The Beauty of B.E.A.S.T.

Description: After catching up with the week's security news, Steve and Leo examine the implications of a recent Internet-wide exploit known as BEAST: Browser Exploits Against SSL/TLS. They share the process used by the discoverers of an exploit for this long-known vulnerability and consider its implications.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-321.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-321-lq.mp3

Leo Laporte: It's time for Security Now! with Steve Gibson, Episode 321, recorded October 5, 2011: The Beauty of BEAST.

It's time for Security Now!, the show that protects yourself and your loved ones and the Bank of America online. Here he is, ladies and gentlemen, the star of our show from GRC.com, the creator of SpinRite, the world's best hard drive and maintenance and recovery utility, but also a lot of security utilities, and a man who's really made it his mission in life to study security, Steve Gibson.

Steve Gibson: Sometimes we scare our listeners, but that helps to keep them on their toes.

Leo: Sometimes they need to be scared.

Steve: I do think, yes, I think in general if we can show people the actual consequences of clicking on links in email, then they are less likely to succumb to that ever-present temptation to do so.

Leo: Well, we do more than that, of course.

Steve: Yeah, of course we do. We've got a great show today. We've talked about something that has been much in the news recently: A couple of fun security researchers managed to turn a decade-old theoretical problem in secure connections - as we know, that's SSL and TLS - into a working exploit. So the news went all over the place. They presented at Black Hat and also at a conference more thoroughly in Buenos Aires

recently. And I promised our listeners we would, like, really dig into this. Well, when I did, I thought, okay, maybe the excruciating details would be apropos to a graduate course in cryptography, not so much an audio podcast.

Leo: It's that bad.

Steve: Oh, my god. But what I found was actually something even better, which is one of the guys' write-up of the process they went through. And so that's mostly what I want to share. We'll talk about what it is they did in enough detail that nobody will come away unsatisfied. That I can guarantee you because I understand how they did this. And it's like, okay, how do I ever explain that? But what's cooler is their chronology of the development of this because that, of course, in its nature applies, not only to this particular instance, but this is in general how these things happen. And I think that gives us a really interesting - that's something we've never had before, a really interesting look into the development of an exploit from the first little itch that something might be possible.

Leo: Hmm.

Steve: Yeah. It's going to be great.

Leo: All right. We've going to get to that in just a second. We also have security updates and news and a programming note. Next week you may be on a jury yourself, I understand.

Steve: Well, I'm on call. I start phoning in Friday evening of this week. And the good news is the court is close by. And this is, I guess, what you went through about a month ago; right?

Leo: I did, yeah. I got on the jury, which I was shocked.

Steve: Yeah, and I've never been impaneled before. But I guess - and it's funny, too, because who was I - oh, I know. I've got an NPR spot set up for next week, and the producer of that, when I was explaining that, well, I hope I'll be available, but I'm not sure that I will be, he said, well, just show up in your security officer uniform from Star Trek, and that might put them off. And I thought, yeah, that'll probably just piss off the judge because...

Leo: Yeah, the judge knows.

Steve: ...I don't think they take that with much humor any longer.

Leo: People don't like it when you try to skate.

Steve: Unh-unh, exactly.

Leo: And you know what, it's kind of fun being on a jury. So if you're on a jury, so be it. But we'll figure it out.

Steve: Oh, absolutely. I had a friend who said, just don't send back the form, and they'll, like, forget about you, even though it says you can be hauled off to prison. And I thought, well, hey, you know, I vote. And I've seen some juries; and, boy, they could use some help. So I'm happy to do that.

Leo: Yeah, good for you. You're a good, civic-minded person.

Steve: Oh, and I did exchange some email already with your staff, and I'm sure we can arrange to record Security Now! even, like, in the late afternoon or early evening. I think that Tom would be available for doing that, so that'd be great.

Leo: And also the courts in California take one weekday off a week. So you'll have a day that you could do this, even if you get on a jury. So we'll figure that out. We'll make it work. I am taking next week off for just the five days, just the Monday through Friday. So it will be Tom Merritt.

Steve: Try to even out your tan?

Leo: Yeah. Well, now, yeah. I was just saying how blotchy I got from getting some sun the other day. I'm thinking old age has struck. I need that, what is that they call Porcelana age cream, that smoothes out the age spots? But anyway, enough of that.

Steve: So is Tom going to be covering all the shows?

Leo: No, no, no. We have many people now. Iyaz will be doing This Week in Google. We have - MacBreak will be Alex Lindsay. No, we're pretty well covered now. We've got hosts. Hosts galore. We're going to get to - go ahead.

Steve: Sorry. I was just going to say Tom and I will have fun with a Q&A, as we always do.

Leo: Yeah. Yeah, yeah. That's what's next week, by the way. You can leave your questions for Steve at GRC.com/feedback. All right, Steve Gibson. We have, I think, a few security updates.

Steve: Yeah, we do. Well, actually one that's almost, well, everyone thought it was - well, okay, nobody thought it was funny. I was thinking maybe everyone but Google. As some of our listeners certainly know because I got a lot of tweets about this, an update

last week, since we've last spoken to our audience, of Microsoft Security Essentials mistook Google's Chrome browser - Google's competitive Chrome browser - as a notorious ZeuS botnet Trojan and removed it from apparently, Microsoft's instrumentation says, about 3,000 customers. So, oddly, not a huge number of customers, but a bunch.

So Microsoft posted on the 30th, they said an "incorrect detection" of this botnet "was identified; and, as a result, Google Chrome was inadvertently blocked and in some cases removed from customers' PCs. We have already fixed the issue. We released an updated signature" - actually it was a few hours later - "at 9:57 a.m. PDT, but approximately 3,000 customers were impacted." Oh, so that's why. It's probably because they caught it - it was reported, and they caught it, and they fixed it so quickly that they were able to fix this before it made more problems.

Leo: Yeah, but you've got to wonder how many people are using Microsoft Security Essentials if it only affected 3,000 people. I mean, what?

Steve: Probably it's that it was - not all systems are checking all the time and downloading it constantly, so it's being staggered out over time. And they got it fixed before it got out to many of the Windows users. And they said, "Affected customers should manually update Microsoft Security Essentials to the latest signatures. To do this, simply launch MSE, go to the update tab and click the Update button, and then reinstall Google Chrome."

Now, it turns out it wasn't quite that easy because some users were trying to do that who had had their Chrome.exe executable deleted. And it wasn't until they went into Add/Remove Programs and manually removed all of the other components of Chrome that then an updated Chrome with their updated MSE signatures was able to install. So this is the first time, you know, false positives happen from time to time. We've talked about it. I've had random AV suites every couple of years will declare one of my EXEs that's been sitting around on my server untouched for five years, it's like, oh, suddenly it's infected. It's like, no, no, it's not. Just update your patterns, report it to the vendor, and let them fix their stuff. So this sort of happens.

The problem is you can't - Microsoft can't, for example, make an exception for files named Chrome.exe. Or if they did, bad guys would start naming their malware "Chrome.exe," knowing that MSE would skip over it. So this is inherently an error-prone process. They have to do things heuristically. They can't simply make a fingerprint, like a cryptographic fingerprint, to check to see if something is a known piece of malware because now the malware is all polymorphic and changing itself from instance to instance and from infection to infection on the fly.

Leo: Explain what that means, "polymorphic." That's a good term.

Steve: Yeah, it essentially means that the malware is deliberately working to make it difficult for it to be recognized. It's built from many different small pieces, and it'll dynamically rearrange them so that the same code is not in the same location. Or there are even some algorithms that will, on the fly, rewrite themselves in different ways to achieve the same end. So it's truly different code on a per-instance basis. We know that computers are powerful. Software is powerful. And often the bad guys are spending an awful lot of their own mental cycles trying to figure out how to get around being caught

by antivirus. And so this is sort of the escalation of the cat-and-mouse has been the evolution of malware so that individual instances are actually different files. They do the same thing. They're technically the same malware. But they don't even look like the same program.

Leo: It's amazing. That's so sophisticated. It's like the rhinovirus of malware.

Steve: Yeah, exactly.

Leo: It's mutating.

Steve: So consequently that means that the AV guys have to actually do behavior analysis. They have to look at what this thing does, what things it hooks, because they can no longer look at what it looks like statically. They've now been reduced to watching its behavior dynamically and see if it's behaving in a way that seems suspicious. And so you can imagine programs that want to do the same sorts of things with a system for benign reasons sometimes get caught.

Like my code, the very popular tool I have that allows people to see what capabilities their processor has, which has been used because people have been wanting to see which levels of virtualization they have, that got called malware for a while until people complained and the AV people fixed it. And the reason was I was looking at the same sorts of things that malware needs to look at to decide how it should act. And so immature AV systems said, uh, this might be a bad thing. It's like, no. I'm on your side. So it is, it's tricky.

Leo: Well, I'm glad they figured it out. False positives are a real problem.

Steve: They're not going to go away, either. It's just...

Leo: No. And I don't know if heuristics are enough. That's the proactive kind of looking at bad activity.

Steve: The only ultimate solution is just not feasible within our current ecosystem, and that is to have all applications signed, cryptographically signed, and only run the things that are signed. We talked about this in a similar vein last week when we talked about secure boot, where every single component of the system that would be booted would have its signature cryptographically verified prior to it executing. And the immediate hue and cry went up, it's like, wait a minute, you're anti-Linux because Linux is open source, and we want everyone to be able to submit code and blah blah blah.

So the idea there is you would deny all execution unless it was proven to be a known and trusted publisher. So that flips the model upside down, and there's just no way that's going to fly because people who make freeware who aren't able to pay for and maintain certificates would just not be able to have their software run, I mean, without all kinds of warning dialogues. And then you train users just to click yes, yes, yes, yes, yes, and then you've lost all your protection. So this is a problem that we don't have a good solution

for.

The other big news was a huge problem that was uncovered in HTC handsets. What happened was HTC added a raft of logging technology relatively recently to a large number of their handsets. Essentially they were rolling it out across their line - the EVO 4G, the EVO 3D, the Thunderbolt, the EVO Shift 4G, probably the myTouch 4G Slide, the Vigor, the View 4G, and probably the upcoming Kingdom. Oh, and those are, I guess, HTC Sensation models that were affected.

So the discoverers of this notified HTC, doing the right thing, received no response. HTC just apparently blew them off. So they thought, fine, we'll go public. So of course HTC has now responded and in fact acknowledged with a blog posting that said, "A privilege escalation vulnerability allows a potentially malicious app that uses only the Internet permission [token] to connect to HTC's HTCLoggers service and get access to data far exceeding its access rights. This data includes call history, the list of user accounts, including email addresses, SMS data, system logs, GPS location data, and more."

So the guys that found it were the Android Police group, and they said, "In recent updates to some of its devices, HTC introduces a suite of logging tools that collects information. Lots of information. Lots. Whatever the reason was, whether for better understanding problems on users' devices, easier remote access, corporate evilness, it doesn't matter. If you, as a company, plant these information collectors on a device, you'd better be damn sure the information they collect is secured and only available to privileged services or the user, after opting in.

"That is not the case. What Trevor found is only the tip of the iceberg. We're still digging deeper, but currently any app on affected devices that requests a single android.permission.INTERNET, which is normal for any app that connects to the web or shows ads, can get its hands on the list of user accounts, including email addresses and sync status for each; the last known network and GPS locations and a limited previous history of locations; phone numbers from the phone log; SMS data, including phone numbers and encoded text," and then he wrote, "not sure yet if it's possible to decode, but very likely; and system logs, both kernel level and app level, which includes everything your running apps do and is likely to include email addresses, phone numbers, and other private info."

And so they said, "Normally, applications get access to only what is allowed by the permissions they [explicitly] request [and the user explicitly allows]. So when you install a simple, innocent-looking new game from the Market that only asks for the Internet permission, for example to submit scores online, you don't expect it to be able to read your phone log or [your] list of emails." So that's the story behind that.

So essentially, obviously, the model that we have in our phones is one where - and we've talked about this in more detail in the past about Apple's model for the iPhone, where you do have careful inter-application sandboxing, where each app is only able to see those services and those portions of essentially a shared file system that you explicitly give the app to. This has been a source of some chafing for iPhone users because it does certainly restrict the kinds of things the app is able to do. People who wish their smartphone was more like a computer, where they could freely browse through the phone's file system, drag and drop files around, download with one app and transfer it over into another, those features don't exist specifically for the sake of security. It's those features that do make our PCs much more susceptible to exploitation. So we don't have that in our smartphones. And what was found, of course, was that this logging system really did not have sufficient security. So HTC probably, I expect, in the future will be a little more responsive to people who find and report problems.

Leo: Well, they only gave them five days. I guess that's normal, though.

Steve: Well, they gave them five days after no response. I mean, they didn't - it wasn't five days to fix it, it was...

Leo: They said they didn't hear anything for five days, right. Not even a, oh, we're looking into it kind of a thing.

Steve: Not even an acknowledgment, let's work together, show us your data kind of thing. Just nothing.

Leo: Right. You know there was a security flaw with the Samsung Galaxy S2 on AT&T that was kind of silly. If you had a screen lock on it, and you turned this phone on and just let it sit for a second until it turned itself back off again, and then turned it on, there was no screen lock at all. So they have to fix that, too. Both HTC and Samsung have patches to do. That was only on the AT&T platform, by the way. There's a little sloppiness going on here, and I think these handsets are just going to be so ripe for attacks, they'd better start paying attention.

Steve: Oh, Leo. Yup, absolutely. So Brian Krebs, our friend who watches security things very closely, blogged an interesting post that had a couple lessons that I wanted to refresh our listeners about. What he blogged was that - the title was "Monster Spam Campaigns Lead to Cyberheists." And a little excerpt from it said that "Phishers and cyber thieves have been casting an unusually wide net lately, blasting out huge volumes of fraudulent email designed to spread password-stealing banking Trojans. Judging from the number of victims who reported costly cyberheists in [just] the past two weeks, many small-to-medium-sized organizations took the bait."

Now, interestingly, I deliberately maintain some honeypot email accounts, and I have noticed a huge volume of subject lines "ACH Payment Cancelled" notices. And that's the subject which - in fact, I just got two this morning in one of those accounts. So, for example, just in three instances, recent known attacks within the last two weeks: On September 13th, computer crooks stole approximately $120,000 from Oncology Services of North Alabama. Also in late September, $98,000 was stolen from the coffers of North Putnam Community School Corporation, which serves the children of six northern townships of Putnam County, Indiana. And in a separate attack on a public institution, malicious hackers last month struck the city of Oakdale, California, stealing $118,000 from a city bank account.

So these are large bank transfers. And what happens is, the money is - so you click on one of these malicious emails. It installs a Trojan horse in your machine which is designed to watch you log into your electronic banking. It hijacks your credentials, uses them to then transfer funds to so-called "mules," which are anonymous individuals that have been recruited, third-party individuals recruited by these organizations to receive the money and immediately cash the money out of their accounts and then forward a portion of it, they get to keep a commission, essentially, and forward the bulk, the rest of it on to the actual perpetrators of this. So this is all going on.

And Brian concludes his posting with exactly the advice I have often stated. He said, "No

single approach or technology will stop all of these account takeovers. But preventing the theft of your online banking credentials is a critical first step." Yeah, no kidding. He says, "That's why I continue to advise that small- to mid-sized organizations use a dedicated computer for online banking," which is what we've talked about several times. "Using a non-Windows PC, such as Live CD or a Mac, is the safest approach," because these are Windows-based Trojans because they're trying to cast the widest net possible, "but not necessarily the most practical or affordable solution. An alternative approach is to access bank accounts from an isolated Windows PC that is locked down, regularly updated, and used for no other purpose than online banking."

And of course what I'll just add to that is what I have done for myself is I have said to Sue, I know doing these things online would be a convenience, but we just can't. So our accounts are locked down. The accounts themselves deny all electronic fund transfers, these so-called ACH transfers, these clearinghouse transfers. And though it means that she has to physically go in, she's got to write checks from one account and deposit them in another, it's a pain; but, boy, you know, we're talking about hundreds and thousands of dollars that disappear. In some cases there's insurance that will cover this. In some cases some of the funds can be recaptured if this is discovered quickly enough. But you just don't want this exposure.

So this is really happening. And I would imagine among our listeners that, if I talk about this "ACH Payment Cancelled" email, I mean, I'm seeing so much of it. I am absolutely seeing, in my honeypot email accounts, what Brian is talking about. There's been a huge escalation, only in the last few weeks. And of course it's because these attacks are so successful. People just can't help themselves. They click on the link.

Leo: Of course, if you're using Gmail, which I'm sure most of our listeners use, I'm sure that Gmail is catching them all and killing them.

Steve: Yes, it would have immediately come up to speed and said, whoops, this is spam, and put it in. Although remember that RSA was hacked because someone went into their spam folder and clicked the link. So even that...

Leo: Bad idea. Kids, don't go into your spam folder and then click links. That's a bad idea. So what is the subject line? ACH Cancelled?

Steve: ACH - I just scrolled past it. "ACH Payment Cancelled."

Leo: Payment - I'm just going to search in my spam and so forth, see if I see it. So when you do a honey bucket, it's something that has no filtering.

Steve: Yeah. So I'm looking at two emails. ACH01 is the "from" address. And the subject line is "ACH Payment," and then it's got a random number, a random seven-digit number to make it look more authentic, and so that a subject line match won't.

Leo: Ah, I found a couple.

Steve: And then it says - huh?

Leo: I found a couple.

Steve: Uh-huh.

Leo: Escalation. Re action required.

Steve: Yup. In fact, I've got one here on 8/2 and 8/3. I've got two more on 8/2 and 8 - so that's just in the last couple days.

Leo: These are from - this is in July and from AP Associates. They're meant to look like an Amazon Associates account.

Steve: And actually what's changed, and Brian talks about this, is they're now coming from a different source that apparently - I'm looking at, okay, here's 8/3.

Leo: I think I fell for this, actually.

Steve: It's NACHA.org, which is a well-known non-profit organization. And in fact the one I'm looking at here has a .pdf.zip attached. And there's a different attack which is using a subtle glitch in URL rendering. There's a Unicode character that induces right-to-left reading. And so it's been used as a hack, a new way of covering up the fact that this is actually a .zip, which is - it's made to obscure that and make it look just like a .pdf. So, I mean, these guys are just doing everything they can.

Leo: Now, I have to say, this is I think actually legitimate. ACH is Amazon Associate payments. And the link in here is - and I can see the actual link - is associates.amazon.com. So it's not a bogus link or a hidden link. It's not an HTML email. And then you say there might be a PDF attached? Is that another vector of attack?

Steve: Well, what I'm seeing is, yeah, I've got two of them.

Leo: So what I'm saying is the one you've got is an attempt to duplicate legitimate Amazon emails.

Steve: Ah, okay.

Leo: I would guess.

Steve: Although mine clearly comes from NACHA.us.

Leo: Yeah, this is from Amazon.com. Although you could spoof that.

Steve: Yeah. And Brian did talk about this as the source of these new emails. So anyway, there just is a lot of this going on. So all of our listeners, don't click those, no matter where you find them. And really do take the issue of protecting your online electronic funds transfer-accessible accounts carefully because the bad guys want to get your money, unfortunately.

Leo: Yes, they do.

Steve: Yeah, and they're clever. So, last week I completely forgot to talk about something from the prior week. And if it's not in my notes, because I'm notes-driven here during the podcast, I forget it. And so I forgot to write it down, and I just cruised right past it. We talked two weeks ago about how Bitcasa was doing file merging, and how they were managing to offer truly unlimited file space under the explanation that they were merging files from different users, yet they were unable to decrypt it.

And I'm really impressed with our listeners because I on the spot cooked up a complex means using public key encryption which would allow that to be made true, using a random encryption key for the file, which was then encrypted under the user's private key. And if Bitcasa wanted somebody else to have access to it, then they were able to pass that through using the other person's public key and so forth. Well, a bunch of listeners said, uh, Steve? And actually it had occurred to me, too, by the time I saw it. But I've got to give credit to all of our listeners who came up with a super elegant, much simpler solution, which is exactly what Bitcasa, among other things, is doing. You're going to love this, Leo. This is just beautiful, beautiful crypto.

Leo: Okay, cool.

Steve: They hash the file, and that's the key.

Leo: Okay.

Steve: So think about it. If you...

Leo: So they're using the plaintext hash as the key?

Steve: No. Well, you take a file...

Leo: You're hashing the plaintext.

Steve: Yes. You hash the plaintext, and that gives you a signature. And you encrypt the file with that.

Leo: Clever.

Steve: So only if you have the file...

Leo: You can only unencrypt if you have the plaintext.

Steve: Yes. And so the idea is, well, just incredibly elegant. So all that you ever share with Bitcasa is the - well, okay. All you ever share, all that is ever shared with Bitcasa is an encrypted file. And the file is encrypted under or using that unencrypted file's hash. So only somebody who legitimately has the identical file is able to hash it and get the same key.

Leo: Ah. Got it.

Steve: And then you could either do a different type of hash of the plaintext, or simply hash the encrypted file sort of as the token. So you, a user, you hash the unencrypted file to get the key. You encrypt the file with the result of that hash, with that key. Now you have the ciphertext. Then you hash that to get a token which represents that file.

Leo: Now, the hash is considerably shorter than the original file.

Steve: Oh, gosh. The hash, remember, it's going to be 256 bits. It's going to be, like, that's - it's going to be a fixed-size digest of the entire file, no matter how big the file is.

Leo: So you run it across - so let's say we all have the song "I've Got a Lovely Bunch of Coconuts." And I run that...

Steve: Leo, that's the song I had in mind.

Leo: I thought it might be.

Steve: Amazing.

Leo: And so then I - and I don't know what it is, an MD5 hash. It's some common hash algorithm.

Steve: Let's hope that it's an SHA-256.

Leo: Yeah, MD5 was cracked, we know that, yeah.

Steve: We would like to have it also produce a 256-bit key, which we then use AES-256 in order to encrypt.

Leo: So we take the plain file, the unencrypted song. We hash it using SHA. And we get a 256-bit key.

Steve: Exactly.

Leo: Okay. And then, now, then we encrypt the file using that key and upload it to Bitcasa. Any - no?

Steve: Well, actually we - no, because we don't want to be uploading files they already have.

Leo: Well, they get, oh, yeah, yeah, right. We push the button to upload it, and Bitcasa says, no problem, I already have that key.

Steve: Yes. So actually we push the button to upload it, and the pushing the button hashes the encrypted file to produce a signature for it. That gets sent to Bitcasa, 256 bits.

Leo: Just to check.

Steve: They check.

Leo: They say, yeah, we got that, yeah, yeah.

Steve: Yup, exactly. And they say, you're covered. So...

Leo: Now, this still would make you open - but this all comes from this whole issue of how do you encrypt and only keep one copy of a file, which we were trying to figure out. Now, this still makes you prone to the same problem that, let's say I'm the MPAA, and I'm looking for people who have "The Hurt Locker."

Steve: Yes, and so...

Leo: I do the same thing, and now I have access to all the people who have "Hurt

Locker."

Steve: And the lesson is, this is not to avoid, I mean, this is not to enable piracy. Everyone...

Leo: It doesn't make you anonymous.

Steve: Precisely. Well, it doesn't make it impossible to prove what files you have. If that's what you want, you've got to encrypt with your own private key and pay some money to use the cloud. This is free because it's unlimited because it is amalgamating all the files across all of their customers, which allows them to offer unlimited storage for free. Now, frankly, Leo, what you could do is there's lots of metadata in the headers of audio files and video files. So you could change, you could just change one bit...

Leo: One bit, and it's different.

Steve: Yes. And so, strange how this one guy doesn't share any files in common with everyone else. Yet we're still offering him unlimited file upload storage.

Leo: So that's the problem. If everyone were to do that, it would break Bitcasa's model.

Steve: Exactly.

Leo: So is this - I forgot how we - okay. This all came up because Bitcasa, I think, won TechCrunch Disrupt, or was one of the - now, didn't win it, but it was one of the big names that was very excited, people were very excited about it. And they purport to offer unlimited cloud storage for, I think it was 10 bucks. Or free, or whatever. And we were trying to figure out how can - and they claimed that it would be private because they support encryption.

Steve: That they were unable to respond to a subpoena to produce the file contents.

Leo: Right, right.

Steve: And so we quickly, two weeks ago, came up with a hack that was overly complex.

Leo: This is easier.

Steve: But really what I wanted to share with our listeners was just the elegance of hashing the file to produce its key that you then use to encrypt it. That's just spectacular

because...

Leo: Web8349 asks about hash collisions. How often does that happen?

Steve: Yeah. With an SHA 256-bit hash - and that's, again, another reason you want a lot of bits. You'd rather - many hashes are secure with 128 bits. 256 bits, oh, my goodness, even a birthday collision where two arbitrary files have a collision, it's just, it's apt, it's, well, and you could also keep the size. So do the hash and the size. And the possibility of a collision both in hash and size is, like, astronomically low.

Leo: Got it.

Steve: But that's a great question. So this is completely miscellaneous, but I just wanted to remind our listeners. I encountered somebody the other day who runs their Kindle battery to the ground. That is, they read and read and read and read, and actually my mom does that, too...

Leo: I do, too.

Steve: ...but I'm not going to try to change her behavior.

Leo: I do it, too. Is it bad to do that?

Steve: Very bad. Lithium ion batteries hate that. I've done, just in the last few months, I did a lot of research into care and maintenance of lithium ion batteries for the whole Portable Sound Blaster project. And I saw again a lot of information that says that's not the way to get the maximum lifetime out of lithium ion batteries. Now, it's one thing if the batteries are removable. But increasingly on our electronics devices they're not. Kindle's batteries are not user serviceable.

Leo: They used to be. They were in the first one. Not anymore.

Steve: Yup. Mac Air batteries are not; iPhone, iPod and so forth batteries are not. So I just wanted to say to our listeners, what lithium ion batteries want is to stay charged. So, I mean, if you want to get the maximum lifetime out of them, just in the same way that phones - phones generally stay charged because they have such a short lifetime of use that people get into the habit of docking them at the end of every day. Sometimes even at work they'll go plug them in to bring them back up. But something like the Kindle or like the Air and so forth, in general, if you can, plug them in. So just, again, pure miscellany, but I just - every time I encounter people running their batteries to the ground, I think, oh, lithium ion doesn't like that.

Leo: See, the reason people do it is because that was the thing to do with nickel

metal hydride.

Steve: Exactly.

Leo: You wanted to fully..

Steve: NiCads.

Leo: And NiCads, too. You want to fully discharge them because of the memory effect.

Steve: Exactly. And so that absolutely is the wrong procedure for lithium ion.

Leo: It's the opposite for lithium ion. Just remember this, folks. So you're saying, whenever you have the opportunity, plug it in.

Steve: Yes.

Leo: Smartphone, anything. This is good to know.

Steve: Kindle. You'll get much more lifetime if you keep those guys charged up.

Leo: So you just don't want to deplete them all the way.

Steve: Right.

Leo: Is that the issue, is letting it go all the way down?

Steve: Well, it's just the nature of the way at the molecular level that the chemistry works, is the chemistry of the battery is happier, for lack of a deep electrochemical explanation, if you keep it charged. They like to be charged. And in fact, there's one thing that can happen is that the batteries of unused devices will drop below a threshold, and then they say that they're dead. It's like, not even...

Leo: They can't be recharged, yeah.

Steve: It cannot be recharged. There is actually some circuitry on every single cell that prevents the cell from being overcharged. That creates a tiny drain on the cell, which is why, if you don't use your device for a long time, like you pick it up after a few months,

you'll find it died, even though you weren't using it. It's because there's a little bit of drain on lithium ion cells which is part of the cell protection circuitry. So, again, keeping it charged makes it happier.

Leo: Wow. That's really good information, thank you. I'm going to plug in everything.

Steve: Yeah.

Leo: Like your laptop. You should leave your laptop plugged in when it's sitting on the desk.

Steve: Yes, yes. And what will happen is sometimes the laptop battery management circuitry, or actually the software, will say, you know, we need to do a couple charge cycles to recalibrate our battery meter. And that's an explicit sort of reconditioning process that the laptop will take you through. Normally you leave it plugged in, and the circuitry itself deliberately stops using the AC power, takes the battery all the way down, brings it all the way up, takes it all the way down, brings it all the way up, and that allows the metering circuitry to see how long, what the lifetime is in order to then properly guess what percentage of fullness the batteries are at any given time. But you'll generally get a pop-up that says, hey, we need to do some reconditioning, so leave your laptop plugged in overnight, and we'll take care of it for you.

Leo: Good to know. This is very useful. Thank you.

Steve: Yeah. So I just did want to mention that a lot of our readers are picking up on Honor Harrington. I wanted to encourage...

Leo: It's time - wait a minute, I should get a sounder.

Steve: No, no...

Leo: Doo doo ta doo, it's time for the Honor Harrington Update, de de de.

Steve: I'm just - I'm going to keep this very short because a couple grumpy people said, okay, you need a sci-fi podcast...

Leo: Well, there is one. But okay.

Steve: So I just wanted to encourage people to send me feedback. I would love to hear what people think either way. So put it in your GRC.com/feedback, and put "Honor Harrington feedback" or "Honor Harrington whatever" in the subject line so it'll catch my eye because I would love to know what people are thinking.

Leo: Yeah, remember I was going to try it and report back to you. And I have downloaded - I downloaded the Audible version, and it was horrible. The reader on the Audible version is the worst. So I started it, and it was just - just my opinion. And maybe others like it, I don't know. But I just - I was very disappointed. So, and I didn't go very far, so maybe it got better. But I just didn't want to spoil it; right? It was just the pro- I didn't even finish the prologue. Just a page, and I'm thinking, this woman is the worst.

So then I downloaded - the nice thing is you can down- now, I felt I've already bought the book because I bought it on Audible. So I felt like, well, now I can download the free version, which I did, and put it on my Kindle. And it formats quite nicely on the Kindle when you download the free version. So I did that, and I'm reading it on the Kindle. But that put me back a little bit. So I've only read the first chapter. I like it so far. It's a little - it's a little bit of a potboiler for me. But I like it all right. I think Peter F. Hamilton's a better writer.

Steve: Oh, well, no one's better than Peter.

Leo: Yeah, he's a great writer.

Steve: He's top of the food chain.

Leo: Yeah, yeah. And I will go on Audible and rate that as a poor performance because you can rate the performance as well as the book. So after I read the book, I'll rate the book highly, and I'll rate the performance low.

Steve: Well, so we will like to know what you think.

Leo: I will let you know, yeah.

Steve: I tweeted about a strange article that I encountered on Gizmodo when I was doing some research for the HTC attacks. I was looking at Gizmodo's page because they had some good coverage of it, and this other, like other top-read stories caught my eye. And this was one about claiming that the big problem, Leo, the big problem...

Leo: Yes. The big problem.

Steve: ...with interstellar space travel...

Leo: Yes?

Steve: ...was the difficulty of having sex in space.

Leo: That's the big problem? I think - is it difficult or more fun? I don't know. I think the jury's still out.

Steve: Nothing I have ever tweeted before...

Leo: What did you tweet?

Steve: ...generated so much reply, many of them clever, like, well, they're just not doing it right.

Leo: That's right. You're holding it wrong, yes.

Steve: And more, you know, give the engineers some time, they'll figure it out.

Leo: You know, there have always been rumors, much denied, but that both the Russian and the U.S. space programs have experimented with that because, for that very reason, because that's an important human function, and five years' travel to Venus or whatever, you don't, you know - but the Americans hotly deny it. The Russians not so much.

Steve: What do you mean, "hotly"? Well, anyway...

Leo: NASA says absolutely not.

Steve: I'll tell you, the space suits make it a real problem, though.

Leo: Well, but in the shuttle, you know, or the ISS. You could do it in the ISS. And I would, not to get too randy here...

Steve: How many miles high club does that make you?

Leo: That's pretty high.

Steve: That's way out there.

Leo: But I would think, and I may be wrong, maybe I've read too much science fiction, but weightlessness would make it interesting, to say the least.

Steve: Oh, I mean, and...

Leo: Peter F. Hamilton has a lot of sex in space.

Steve: Anyone who's read enough science fiction knows that we have solved this problem. This was - we need to deal with faster-than-light travel. We don't need to worry about how to have intercourse with zero gravity. And some people said - and this was quoted. This was a serious biologist in a university...

Leo: Well, it's legitimate. That's a...

Steve: ...who was really very concerned about this problem. And I'm thinking, boy, she doesn't get out very much because there's definitely going to be a way to solve that. So anyway, I just - I wanted to thank everyone who tweeted, got a kick out of my tweet and sent their comments back because...

Leo: Now, on a waterbed, forget it. But space I'm sure.

Steve: Yeah. And I did want to mention that in my opinion it took four generations, but Amazon has perfected the Kindle.

Leo: Oh, I agree. You got the little one, the basic.

Steve: Oh, it is done.

Leo: Isn't it sweet? Just the right size now.

Steve: Yes. It is cute, and it, I mean, it's done. It is perfected. Again, it took them four shots. But, boy, they're done now. So I'm just completely happy. It's a spectacular little device. And for $79, come on. I just hope they do really well with it.

Leo: I think there's no doubt they will. And actually I can't wait to get a Fire. I know you ordered one, as well. The Touch hasn't arrived yet, has it?

Steve: No. And that's actually a little bit later than the Fire is expected to.

Leo: Oh, interesting.

Steve: Yeah. I did want to share an anonymous note from somebody who had some feedback about SpinRite, my bread-and-butter product. He said, "I've been a user of SpinRite through its many versions, and it has saved me from complete disaster more than a few times. Months ago I upgraded to SpinRite 6.0" - the current version - "even though I didn't have an immediate need for it. But this morning my normally problem-

free laptop wouldn't boot. It just went into the BIOS setup over and over. So I booted and ran SpinRite 6.0 from the CD. It immediately found and fixed a defective sector at the beginning of the drive. I then rebooted normally, and everything was fine. Thanks again, Steve. SpinRite has saved me many, many times what I've spent over the years. It's no doubt the single most valuable disk utility in existence." So I wish I had his name to thank him, but he sent it anonymously. So thank you, Mr. Anonymous Person.

Leo: Yay. Happy anonymous person, thank you.

Steve: Yeah.

Leo: Okay, Steverino. What the heck is BEAST?

Steve: So the story begins a decade ago. I checked some dates, and the Transport Layer Security, TLS, which is the evolution from the so-called SSL, Secure Sockets Layer, its RFC 2246 is dated January of 1999. So what's that, that's 12 years ago. The next generation, the point update to it, TLS v1.1, has an RFC of 4346 versus 2246, so, whoa, [2100] RFCs later. Of course, you know, the Internet's been going crazy since then, so lots of publications. But it's dated April '06.

In drilling down to find out where this problem first occurred to someone, I found some email dated 2001, so fully a decade ago. And in some of the documentation at OpenSSL.org, so the open source version of the Secure Sockets Layer and Transport Layer Security, TLS, which was - and that document I found was last dated '04, but was originally written earlier - said, quote, "There are some problems with the CBC-based cipher suites in SSL 3.0" - which is the last version of SSL - "and TLS 1.0" - which was the first version, obviously, of TLS. And they're actually the same suite. They basically renamed - they made tiny changes to SSL 3.0, nothing substantial, and basically renamed it because they liked TLS 1.0 better - "that can be exploited by active adversaries under specific circumstances."

So that of course says nothing about the problem. But there was lots of analysis of the problem at the time, and it was one of the major fixes in TLS 1.0, which then abandoned SSL completely. SSL is no longer moving past 3.0, where it stopped, where it died, essentially, and was taken over by TLS, which is the same thing, just under a different name. So one of the things they fixed with that updated RFC, but essentially this has been a problem that's been known for 10 years, appeared in TLS 1.1.

Now, the problem is that there was never any pressure on anyone, really, to adopt TLS 1.1, let alone 1.2, which is the currently available version. The standard exists. It's even in Windows 7 and Microsoft Server 2008, and it's disabled. It's in OpenSSL and it's disabled, that is, these later versions, because there have been some compatibility problems, notably with IE6 that had some trouble with it. So because there was no, like, urgent need to move off of 1.0, no one has.

Well, that changed, happily, when a couple very clever security researchers, Juliano Rizzo and Thai Duong, settled down to figure out how to make this thing happen. As I mentioned at the top of the show, they talked about it at Black Hat. They demonstrated at the ekoparty Security Conference in Buenos Aires. And I'm going to talk about the attack in detail in a minute. But first I think what's really interesting and educational is to understand how this happens. How do a couple sharp guys who kind of get an itch about

maybe there's something here, how do they, like, nail it down to something that actually works? So in Thai Duong's blog on BEAST, September 25th, so that's the Sunday before last, he writes:

"So we gave a talk and a live demo at ekoparty last week to show how BEAST exploits a weakness in SSL to decrypt secret cookies.

"Please note that BEAST does not do any harm to remote servers. In fact, no packet from BEAST has ever been sent to any servers. We chose PayPal because they do everything right when it comes to server-side SSL, and that is good to demonstrate the power of BEAST, which is a client-side SSL attack. We reported the vulnerability to browser, plug-in and SSL vendors several months ago." So these guys were being very responsible.

"Current version" - and English is not this guy's first language. So I'm going to read this as he wrote it because it's kind of quaint, and they have a sense of humor which comes through, too, which I like. "Current version of BEAST consists of JavaScript/applet agents and a network sniffer." And actually it's more than that, it's a full-on man in the middle that needs to be able to not only sniff, but block and intercept and alter network traffic. But we'll get to that in a second.

"We have some choices for the agent. At the time we reported the bug to vendors, HTML5 WebSockets could be used to build a BEAST agent." Now, that's significant because all modern browsers support HTML5 with WebSockets. So that meant that just native technology in browsers would work. "But, due to unrelated reasons, the WebSockets protocol was already in the process of changing" - this is a few months ago - "in such a way that stopped it. We can't use the new WebSockets protocol shipped with browsers. We use a Java applet in this video, but please be aware that it may be possible to implement a JavaScript agent with XMLHttpRequest, as well." Now, that's the whole - that's the part of JavaScript where the browser is making queries back to the server in order to do all kinds of fancy, on-the-fly things. So this XMLHttpRequest is a technology also built into contemporary browsers.

So he says, "Be aware that that it may be possible to implement a JavaScript agent" - meaning that doesn't need Java also, a Java applet - "with XMLHttpRequest, as well." Then he says, "Why don't you take a look? Note that it is relatively easy to run a script or an applet in your browser without you doing anything (e.g., by intercepting any HTTP requests from your browser)." That is, like the man in the middle could intercept the request and then return content that would cause your browser to invoke a Java applet from somewhere. So, and that somewhere is another problem because we have the whole Same Origin Policy that prevents things from coming from places other than where we're surfing.

So he writes, "It's relatively easy to run a script or an applet in your browser without you doing anything by intercepting any HTTP requests from your browser. After all, each agent is just a piece of JavaScript or an applet. Once an agent has been loaded, BEAST can patiently wait until you sign in to some valuable websites to steal your accounts." Now, that's significant because some of the advice I've seen, for example, some of the stuff that Microsoft has been advising people who are worried about this is close all the other tabs on your browser and then log in over HTTPS to a secure site, and you're going to be safe against BEAST.

So what they're saying is that some other tab could have been compromised. And because this is not running, even though it's not running in your current tab, BEAST running on that other tab in cahoots with this network sniffer man in the middle could perform the attack, which is true. So, and again, one of the things I'm sure our listeners

are going to take away from this is this is not something to worry about a lot. I mean, this is a very sophisticated proof of concept. The good news it's gotten everybody's attention, and we're going to close this problem before it ends up being escalated into a serious attack.

So he continues, "In order to make the Java applet agent work, we have to bypass the Same Origin Policy," which is to say that has to get bypassed, too, because they've got to load the Java applet from somewhere that won't be the site you're visiting because PayPal isn't going to send you this malware Java. It's going to be, it's going to come from a server that these guys provide. And we already know that the Same Origin Policy in all of our browsers, which is avidly supported because it prevents so many exploits, that's going to stand in the way of this thing happening, too.

So they said, "Some people have gotten the impression that BEAST required an SOP bypass bug to work and so it's not a threat by itself. That's not true. It is well known that even with a SOP bypass in Java, you cannot read existing cookies. [But] you can send requests and may read responses," which actually is part of what their BEAST system does, which he says "may include new cookies; but, no, you cannot read existing cookies. In the video (and the live demo, as well) we show clearly that we decrypt existing cookies that were already stored in the browser's cookie jar. During our research, we indeed found" - and here, this is key. "During our research we indeed found a Java SOP bypass. We wanted to focus on more important parts of BEAST, such as the actual crypto attack and optimizations, so we stopped looking for alternatives and used the SOP vulnerability [that we had found for our purposes] to make an agent."

So that's really significant and interesting, too. So here are some guys who say, okay, in order to do this, we need to be able to load malicious Java from somewhere else. So Java doesn't - Java is designed to prevent that. The browser is designed to prevent that. But we need that. So they just do. Like there just happens to be one when they go looking for it that exactly suits their purposes. So that's sort of a summary.

Then he says, "It began [this search of theirs] with the alleged backdoor in OpenBSD's IPSEC stack." And Leo, you'll remember we covered that and talked about it like, what, a year ago or so. He says, "One day in late December 2010, Juliano sent me an email telling me that he got a new idea. He was at some beach in Indonesia reading tls-cbc.txt."

Leo: Oh, that's just sad.

Steve: And then he says, "I know that 'beach' and TLS and CBC should not appear in the same sentence [as the beach]."

Leo: I agree.

Steve: "But, well, maybe [Juliano] is not very interested in bikinis." Actually, later we learn that Juliano was interested in bikinis because he's not making much progress on the beach. And so he says, "And he came up with the chosen-boundary attack," which I'll get to at the end of this, the "chosen boundary attack on TLS/CBC, which is the TLS protocol when you're using the CBC cipher."

TLS, and our listeners who remember, we did a whole podcast on how SSL works. You'll

remember that the browser in its initial SSL handshake sends a list of all the different cipher suites that it supports up to the server. And from among them the server chooses whatever one it wants in order for them to agree on something that they both know about. So not all of these cipher suites use CBC. Many of them use RC4. Now, RC4 got a bad rap because that's what WEP used, and it was the misuse of RC4 - which is just a fine cipher, but it was misused in WEP, which meant that some keys that users might use were weak, and essentially that was one of the ways in. But RC4, when used properly, as it's used by SSL and even TLS properly, is perfectly secure.

And so one of the things Microsoft has suggested is that webmasters could change the order of preference of cipher suites used by SSL and TLS so that RC4 suites are chosen before CBC, and that also completely solves this problem. And everybody supports RC4. Google has actually been doing this for a long time. They didn't move to CBC, so they've never had a problem. So he says, so Juliano on the beach, reading this technical document when he wasn't being distracted by the bikinis, came up with the chosen-boundary attack, which I'll explain at the end.

"In hindsight, it's obvious that somebody would think of that when they read about Dai's attack." Dai is a previous security researcher who laid some of the fundamentals of this. "In hindsight, however, everything is obvious." Which of course is true. "It takes somebody like Juliano to have such a good idea. I am so lucky that he always shares his ideas with me. I wrote some test cases using the wonderful TLS Lite library, and after some hours my conclusion was it can't work in browsers. I then moved to the States and was busy with new life, new job and schooling, so I didn't have time to research that idea any further, even after Juliano kept asking me to check it again." And this guys says, I love this: "Don't listen to me if I tell you your attack can't work," and he has a smiley face. Don't listen to anybody telling you that.

"Fast-forward to early April [2011]. I was working on some project at Matasano, and some SSL code that I saw that day made me realize that I wrote wrong tests for Juliano's idea. In fact, it seems that I didn't quite understand it back then. As soon as I got back home [that night], I re-read Juliano's email [from December], my notes and scripts. I decided that I needed to make it work with pen and paper first, so I drew some diagrams. The result looked hopeful. I then modified the test cases, re-ran them, and for the first time I saw that chosen-boundary attack may work. That moment was so wonderful.

"It turns out that I had to 'reverse' the idea to make it 'compatible' with browsers; and, after some more hours, not only I saw that the attack is possible, but I also understood which conditions I need to make it really work. The conditions looked easy, too. I was very excited. I would have screamed loudly, ha ha ha. I took note of what I had seen so far and sent it to Juliano.

"At first, Juliano didn't understand my note because it's a reverse of his idea, but he caught up very quickly, and he agreed that it looks doable. So the main condition is to be able to send two SSL records in the same cookie-bearing request. If you've read the leaked draft, we call this the 'blockwise privilege.'" And I'll explain what that is, also. "We were both very excited because at that moment we know that it is just a matter of browser features for us to create a reliable exploit for something that people have thought un-exploitable in years.

"I collected a list of browser features and plug-ins that allow me to send cookie-bearing requests." So now they've got the fundamental concept. They have figured out, okay, this kind of looks like we can make this happen. So now it's time to convert it to code. He says, "I took a look at the "Browser Security Handbook" by Michal Zalewski" - who we've

also talked about and mentioned many times in the past - "and found some candidates: JavaScript XMLHttpRequest" that we talked about, "Java URLConnection, Flash URLRequest, and Silverlight WebClient API. We really wanted to make the attack work with [just] native JavaScript, so we spent a lot of time on XMLHttpRequest object. At some point, Juliano and I were even reading C++ source code of various browsers (which is even harder to understand than assembly, as a friend once said).

"Maybe we've missed something really obvious, but we've never made XMLHttpRequest work the way we need. It was both surprising and frustrating that it is so hard to do request streaming in modern browsers. People have to invent a lot of ugly" - and he says "bitches," I think maybe hacks - "known as Comet techniques. I studied every single one of them, but none of them meets what we need. I also studied HTML5 WebSockets, then concluded that it's not what I need because it includes a [null] byte in front of every packet. More about WebSockets in a moment.

"So I moved on to Java URLConnection." So that's an API in Java that allows outbound connections to be made. He says, "I had never written an applet before," so he'd never written a Java applet before. He briefly refers to learning Java here in a second. He says, "But researching is doing exactly things that you haven't done before. So I wrote an applet and tried to make it perform the blockwise step. The Internet is really helpful. I found a wonderful 'URLConnection Mini Guide' in Stack Overflow. I also wrote a small SSL server to test my applet. It worked as expected. I could open a request, append more data to it as long as I want" - and that's the key, as we'll see in a second - "and each time I 'flush' the output stream, Java would send out a record in the same SSL connection. So wonderful, but I want more. I want something that works without any plug-ins. Once again I started writing tests...."

So he's got something using Java, and the idea is he's able to send out multiple HTTP requests, or HTTPS requests actually, over the same connection. Thus he calls that "streaming." And these are requests from Java out to a remote server. But as we'll see, they don't make it. They get intercepted by this man in the middle. But so now he's got Java, but he wants to refine it. He really wants to just use JavaScript because it would just make it a better attack.

So he says, "I want more. I want something that works without any plug-ins. Once again I started writing tests for XMLHttpRequest," which is just the Java, which is the Java command or API for doing the same sort of thing, for JavaScript being able to reach out and query remote servers, generally using XML, but it really can be anything you want. So he says, "...reading [more] C++ code or studying Comet." And he says, "Rinse and repeat. Nothing worked.

"One day, while in the shower, I realized that those things haven't worked simply because they can't work. That's why people have to invent WebSockets. Hmm, why couldn't I use WebSockets? I re-read my note. So they have a" - and he has a "\x00," which is C notation for a null byte - "prefix, which prevents me from fully controlling the chosen block [I want to send]. I remembered that [Mihir] Bellare et al. also had the same problem when they tried to attack SSH, so I re-read their paper. I was quite disappointed to know that they didn't describe any practical way to solve that problem.

"Then I came up with the idea of chaining of predictable" - he says IVs, and that's Initialization Vectors that we'll talk about in a second. "I wrote a small WebSockets simulator to test it, and it works. Not very fast, though, since WebSockets requires that my chosen block to be a valid UTF-8 string. It's funny that we also had to deal with UTF-8 in the ASP.NET exploit. Anyway, it doesn't need any plug-ins. It was late April.

"We split the work. I wanted to work on the paper, and Juliano wanted to work on the exploit. I started writing [the paper] right away, but Juliano had to delay the exploit several months later. He got a name for it, anyway. 'This thing is so complicated, I would like to call it B.E.A.S.T so people kind of get stuck calling it "the BEAST attack." We can figure out what BEAST means later,' [he wrote].

"Writing in English has never been easy for both of us. It took us several months with a lot of help from friends to finish the ASP.NET paper, and I couldn't believe that I had to write" - that's not this one. That's the work that they were known for before, about a year ago - "and I couldn't believe that I now had to write another one even before releasing that [one]. But I did eventually. Along the way, I found Bard's papers, which was extremely helpful for me. I read his paper carefully. So many 'We believe,'" he has in quotes. We believe this; we believe that. "I have to confess that I am a non-believer, so I made a rule for myself: no 'We believe' in my paper. Anyway, although Bard's attacks can't work, his papers were written in very nice English. So I stole a lot of phrases, expressions and statements from him."

Leo: I believe he did.

Steve: "Of course I cited him many times, [too]. So I kept writing, and Juliano helped with editing. By mid-May, we finally had something good enough to ask for review from friends. I guess that no 'We believe' is a good rule, since everybody said that the paper is easy to understand. We also got an" - I don't know who his friends are, but boy, they're post-graduate cryptographers because I've read the paper, too, and it's like, well, there's no 'we believes' in it, but there sure is some serious, whoo, some serious crypto.

Leo: I love it.

Steve: "We also got an awesome review from Kenny Paterson. If you happen to write a crypto paper and need some cryptographer to review it, you may want to ask Kenny. He's very friendly, encouraging, and his review always teaches us a lot. So we got a not-so-bad paper, but still no actual exploit because Juliano was still in some beach somewhere. We agreed..."

Leo: I guess he found the bikinis.

Steve: I think the bikinis actually were a distraction. "We agreed that we should contact browser and SSL vendors early so they can work on the patch, since we planned to, but didn't, release something in July. We sent the paper to Google, Mozilla, Apple, Microsoft, Opera, and Oracle. All of them responded very quickly, which is [very] impressing. Mozilla created bug 665814, and it became the discussion board of all people working on the fix [within the industry]. Later I discovered that there were also people from IETF and other vendors [who] we didn't contact.

"It turns out that fixing this is not easy because every proposed solution is incompatible with some existing SSL applications. OpenSSL has already included a fix several years ago, but it is turned off by default, thanks to Internet Explorer 6's broken SSL implementation." And we remember how many IE6s are still out there, like half of them. Microsoft just can't seem to kill that thing. "Somebody also pointed out that, due to

another attack released in March, the WebSockets protocol was already in the process of changing in such a way that stopped our attack." So that forced them to go back to Java.

"People kept asking for a working [proof of concept], but we could not give them anything [because we didn't have it yet]. At some point, it seemed that no vendor wanted to patch this. We also started losing interest in convincing them. We got more compliments from cryptographers we contacted. We didn't release anything in July as planned, since nothing has been fixed." Meaning that the patches for the problem didn't exist then because vendors were kind of wandering off. "Instead we went to Black Hat..."

Leo: I can't understand why.

Steve: Yeah, well, it's like, because the problem has been known for a decade, and these are the first two guys who are claiming they can make it work. But you know there's a lot of that going on in the background that we never hear about. And so the vendors are like, well, okay, what do you want us to do? Anything we try breaks stuff. And so there was counter-change tension from the fact that no one could figure out how to fix this without breaking things.

So they won a "Pwnie," the P-w-n-i-e, at Black Hat for their ASP.NET bug. "Delivered a presentation on the attack at Matasano's private dinner." That is, on this attack. "People liked it. We got several follow-up emails and an invitation to present it again at some internal conference of a client. Juliano and some other friends rooted all servers and yet lost their CTF game. Still no actually BEAST. 'We can work together on BEAST when I visit you next week,'" - that must have been Juliano writing, "but we ended up drinking all nights. So many high five," he writes. "Then Juliano submitted the talk to ekoparty. Opera patched. We got excited. That was five weeks ago.

"Juliano told me that the talk got a 10/10 rating from all of the judges, and he felt that it's time to give birth to BEAST. Since WebSockets is not a good option anymore, and there was so little time to research other alternatives, we agreed that we should focus on Java and Silverlight. He worked and worked and worked. It turns out that BEAST is not easy to code. At some point, Juliano had to install Windows and Visual Studio..."

Leo: Oh, dear.

Steve: I know, horrors, "...to write a Silverlight applet in, cough cough, VB.NET. Well, he has to do things he's never done before, [too]. BEAST finally decrypted the first byte of some cookies." So he says, "It is so surreal to witness some idea that exists only in your mind actually evolves into working code. Moments like this are very rewarding, and it makes researching very worth doing. Juliano was so tired, so I asked him to send the code to me. This is why we've enjoyed doing things together. We can make progress as long as there is at least one guy up. He has done the heavy work; now it's my turn to baby-sit BEAST.

"I installed Eclipse, learned Java, and started hacking the code. Since Juliano stopped as soon as BEAST decrypts the first byte, I had to fix bugs to make it decrypt more bytes more reliably. The next thing I did was to find a way to bypass the Same Origin Policy with some agent. A friend was so generous that he gave me one of his Java zero-days to bypass SOP." So there again, they're out there, and it's like he says he asks one of his hacker friends, "Hey, do you happen to have a zero-day, something no one has ever

seen before, that will bypass Java's zero-day?" And the friend says, "Oh, yeah, hold on a second, I'll email it to you."

"Anyway," he says, "that bug [that his friend provided] has a dependency that we couldn't satisfy, so I had to find another one. I started reviewing JVM's source code." Okay, the source to the Java Virtual Machine. "Honestly, I didn't expect to find a Same Origin Policy, but I did." What do you know?

Leo: What the hey.

Steve: Want some source code, there it is. "What's interesting is that the bug is very good for BEAST, but not so good for other types of attackers. You need to be able to do MiTM to use it." But that's what they needed, so it worked perfectly for them. "Well, that makes sense when you know that I found the bug when I was MiTM-ing a Java applet. I could look for other ways to create an agent, but both of us agreed that we should focus on optimizations. I needed to make BEAST fast. The version that Juliano sent me was so slow that all we got were expired cookies." I love that. "The version that Juliano sent me was so slow that" by the time they managed to decrypt it, the cookie had expired.

Leo: They were gone.

Steve: All we got were expired cookies. And then he writes, "BEAST is just a baby. It deserves fresh cookies."

Leo: Aw, yes.

Steve: "I spent the last week or so working on that. As you see in the demo, BEAST now takes [only] minutes to decrypt very long, unexpired cookies.

"In summary, we had an idea, and we've done several things we've never done before to make it work. That's our story of penetrating crypto. Thank you for your time."

Leo: Oh, wow.

Steve: So, really neat. Now, okay. I've promised a whole bunch of things, but we're running short of time. But that's kind of okay because this is not a post-graduate course in cryptography. And that's really what we get down to. What I will explain is that CBC, that you've heard me - that acronym, Cipher Block Chaining. That's a means for using a cipher like AES in practice. The problem with a block cipher as opposed to a stream cipher, a stream cipher like RC4 simply generates a pseudorandom stream of noise, of absolutely unpredictable noise. And we know, those listeners who've been listening for a long time, although I'm constantly reminded that people are still just discovering the podcast or only recently discovering it from the things that we see in our Q&As, if you mix a pseudorandom noise stream with data - by "mix" I mean XOR individual bits from the pseudorandom stream of data - what you get is a different pseudorandom noise stream, and it is absolutely secure because you've mixed noise with your real file. If you then send that somewhere, and the recipient can regenerate that same pseudorandom

noise stream and mix that with the encrypted data, out drops the original unmixed plaintext. So that's the way a stream cipher works. That's all you need to do.

A block cipher encrypts, as we know, blocks at a time. In the case of modern block ciphers, they use a 128-bit block length. Older ones use 64 bits. The problem is there aren't that many possible combinations of "just" 64 bits. So those block ciphers are not considered secure enough for, like, way into the future. AES is 128 bits. And even though the bit length is only twice as long, it's, whoa, I mean, you're going up 2^64 times more combinations when you add that 64 bits to another 64. So the block cipher, though, it takes, in the case of 128 bits, it takes 16 bytes at a time and ciphers them into a completely different 16 bytes.

But the problem is, if you simply took your plaintext and did that for 16 bytes at a time, if you ever had the same plaintext occur, then it would encrypt to the same ciphertext. So that presents some information leakage. And as we'll remember, on the Wikipedia page talking about these encryption modes, modes of encryption like CBC, they have a startling image of the Linux logo where you can see, you can still see what the image is, even though it's been encrypted in a non-modal, simple AES cipher. The penguin bleeds through, and you can see it. So it's clearly not good.

So instead, what cryptographers did was invent like a little addition to just using a block cipher. And that is, for each successive block, before enciphering that block, they take the ciphertext from the previous block, that is, the output from the previous encryption, and they exclusive-OR it, they mix it with the plaintext prior to that plaintext encryption, which creates the next block. Then they take that output and XOR it with the third plaintext and feed that through. Well, then, what about the first one? And that's the key. We need something called an "Initialization Vector," an IV, to start this chain of ciphering and exclusive-OR operations.

The mistake that was made, which was figured out a decade ago, is that there's a weakness in SSL, and that followed over in to TLS 1.0 because that chaining from one block operation to the next, it extends out between packets. That is to say that in blocks of encryption - actually it's a little more complicated because packets and blocks aren't necessarily the same thing. But AES does things in blocks. And the last cipher of the last block is used as the IV, the initialization vector of the first cipher of the next block. In other words, the same way that within a block we're chaining these together, the developers of SSL made a mistake - tiny, subtle little crypto mistake which are so easy to make - of chaining that initialization vector from the end of the prior block into the next one. That gives somebody visibility into the cipher.

So the way this attack works, and I'll say it just in a few seconds, I mean, I'm going to sum it up quickly, is that if you somehow manage to get malicious code into the site that you are contacting, or you get malicious code into a tab, essentially, of the browser, doesn't have to be the site you're contacting, but somehow you want to infect a page with SSL, that invokes Java. And, in fact, as a consequence of this Mozilla has been considering disabling Java because this particular implementation of the exploit does require Java, and so this is a rather brute-force change. Anybody using NoScript is safe from this, as long as you don't enable scripting, because you have to have scripting in order for the JavaScript and the Java to work hand in hand.

But so what happens then is you also need to somehow arrange to be a man in the middle. So this is still not something that could get you, even if you clicked on links in email. But don't do that. The attacker has to be somehow local to you and able to sniff and intercept your traffic. So what happens is, with this evilness in place, you've gone to a bad site which installed BEAST into a tab. Then it's living there. And you also have to

have an attacker doing a man in the middle right, you know, as part of this.

Then you open a different tab, and you go to PayPal. So you log into PayPal, and PayPal will give you a session cookie which is fresh and not expired. Then you poke around PayPal; and, as you do, your browser is sending back - it's making queries. And as we know, it's sending that cookie back to maintain your session state, to remind PayPal this is who you are, and you're logged in. The sniffer, which is sitting there on your connection, sees encrypted packets go by that it has no visibility into. It can know that you're going to PayPal if it, like, figures out what the IP is and does reverse DNS, I guess. But it has no visibility into that. But it sees something happen that's encrypted. And it knows that, if it's an HTTPS query, there will be headers in there that are encrypted that will include the session cookie.

So now this thing comes alive. On a different tab, BEAST wakes up, and it begins working to decrypt that SSL packet that was on a different connection to a different location, and it's able to do so because of this mistake that was made where between SSL blocks they're able, with a man in the middle, to see the last block of ciphertext, which they know is going to be used as the initialization vector for the beginning of the next block. But they know that before the next block is encrypted. That is, they're able to intercept; and, knowing the initialization vector, they can mount a plaintext attack feeding in what they want to have XORed with the initialization vector and see what the output is.

And essentially they've come up with a clever way of, within 128 guesses, they're able to get a byte. So no more than 256, but with 128 guesses, they can get a byte. Then by changing the padding, they shift the whole message over one byte, and then they're able to decrypt the next one. And so they have a byte-wise attack which requires malicious code in the browser; man-in-the-middle attack; oh, and the man in the middle, this BEAST tab which awakens, it's generating queries out to the man in the middle, reusing the connection which the PayPal tab had, so that it's using the same encryption. So they're able to essentially probe the encryption and decrypt that earlier packet that they captured which they assume had the session cookie in it.

So at this point there isn't a solution that doesn't break things. The good news is this video that we have a link to in our show notes is really interesting to watch because you see this happen. And it's probably scarier than it ought to be because all of these requirements are sort of glossed over. But they do capture the cookie in plaintext. Then they go to a different browser. They launch Firefox instead of using Safari, and they launch Firefox and give it that cookie. And sure enough, they're magically logged into that PayPal account without ever having to actually log in. So they steal that PayPal session cookie through no fault of PayPal's at all. As they said, they use PayPal because PayPal does everything right.

Problem is, we've known about this for 10 years. But because nobody actually made it happen, nobody created a video like this or gave a presentation, there wasn't any hurry to fix it. So now there's pressure. What needs to happen is that those implementations that are broken and cannot run with TLS 1.1 need to get updated and fixed, and we need to just move to TLS 1.1. And then we'll be okay. In the meantime, the browser vendors are trying different tweaks and things. Microsoft has one of their little Fixit buttons which you can click on to enable it in Windows 7 and in Server 2008 because it's there, but it's disabled, as it is everywhere else.

And what I haven't been able to track down is the nature of the incompatibility that people report when they simply enable the higher level versions. SSL protocol is designed to allow you to negotiate whichever version you both support. So we'd like to have everybody starting to support the latest versions, and but only when they both

agree would that be the one that we support. So maybe there are some bugs in some implementations of 1.1 such that when you actually try to use it, it doesn't work right. Anyway, I'll see if I can find out, and I may have that news next week. In any event, that's the story of The Beauty of BEAST.

Leo: Awesome. Thank you, Steve Gibson. If you want to know more about this, Steve puts show notes on his website, GRC.com. We also put them on the TWiT wiki. You can also find 16KB versions of the show for the bandwidth impaired. What else? Full transcriptions. Elaine takes, like, a day to do those; right?

Steve: Yeah.

Leo: So in a day, Steve will have those up there at GRC.com. While you're there, check out SpinRite, world's finest hard drive maintenance and recovery utility, and all the freebies that Steve offers, including Password Haystacks. We saw you last night with Diane Sawyer, you superstar, you.

Steve: Thanks to a friend of yours, or a friend of TWiT, Bert Rudman, and the reporter Matt Gutman, they did a nice piece about the BofA site being down for five days and needed a talking head sound bite. So some guy...

Leo: Do we have any why that site was down for so long?

Steve: Well, there's no way to prove it.

Leo: Because they're not saying.

Steve: Yes. They're denying that it was any kind of attack. On the other hand, what else could it be? It wasn't like it was an hour because someone tripped over a cord somewhere at Bank of America, or they updated something that, like, broke their site, because if that happened they would immediately roll back to the code that was working an hour before. And it also came and went. When I first heard about this earlier in the week, I went to BofA, and it was up. But then later it was down.

So, I mean, it has every hallmark of this being a denial of service attack where they were bringing up a page that had links to deeper in their site. So you couldn't get to the home page, but Akamai was serving a page which got you around it. So, I mean, it's exactly what a denial of service attack on a big website would look like, which I think it is. I think they were denying that it was any kind of attack because the public might get confused between a denial of service attack and a breach of Bank of America's security, which this probably wasn't, as far as we know.

Leo: Yeah. Interesting.

Steve: But anyway, it was fun to...

Leo: I guess we'll never know.

Steve: It was fun to be on World News Tonight with Diane Sawyer, so thanks to Bert and Matt.

Leo: Good exposure. Did you talk about, well, I guess you didn't talk about Password Haystacks.

Steve: No, it was literally, if you blinked, you would miss me, so...

Leo: And how long did it take, by the way, to tape something that, if you blinked, you would miss it?

Steve: Well, we did a more extensive interview. But when I saw the piece air last night, I realized, you know, for the general public, you were amazed when I did that piece with KTLA because the general public is really not ready for any of this stuff.

Leo: Yeah, this is hardcore stuff. This is not your - so good on you.

Steve: That's our listeners, hardcore crypto security listeners.

Leo: Hardcore crypto fanatics.

Steve: I'm glad to have them.

Leo: Steve, we'll be back next week. We do this show at 11:00 a.m. Pacific, 2:00 p.m. Eastern, 1800 UTC on Wednesdays. And it will be a Q&A with Tom Merritt. Now, there is a schedule calendar at TWiT.tv. If jury duty calls, we'll reschedule.

Steve: As soon as I know anything, I will tweet, and I will let your staff know so that we can update the calendar, and Tom and I can find an alternative way. The podcast must go through, Leo.

Leo: GRC.com the website. But the Twitter handle is @SGgrc. So that's easy to remember.

Steve: Yup. And if anyone wants to check on all of the tweets that were resulted from the sex-in-space problem...

Leo: [Laughing] Did you come up with a solution? Just out of curiosity.

Steve: Uh, I'd love to try it, Leo.

Leo: And we'll leave it at that. Thank you very much, Steve Gibson. Thank you all for being here. We'll see you next week on Security Now!.