptr2, returning zero if they all match or a value different from zero representing which is greater if they do not." It takes three parameters: ptr1, which

is a pointer to the first block of memory; ptr2, pointer to the second block of memory; and num is the number of bytes to be compared.

And it says under return value: "Returns an integral value indicating the relationship between the content of the memory blocks. A zero value indicates that the contents of both memory blocks are equal. A value greater than zero indicates that the first byte that does not match in both memory blocks has a greater value in ptr1 than in ptr2, as if evaluated as an unsigned char value," an unsigned character value. "A value less than zero indicates the opposite."

Now, if you were to compare two blocks of memory - in this case we're comparing two hashes. We're comparing the stored password hash to a hash made when the user entered their guess. So we've got two blocks of memory. The normal way this would operate is you would do a byte-by-byte march along both of these blocks of memory of length num, and essentially you'd subtract one byte from the other. And if you get zero, that means they're identical. So you go to the next pair of bytes, the next byte in each of these two memory blocks. You subtract one from the other. If you get zero, you continue. So you keep going until either you don't get zero as a result of a subtraction of the two bytes, or you finish the num number of comparisons.

So what you'll always end up with is a one-byte value that is either zero, if you got all the way through, meaning all of the bytes in these two blocks were identical and the hashes match, or you'll end up with this one byte that won't be zero, and the direction in which it's not zero is which one or the other was greater. So, and that would be useful, for example, if you were comparing alphabetic strings, and you wanted to do an alphabetic sort. You'd like to know if the two strings are different and also which one is alphabetically larger or smaller than the other.

But when he said, depending upon what optimizations were applied, now, the byte-by-byte comparison I just suggested is robust and reliable. But there would be a great tendency to speed it up. And in the list of vulnerable OSes, the 64-bit OSes seem to be, like, disproportionally represented. The great temptation would be to compare eight bytes at once. That is, if num is greater than eight, then, for example, if it's nine, then you know - or if it's eight or greater. So then the temptation would be to do a 64-bit, that is, an 8-byte comparison; instead of doing eight single-byte comparisons, do one 8-byte comparison.

The problem is that, if you do that, and notice that hashes are often multiples of eight bytes long, so they're going to finish, if you compare 64 bits at a time, they're going to finish at an even 64-bit boundary. It would be very possible for seven of those bytes to be completely wrong, but the final bytes to be the same. Which means the least significant byte of the subtraction could be zero, even though the rest of them are not. So it is a classic coding error to not check the entire value's zero-ness, but to only check the least significant byte. And because the function is cast to return a signed byte, all of the more significant data gets discarded. And this thing just returns the last byte. So the chances are one in 256 that you're going to end up with a zero final byte, even if the other seven bytes are non-zero, and you wouldn't know it. The function would ignore those more significant bytes, only return the least significant one, and...

Leo: Whoops.

Steve: Oops. So what we have is...

Leo: And this, by the way, this library routine is all over the place. I mean, this is...

Steve: Oh, it's a fundamental...

Leo: ...fundamental.

Steve: Exactly. It's very, very key. So here's an instance where a security vulnerability was found, but this probably represents a bad bug in code all over the place, which is using this optimization in order to jump through the buffers eight bytes at a time, if they're not looking at the entire size of the final 8-byte by 8-byte subtraction for comparison.

Leo: And most people are going to use memcmp() because it's a library routine. It's stood the test of time.

Steve: Uh-huh, and then it got broken.

Leo: Whoops.

Steve: This is the way, I mean, this is why computers have bugs. It's like little tiny things like this.

Leo: You hate to see a bug in a library routine, especially a very commonly used library routine, because that means we don't know how widespread this could be.

Steve: And I should just say I'm just conjecturing. I wanted to take our listeners through a how-this-could-have-happened. I didn't look at the code. I don't...

Leo: Right, no, that makes sense, yup. And this is in C++. So anything that's written in C++ potentially...

Steve: Or just C. I think has the C has the same...

Leo: Actually, you're right. I don't know if the same bug exists, though. It's the C++ library; right?

Steve: Ah, good point. Okay. So we did talk last week about LinkedIn. And the, what is it, 65 million, or, no, 6.5 million.

Leo: Yeah, one tenth of their users.

Steve: Had passwords leaked. I was very pleased that LastPass got themselves involved because I like LastPass so much. They've done such a great job. Our friend Joe Siegrist blogged, he said: "LinkedIn was hacked, confirmed by LinkedIn on 6/6/2012. LinkedIn has updated their blog, indicating that there was a breach, and several LastPass staff members who used unique passwords for LinkedIn only" - and you could imagine anyone working for LastPass is going to have one of those wacky LastPass-derived, fully pseudorandomized passwords.

They said: "Several LastPass staff members who used unique passwords for LinkedIn only, as well as numerous individuals not associated with LastPass, have confirmed that LinkedIn's database has indeed been hacked." Meaning, as we talked about last week, they put in their password into a hash, hashed it, and found it in the database. So Joe wrote: "If you have a LinkedIn account, we strongly suggest that you immediately, A, change your LinkedIn password; and check if you have re-used your LinkedIn password on any other websites and, if so, change those passwords, too."

And the question was, "Was MY LinkedIn password hacked?" So they created a page, very nice, simple page, LastPass.com/linkedin, where they offer the ability to check your password, your LinkedIn password against the master database that was leaked. So they've made this simple to do. And of course I trust LastPass not to do anything wrong.

Now, you don't have to trust them because the first thing you should do is change your LinkedIn password, then test your old one there. But since we've talked, eHarmony also lost 1.5 million unsalted MD5 passwords. So LastPass.com/eharmony will do the same thing there. And Last.fm also updated their blog indicating there was a breach and have confirmed that they are forcing password changes because the entire password database has been floating around the Internet for years, it now turns out.

Leo: Oh. What? Oh, geez.

Steve: So LastPass.com/lastfm. So there's three pages at LastPass (linkedin, eharmony, and lastfm), any page at which you can submit the appropriate password, and LastPass will hash it on your browser - they're not getting anything but the hash - and then check it against each of those three leaked databases which they have collected and are maintaining for us. So thank you, Joe and company, for that nice service. And I wanted to let our listeners know there's somewhere trustworthy they can go to have their passwords checked. But of course always change yours first and give your old one to the site, just for prudence sake.

And last week we were talking about, in the LinkedIn context, we were talking about the need for salt. And I got a couple tweets that made me feel like people believed that I believed that salting was all that was necessary. And although we've covered this before, I thought it's worth reminding people that that's really not all that's necessary for anyone implementing a password database. There's now a very well-known, well-understood best practices.

We know we need to salt, but salting is not enough because hardware exists which, if the salt is known, it's still possible, even though you wouldn't have a rainbow table - a rainbow table is basically a precomputation table for a hash, which could be and has

been, for the popular hashes MD5 and SHA-1, unsalted, has been created to allow a password to immediately be looked up from its hash. So given a hash, you could find a password, not necessarily THE password, but it doesn't have to be THE password, just a password, that will hash to the same thing, which would allow a bad guy to log in and impersonate you. If you salt, that doesn't work. It immediately busts any precomputation table.

But the problem is precomputation tables are a decade old. Now we've got walls of PlayStation 3s. We've got field-programmable gate arrays. We've got the NSA building secret facilities in Utah. We've got all kinds of things happening that are specifically designed to crack hashes, whether they're salted or not. And so, if the salt is known, then you can still apply - although you can't precompute, you can now compute on the fly because, sadly, these hashes were designed for efficiency. MD5 and SHA-1 were designed back in the early '90s when computers were two decades slower than they are now, and so they were designed for speed when that was a priority.

So what that means is that they are parallelizable and pipelineable, meaning that they could be implemented to be extremely fast in hardware. You can either have tens of thousands of them running in parallel at once, simultaneously testing 10,000 different passwords; or, if it's pipelineable, you could have 10,000 passwords moving through a pipeline coming out the other end to be tested. So this makes them very fast.

So consequently, you want to use something that we talked about before, a PBKDF function, a Password-Based Key Derivation. WPA, that was designed more recently and much more strongly, famously does use a Password-Based Key Derivation Function, the idea being that you use iterations of a salted hash where you take the output of it, you put it back in, you add the salt, you hash it again, take the output, add the salt, hash it again, and you do that some number of times. WPA uses 4,096 iterations. iOS 3 used 2,000 for its security, and iOS v4 increased that by a factor of five to 10,000. So this is all part of what you need to do.

Now, I would say that best practice, we talked about not storing, in a best practice situation, not storing the salt with the database. Well, what you really want to do to make the strongest system you can is you custom salt every account. So when somebody is creating their password, you generate a pseudorandom salt for that user. You also have what I would call a sequestered system-wide salt which is not stored with the database. So the per user salt, which does need to be stored with the database, that makes each user's hash custom to them. The sequestered system-wide salt, which is not stored with the database, means that if only your database is compromised, they still don't get that.

But then, once you've got these two salts, a per user salt and a per system salt, then you apply Password-Based Key Derivation, whatever, 10,000. And the idea there is that you are slowing down the process of processing this hash in such a fashion that even guys with really fast GPUs and hardwares are slowed. So that's best practice for hash storing today.

Now, the next generation, I was glad to see that Colin Percival is the guy behind Tarsnap. And I've talked about Tarsnap. This is the Linux-based cloud storage system that is extremely secure. He's done some of the work on the so-called "memory hard functions." The other problem with MD5 and SHA-1, in having been designed for speed, is that they don't require much memory. And if you don't require much memory, you can make them much more parallelizable because essentially the number of these that the NSA can create is a function of the die size. And so if you just build yourself an MD5 engine or an SHA-1 engine, the smaller it is, the more of them they can put on a chip, and the faster

their overall processing is going to be.

So what you want is an extremely memory-hungry algorithm. That is, an algorithm that cannot be run in parallel. It's parallel hostile. It cannot be pipelined. It's pipeline hostile. And it uses - it has to have a huge amount of memory. What that means is that it would just be impractical to create chips. The chip would have to be big because every single instance would have to have a big plot of land allocated to ROM or RAM just by its very nature, and probably RAM, and the algorithm is going to use it in such a fashion that the whole chunk of RAM is being churned in some fashion. It's going to be very slow. There's just no way to make that better.

So we're learning a lot from all of these attacks about how to create really robust anti-password-cracking technology. Of course, it doesn't help when major sites just use an unsalted MD5 hash because that's just no longer secure against these kinds of attacks. So we know how to do it well. I'm glad that we're seeing high-profile breaches like this because it's got to bring management's attention to, like, asking the question of their programmers - hey, we're not using an unsalted MD5? - even if the guy in the suit doesn't know what that means. The programmers can say, uh, we'll get back to you on that. And then find out whether they are or not.

Leo: Interesting.

Steve: I did talk last week with some excitement about how Microsoft's IE 10 was going to be getting the Do Not Track header enabled by default?

Leo: They changed their mind. No.

Steve: They didn't last a week, Leo.

Leo: Actually, they didn't change their mind. Their mind was changed for them.

Steve: Correct. The industry, the advertising industry had a meltdown and laid it on Microsoft's doorstep. It turns out, though, that, I mean, and this is - we need agreement from all the players. So what the advertising industry reminded Microsoft of was the fact that the Do Not Track spec - which I really like because it's got some great guys behind it. Peter Eckersley from the EFF, the Electronic Frontier Foundation; Tom Lowenthal from Mozilla, and Jonathan Mayer, who we've spoken of many times, at Stanford, are all privacy advocates. And they're the editors of the W3C's working paper for the standardization of DNT. This is going to get W3C standardization.

However, the way this has been written, and this is a compromise I can live with, explicit consent is required. So in this W3C standard, it will say that an ordinary user agent, meaning a browser, must not send a tracking preference signal, that is, any tracking preference signal, without a user's explicit consent. Example: The user agent's privacy preferences pane includes controls for configuring the tracking preference signal. Or, on first run, the user agent prompts the user to configure the tracking preference signal. But that says it just can't be on by default.

And so essentially what was said was that, if Microsoft did have it on by default in IE 10,

that would inherently violate the standard and free the advertisers from paying attention to it because it was a nonstandard header as Microsoft had implemented it. So this gives you a sense for the games that are being played. On the other hand, we're talking about having the FTC enforcing this once this becomes a W3C standard. So this is all good news. This is what we predicted when this first surfaced, probably more than a year ago. I was saying yes, yes, yes, that's - this is a good thing. And it has taken a long time. These things do. But we're making some progress.

And Leo, when you turned your Mac on before we began recording the show?

Leo: Oh, I got a few little updates there.

Steve: Yup. We did have, across the board, from Oracle and Apple, an update to Java. I found mine this morning when I unblanked my screen. I'm still using v6 of Java, so I received Update No. 33. People who have moved to 7, both are parallel tracks at this point from Oracle. Version 7, being newer, has its update No. 5. So those are available.

And in Oracle's fixes, Brian Krebs noted that Oracle patched 14 security flaws, only 11 of which were patched by Apple. So he didn't know, and I don't know why. Either Apple didn't think they were important, or Apple's implementation of Java. They may have been a Windows-only thing, so they don't affect Apple at all. Apple is deemphasizing Java, no longer ships it. But for those systems that have it, and many do because many need it, they are maintaining their version, and they probably are stuck doing that pretty much forever.

And lastly, Mozilla is heading toward opt-in activation of browser plug-ins, which I think is just all good news.

Leo: What is that? Isn't it already? I mean, don't you have to install them?

Steve: No. Many of these things are in the browser, or they'll be - I'm sorry, I wasn't clear. Per site, that is, site-specific permissions.

Leo: Ah.

Steve: So essentially moving some of the technology that we have known and loved from NoScript into the native Mozilla platform. It's in beta in v14 of Firefox, and they're targeting it for release in v16. And what they're saying is that so many exploits are being caused by obsolete plug-ins, that they are, in very much the same way that Chrome is, they're becoming much more proactive about - they've got their browsers secured, largely. And now they're saying, okay, well, we've got to secure the things that run in the browser.

So this would be - okay. So these would be site-specific, enforced by the browser. So you would tell it when you want to run plug-ins by site, and the browser, Firefox, would remember that. I think that's just all good. And as I'm reading this, I'm thinking, okay, I mean, this makes sense because there isn't just one type of person on the Internet. I know that many people will be using IE, and IE will just run everything. But there are many people who care more about security. And I'm just comforted by having to enable

scripting. I'll go to a page, and it'll be blank. And I'll think, okay, I do want to see what this page has. So for me it's a quick right click. And I often just enable it temporarily, unless it's something that I know I'm coming back to. And then the page loads, and everything's fine.

So I like the control. I, knock on wood, have never had anything crawl into my machine. I don't surf promiscuously all over the Internet, so I'm not a typical target for this kind of stuff. But still, I'm glad to have my guard up. And I fully respect that not everyone is as cautious as I, and probably many of the podcast listeners are. There are solutions for them, too. So we're not just in a one-size-fits-all world anymore, relative to Internet security.

And lastly, this is only tangentially related to security. I have it under my Miscellaneous category. But I just thought I'd mention it, an interesting facility coming to Facebook, called Pipe, from a small German team. This uses something that Adobe has in Adobe AIR called Adobe Real Time Media Flow Protocol. And this is a very simple, encrypted, peer-to-peer file sharing for Facebook users. And it's, like, feature sparse, which is actually why I like it.

If two people are, like, on the phone, or talking, or both on Facebook, it's a tiny app that you can add to Facebook, called Pipe, and it simply allows you to drop a file into the pipe, and it comes out the other end. And that's all it does. It doesn't use Pipe servers because it is truly peer-to-peer. The Real Time Media Flow Protocol solves the NAT traversal stuff, and it's peer-to-peer in the same way that Skype is. And it's powerfully encrypted. I think it's AES-128, if I remember. And it's not Dropbox. It doesn't do storage in the cloud or synchronization among multiple machines. It's just a pipe. And so you can - and you don't need to friend other people to use it, as other systems do. It's just a pipe.

So if in any situation someone's on Facebook and they want to - oh, and it's got a size of a gig limit because it uses the browser's cache. So it's whatever you drop in is going - it's a browser-to-browser technology. So you're limited. You just can't send massive things. But my goodness, a gig. So, I mean, that's going to take a while to transfer anyway. But as a super, super simple, clean means of moving files between Facebook users, I think it's going to be popular. I love the fact that it's that simple to use. You just drop it in, and it comes out the other end. It's a pipe.

Leo: Very nice. Yahoo! has another product called Yahoo! Pipes. But that's not the same.

Steve: Not Facebook.

Leo: Oh, okay, this is Facebook Pipes.

Steve: Facebook, yes, it's just Facebook to Facebook users. And so no client to install. It's a little Facebook app, very small, very lightweight. And it just allows, I mean, it's like your mom could use it because it's simple.

Leo: Interesting.

Steve: And I got a nice note from someone named Ray, who maybe didn't give me his last name because he didn't pay for SpinRite, apparently, but it saved his life. And he said maybe literally. So I guess I'll say, Ray, maybe you'll buy SpinRite at some point. In the meantime, I thank you for sharing the story, this lifesaving story.

He said, "So I decided to reinstall Windows on my computer for the sake of speeding it up." Which is a good idea. I do that every few years. "But of course I had all kinds of things to be backed up. So instead of just burning a few DVDs or borrowing an external hard drive, I decided to transfer it through my network onto my other computer, a computer I've had for about six years. Everything transfers over, and I start my reinstall of Windows on the first machine. When the time came to transfer it all back to my main computer, something went wrong. All but about five files did not transfer over." So, wow. All but about five files did not transfer over. The first time I read this, this morning, I thought he said...

Leo: All but five.

Steve: Only five didn't transfer. So, yikes.

Leo: So he only got five.

Steve: Yeah. "Just a bunch of unable-to-access-drive errors. At this point I'm thinking, oh, great, I guess I'm going to lose some stuff."

Leo: Yeah, guess so.

Steve: "So I tried to restart the computer." And this must be the second computer that received this archive. "And nothing at all. The hard drive cannot be read to boot. Right about this time is when my thought process went from, 'Oh, great' to 'Oh Lord, please protect me. My wife is going to kill me.' It had occurred to me that all of my wedding pictures were part of those files.

"So lucky me, I knew someone who owned SpinRite. I called him immediately and got his copy and ran it. The drive was in VERY" - he has in caps - "bad shape. So it took approximately two weeks of chugging away, but SpinRite finally recovered EVERYTHING" - he has in all caps - "on the hard drive. I lost some music that never successfully made the initial transfer, so that wasn't SpinRite's fault. I'm going to appreciate SpinRite for the rest of my life because of this. Also, the next day I purchased Carbonite so this can't happen again."

Leo: Good man.

Steve: "Thank you, Steve, for creating the greatest product in the history of spinning disks." And thank you, Ray. I'm glad SpinRite worked for you. Thank your friend who hopefully was an owner. And maybe somebody you'll become one, too. In the meantime, all of our listeners know that SpinRite can get the job done.

Leo: Sure does. That's nice. We're going to talk about Flame. Flame On! Two big revelations.

Steve: Goose bump raising, yes.

Leo: Oh, boy. I love this stuff. Somebody's going to write the novel about Flame. I can't wait. Flame. Flame On. So Flame, or Flamer - actually has a lot of different names, we'll call it Flame for the time being - is impressive.

Steve: That seems to be the - well, yes. It's beyond impressive. As we know from the last couple weeks that we've been discussing it, it is 20MB. It's 10 times the size of Stuxnet and Duqu. And what we believed last week, what I said from what we believed, turns out to be wrong. We have some revelations...

Leo: What? Oh.

Steve: ...this week. Aleks [Gostev] at Kaspersky blogged about a discovery which they have made since we last talked about it. And he said: "Back to Stuxnet: The Missing Link." He said...

Leo: Ooh. Now, we know Stuxnet was developed by the U.S. and Israel.

Steve: Exactly. And so Aleks wrote: "Two weeks ago, when we announced the discovery of the Flame malware, we said that we saw no strong similarity between its code and programming style with that of [what they're calling] the Tilded platform which Stuxnet and Duqu are based on." So this Tilded platform is the common derivative from Stuxnet and Duqu. "Flame and Tilded are completely different projects based on different architectures and each with their own distinct characteristics. For instance, Flame never uses system drivers, while Stuxnet and Duqu's main method of loading modules for execution is via a kernel driver.

"But it turns out we were wrong. Wrong, in that we believed Flame and Stuxnet were two unrelated projects. But it turns out we were wrong. Wrong, in that we believed Flame and Stuxnet were two unrelated projects. Our research unearthed some previously unknown facts that completely transform the current view of how Stuxnet was created and its link with Flame."

So to go back a little bit, we covered this extensively when it was happening. There were three main variants of Stuxnet. There was one that was built, the first one, in June 2009. And then there were two others in March and April of 2010. And the middle one, the March 2010, evidenced the most penetrations, and it was because it was so prevalent, the one that was first detected in June of 2010. Can you believe it's been two years, Leo? That's bizarre. Where did time go?

Leo: Yeah.

Steve: Okay. So Aleks says: "Despite the fact that Stuxnet has been the subject of" - and this is so cool. "Despite the fact that Stuxnet has been the subject of in-depth analysis by numerous companies and experts, and lots has been written about its structure, for some reason the mysterious 'resource 207' from 2009 went largely unnoticed." So that's something called "resource 207" that was in Stuxnet from 2009, that is, the June 2009 Stuxnet, the first one. "But it turns out that this is the missing link between Flame and Stuxnet, two otherwise seemingly completely unrelated projects.

"In October of 2010" - okay, so October of 2010, which was after the discovery of Stuxnet, so June 2010 was when the March 2010 version was discovered. "In October of 2010 our automatic system received a sample from the wild." That is, of something. They didn't know what. "It analyzed the file thoroughly and classified it as a new Stuxnet variant, Worm.Win32.Stuxnet.s. With Stuxnet being such a big thing, we looked at the sample to see what it was. Sadly, it didn't look like Stuxnet at all. It was quite different. So we decided to rename it to Tocy.a and thought, 'Silly automatic systems.'" It misclassified it.

"When Flame was discovered in 2012, we started looking for older samples [of Flame] that we might have received [and not recognized]. Between samples that looked almost identical to Flame, we found Tocy.a. Going through the sample processing system logs," back from October 2010, "we noticed it was originally classified as Stuxnet. We thought, how was it possible? Why did the system think that this Flame sample," which we now knew was Flame, "was related to Stuxnet? Checking the logs, we discovered that the Tocy.a," which they renamed, which their system originally identified as Stuxnet, "we discovered that the Tocy.a, an early module of Flame, was actually similar to 'resource 207' from Stuxnet. It was actually so similar that it made our automatic system classify" that at the time unknown Flame worm "as Stuxnet. As it turned out, Tocy.a was similar to Stuxnet alone and to no other sample from our collection. This is how we discovered the incredible link between Flame and Stuxnet."

So this resource 207 is an encrypted DLL containing what they refer to as a "PE." PE is the acronym for Portable Executable, which is the Windows format for executable files, which is 351K in size. Resource 207 from Stuxnet is a Flame plug-in. And they said actually a "proto-Flame plug-in." So Stuxnet's resources actually contain a Flame platform component, meaning that a week ago we didn't think Flame and Stuxnet were from the same source, from the same people, from the same team, in any way related. Now we know they are.

Once they believed that, and because they're were becoming increasingly familiar with Flame and already knew Stuxnet so well, they began looking for similarities which they hadn't been searching for, and they began to find them. They refer to "mutex names." A mutex is a sort of a fundamental - it's what's called a "synchronization object" in coding, stands for mutual exclusion. The idea is that different threads running in either the same process, that is, the same program, or different processes may need to coordinate their access. Say that they shared a database, and you'd need some way of them not both reading a record from the database, making different changes, and then writing them back because the last one to write it back would overwrite the changes the other one made. So you use something like a mutex or a semaphore to synchronize the execution of these threads.

Well, when the threads are in the same process, they're able just to share the handle of the mutex because they would both be able to know it. But when the threads are in separate processes, because you've got interprocess isolation, which is a key for the security of our systems, there's no way for two different threads to know the handle of something that they need to share unless you give it a name. So what's done in Windows

is that mutexes are named. And essentially the first thread to create the mutex or open it uses the name. And if it doesn't already exist, it's created in the system, and that thread is given a handle that it can use to access it. And then later, if some other process creates the same mutex with the same name, it gets a return code saying, oh, that already exists. Here's your handle to it. And so it's a way, by using a common, a name that they both know, it's a way of synchronizing these things within the system that you wouldn't normally have access to due to process separation. Well, both Flame and Stuxnet use improbably named mutexes: TH_POOL_SHD_PQOMGMN_ and then a %d, so that's probably a short - a printf for...

Leo: For date, yeah.

Steve: ...a date or a decimal number. And then SYNCMTX, as an example. Well, there's no way that's a coincidence that they both had that. And then some similar things. They both share the same string decryption algorithm which is unique to them. They both mangle class names in the same way. They both have similar naming conventions. In Stuxnet there is a temp file created named dat3A.tmp, which is placed somewhere. And when Stuxnet infects a system, it creates that file so that, when other instances of Stuxnet start up, they check to see if that file exists. And, if so, they know Stuxnet is already here.

Well, Flame - okay, so that was, again, dat3A.tmp. Flame uses dat3C.tmp. So what they've found is by looking at closely at this code, they now understand they have common roots. And they know that there was a source code level relationship between these groups or teams, or maybe it's just one group, because this secret resource 207 that just was part of Stuxnet, remember that it was there in the first release of Stuxnet that was found in June '09.

But the functionality of resource 207 that exists in Flame today, it was migrated into other modules of Stuxnet in the March and April 2010 versions. To do that, you pretty much need the source code. So you would take the source code, and you'd say, okay, we're not going to have it in resource 207. We're going to move those functions around into other places. So it's very clear now that there was - we don't know the details of common authorship. But we believed a week ago there was no relationship. Now we know there absolutely, unequivocally was. Probably the same contractor.

Leo: Same author.

Steve: The same author. The same group. In fact, there was some note in Aleks's blog where he - because I was thinking about myself and coding style. They have the same coding style. And that's something...

Leo: Yea, yeah. And that's very distinctive. It really is unique.

Steve: Yes, I was thinking about that with myself. I know, if I looked at my own assembly code, like a reverse, a disassembly of my binaries, I could just see this is me, this is the way I do things. And for this kind of low-level stuff, it had to be written in assembler because the compiler would tend to fuzz that. The compiler, things come out looking all kind of generic from a compiler. So you could identifier the compiler that

produced the code, but not the author who produced the code because that level of translation, it would sort of blur...

Leo: It kind of obfuscates it, yeah.

Steve: Yeah, it blurs it. But if I looked at my disassembly, it's like, oh, I wrote that. And I know there's no question. And so that's what they're seeing. They're seeing a style of coding at the assembly level that at least parts of this had to be created in where it's like, oh, they used the same, just the same fingerprint, same coding style. And there's an infinite number of those at that level. So, yes, these are all the same group who put this together, which is amazing. Now, there's more.

Leo: There's more.

Steve: To pull off this signed certificate attack, what we now understand is this wasn't a matter of simply getting a certificate, like having it issued, because that wouldn't work. What had to happen was something we talked about years ago. We covered a brilliant cryptographer named Marc Stevens, who was involved - and you'll remember this, Leo, because I do - in creating a fraudulent certificate authority for the RapidSSL authority. And you may remember they used 200 PlayStation 3s...

Leo: Right, right.

Steve: ...a wall of these things. And what they did was - the need was to create a new certificate, a different certificate with a different common name. The common name is the thing that you're protecting, essentially, like GRC.com is the common name for our SSL certificates. So I prove to VeriSign that I am GRC.com. I provide them with my public key, which they sign to assert that they have verified that I'm GRC.com, and that's then my certificate. So spoofing a certificate means changing the common name. Well, these things are all fingerprinted, the certificates are fingerprinted using message digests, which is what MD of MD5 stands for. And as we said earlier, Microsoft was using MD5 until last year, despite the fact that it's been chipped away at over the years. I mean, I think it was in '07, so five years go, that you and I talked about, we did a podcast on cracking MD5.

Leo: Right, I remember that.

Steve: Yeah, Marc's wall of 200 PS3s, all using their high-powered GPUs. Now, the reason that was necessary was that we're talking about what's called a "chosen prefix attack" on a message digest, the idea being that, as the digest goes along processing a file that it's creating a fingerprint for, it's evolving a state. It maintains a bunch of memory and has a bunch of static constants. And you're taking new data from the file and munging it in a different way so that all of the history of the file is mixed in with the new data and the constants in sort of an evolving fashion. So the idea is that, if you're incredibly clever, you can design some changes as a prefix such that parts that you don't control of the file will still end up giving you the result you want.

And so the idea is that you want to change part of the certificate in your way, yet still have it be signed by someone you trust, even though they never signed it, meaning that - and remember that, when I say the word "signature," a signing means that they have used their private key to encrypt the result of the hash, and so you verify it by using their public key. Well, that means, since you never get their private key, that means you've got to make the hash come out the same. So if the hash comes out the same, they signed the other hash for the valid certificate. You've managed to create your own certificate with the same hash, so their signature still applies.

So the problem is that these certificates are serialized, and they're time stamped. And so, if our listeners who remember this podcast will remember, the challenge was they had to submit a certificate request at a precise instant in order to get a preknown timestamp and a known serial number. They probably did it in the wee hours of the morning in the time zone...

Leo: When there's less traffic, yeah.

Steve: ...when there's less traffic. So the serial numbers would be advancing less slowly. So they said, okay, at this point in time the serial number is probably going to be this when the certificate is issued. That'll be at exactly this instant. They then used their PS3 wall to crunch ahead and figure out what prefix they needed in order to synthesize their certificate signing request, the CSR file, which they then submitted at the precise instant in order to get the authority to sign it. So that was what Marc did in '07. What Marc realized - and he was involved in this when it became clear that an MD5 collision attack was employed. He said, hey, what did they do? How was this used?

Okay. So what Marc said of this is: "Most interestingly, the results" - oh, and he used his own proprietary forensic tools to reverse-engineer the certificate that was used by Flame to authenticate its code. And he said, "Most interestingly, the results [of our analysis] have shown that NOT our published chosen-prefix collision attack was used, but an entirely new and unknown variant. This has led to our conclusion that the design of Flame is partly based on world-class cryptanalysis."

Leo: Isn't that something.

Steve: "Further research will be conducted to reconstruct the entire chosen-prefix collision attack [which was newly] devised for Flame." And Matthew Green, a professor specializing in cryptography in the computer science department of Johns Hopkins University, said: "It's not a garden-variety" - he says, "It's not a garden-variety collision attack," as if there is such a thing. I mean, what I just described is hardly garden variety. But at this level, the world's preeminent cryptographers are saying this is "not a garden-variety collision attack or just an implementation of any previous MD5 collision papers, which would be difficult enough. There were mathematicians doing new science to make Flame possible."

Leo: Isn't that amazing.

Steve: [Laughing] Oh, Leo.

Leo: Wow. Wow.

Steve: Oh, yeah.

Leo: That's just mind-boggling.

Steve: Yeah.

Leo: I mean, so world-class mathematicians working on this thing. Am I right?

Steve: This is NSA, yeah, this is NSA. This is not some contractors. This is not outside. I mean, this is, I mean, maybe Israel. But, I mean, this is...

Leo: Sounds like the NSA.

Steve: It really does, yeah.

Leo: Golly.

Steve: Oh, I got goose bumps again.

Leo: That's pretty cool.

Steve: And based on their analysis, they're saying that this was probably $200,000 worth of just pure compute time, just computation time. That's what it would cost to do the computation. And the same sort of collision process would have been required. They believe, due to reasons of the specifics, that the window would have been about a hundred times smaller for this versus what was done in '07. So Marc was saying this was orders, two orders, two decimal orders of magnitude more difficult, which it would have increased the amount, the computational complexity associated with synthesizing this cert in order to make it happen. I mean, these guys are just, like, they're stunned that this was done without them. This was not done in the academic community. This was done secretly in a cyberwar setting [laughing].

Leo: Very interesting.

Steve: Yup.

Leo: Well, Steve Gibson, you have once again blown my mind. Flame flames on. So

it looks pretty much like it was, well, since Stuxnet was, I think we now can safely say for sure, an Israeli/U.S. joint effort.

Steve: Well, yeah. Remember that...

Leo: Part of Operation Olympic Games.

Steve: Yep, Olympic Games. And I've been watching the political side. And of course the White House is not happy that this information leaked. On one hand, some people are saying, well, doesn't this make Barack look like he's tough on cyberwar? And it's like, yeah, but it shouldn't have leaked. And reportedly the President is not happy that this story got written. But...

Leo: Well, sorry. I mean, come on.

Steve: It's true.

Leo: People are paying a lot of attention to this. And good luck keeping it a secret.

Steve: It is interesting, Leo, that there is a level, I mean, this kind of level, this goes way beyond script kiddies and Metasploit turnkey exploit packages. This is world-class espionage. And, boy, can you imagine the chagrin of the people who designed it when this got discovered? It's like, oh.

Leo: Well, the big mistake was that it leaked out. It was only supposed to be used in constrained circumstances. And it's leaked out. And that's really the big mistake. If you want to be upset about something, be upset about that. But the truth is, I think that's a valuable lesson that every security researcher knows very well. It's very hard to contain these things, much harder than a bio weapon.

Steve: Yeah. It was used in a targeted fashion, thus the concentration of its being found to a much higher degree in Iran and in the Middle East than elsewhere. So it wasn't just, I mean, Stuxnet was found in people's air conditioners in their homes. So Flame was at least more tightly targeted than that. So that's good.

Leo: Steve Gibson is the best. You can find his work online, GRC.com. That's where SpinRite is, world's best hard drive maintenance utility - Dude, if you use it, buy it - and a lot of free stuff that you don't have to buy, and a lot of research and all sorts of things at GRC.com. Not to mention show notes, 16Kb audio versions, and even transcriptions of each and every one of these programs, all 357 episodes at GRC.com. Now, if you want the video or the higher quality audio, we've got that at TWiT.tv. And you can always subscribe. Is the 16Kb version a podcast? Did we ever make it like a downloadable podcast?

Steve: No.

Leo: Let me know if there's a demand for that, folks, and we can turn that into a subscribable show. Never even thought about it.

Steve: I haven't even checked my stats for quite a while. That's part of my revamping that I'm working on. And do not forget, please, GRC.com/feedback. Next week's episode is a Q&A, and I love to hear what questions our listeners have. We'll address them.

Leo: Good. Steve's Twitter is @SGgrc. And we do this show every Wednesday, 11:00 a.m. Pacific, 2:00 p.m. Eastern, so you can always watch live. We like it when you do, and we can see the chatroom and all that. That's 1800 UTC on TWiT.tv. Thanks, Steve. We'll see you next Wednesday. Or, oh, programming note, three weeks from now is the Fourth of July. So we will let you know when we're going to - we're not going to record on the Fourth of July. The whole place is shut down for fireworks. Except I think OMG Chat is going to do a Minecraft Marathon on the Fourth of July. He's going to launch his new Minecraft show then. So stay tuned for that. But no Security Now!. And we'll let you know what the new schedule is going to be.

Steve: Great.

Leo: Thanks, Steve.

Steve: Thanks, Leo.

Leo: Take care. We'll see you next time on Security Now!.