SECURITY NOW!

Transcript of Episode #865

Port Knocking

Description: This week we examine a critical Java framework flaw that's been named "Spring4Shell" because it's mildly reminiscent of Java's recent "Log4j" problem. We'll also take a look at the popular QNAP NAS devices and several recent security troubles there. Sophos has got themselves an attention-grabbing, must-patch-now 9.8 CVSS vulnerability, and it didn't take long (10 days) for the theoretical Browser-in-the-Browser spoof to become non-theoretical. There's more worrisome news on the NPM supply-chain package manager exploitation nightmare, the FinFisher spyware firm happily bites the dust, and some of the young hackers forming the Lapsus$ gang have been identified.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-865.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-865-lq.mp3

Squarely in the doghouse this week is Wyze, whose super-popular webcams have problems which are just as serious as those of the company itself. And, oh, the authentication bypass details, which I'll share, are so wonderful! Then, after a bit of closing-the-loop feedback with our listeners, I want to talk about and put the idea of "Strong Service Concealment" on everyone's radar. "Port Knocking" is not a new idea by any means; but it is extremely clever, cool and useful. In today's world, there's more reason than ever for ports and the services behind them that are not actively soliciting public traffic to be kept completely hidden. There are a number of ways this can be done which are very cool.

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. We've got a new Java framework flaw called Spring4Shell to talk about. Yes, Steve's going to issue a slight spanking to Wyze, their three-year flaw in their cameras unpatched until January. How did that happen? And then a new way to log into servers. It's a lot easier. It's called "port knocking." But is it safe? Steve explains all next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 865, recorded Tuesday, April 5th, 2022: Port Knocking.

It's time for Security Now!, the show where we cover your security, privacy, and safety online with Steve Gibson of the Gibson Research Corporation. Hello, Steve. I'm coming up with some sort of "Let's get ready to secure" or something. I don't know.

Steve Gibson: Yes, just before we hit 999 we'll come up with a slogan.

Leo: That's right. Well, we have a slogan: "What could possibly go wrong?"

Steve: Yes.

Leo: That's a great slogan.

Steve: I think that's probably the best thing ever, yeah. So we're at, oh, are we at 864 or 865? I've got 864 in the show notes, at the top of the show notes.

Leo: Oh, I don't know. It says 865 in all of my stuff.

Steve: Good. I think I just didn't update the title page or the title of the show notes. So everyone, ignore that. It is definitely April 5th, though. That I'm sure about. We're going to examine a critical Java framework flaw that's been named of course Spring4Shell because it's mildly reminiscent of Java's recent Log4j problem, or Log4Shell problem. We'll also take a look at the popular QNAP NAS devices and several recent security troubles there. Sophos has earned themselves an attention-grabbing, must-patch-now 9.8 CVSS vulnerability. And it didn't take long, like about 10 days, for that theoretical browser-in-the-browser spoof that we talked about to become non-theoretical. It's now in use.

There's more worrisome news on the NPM supply chain package manager exploitation front nightmare. The FinFisher spyware firm happily bites the dust. And some of the young hackers forming the Lapsus$ gang have been identified. Squarely in the doghouse this week is Wyze, whose super-popular webcams have problems which are just as serious as those of the company itself, it seems, and oh my god. The authentication bypass details which I will share are so wonderful.

Then, after a little bit of closing-the-loop feedback from our listeners, I want to talk about and put the idea of strong service concealment on everyone's radar, thus the title for today's podcast is "Port Knocking." It's not a new idea by any means. Articles and conversations about it typically have dates like 2008. But the concept is really clever, I think. I've always thought it was cool and useful. And in today's world, there's I think more reason than ever for ports and the services behind them that are not actively soliciting public traffic, like a public web server does, things that are like you want to log onto your own NAS sort of thing.

If you don't need to publicly expose ports, there's just so much reason to keep them completely locked and hidden. And it turns out there are a number of ways this can be done. Linux now has a port knocking technology built into it, DD-WRT, OpenWRT. There's a lot of ways to do this now. So I just kind of wanted to talk about this, the idea that there is a way for you to be remote from your location, have no exposed open ports, yet by knocking in a certain way have the ports opened only to you at your IP, and only for as long as you want them to be. So I think an overall really interesting podcast for our listeners.

Leo: I hear you knocking, but you can't come in. Actually, if it's me knocking, let me in; right? It's okay. Picture of the Week?

Steve: Yeah. This was one that I had in the collection. It's just sort of fun. Anyone who has actually spent time programming will probably get a sense for this. The picture just depicts your kind of typical programmer dude. We've got some energy drink cans like on the desk, a coffee mug that looks like it's got some coffee dripping down the sides, some crumbled up pieces of paper. Of course the ever - you have to have yellow Post-it notes like stuck on...

Leo: Everywhere, yeah.

Steve: ...the margins of your monitor in order to qualify.

Leo: And crumpled up paper, yeah, yeah. And energy drinks, yeah, yeah.

Steve: Yeah. So at this moment we catch him raising both fists in the air, celebrating. He's like, got some success. And he says: "Wow, a different error message. Finally some progress!"

Leo: We made progress.

Steve: It's like, he's been like, what is wrong? No matter what he tries, just keep getting the same error message. Oh, look, it just changed. Okay, we're getting close.

Leo: I've got another one. I don't know if you ever get this. But I'm still working on the Advent of Code problems, you know, off and on. They're really fun programming problems.

Steve: Oh, my goodness, yeah, yeah.

Leo: Yeah, really, this one's - Day 19 was a challenge. But...

Steve: Well, and Leo, I've always told people the best way to learn a language is to use it to solve problems.

Leo: Yeah. And these are hard problems. So something worked by accident. You ever have that happen? Where I went, this shouldn't work, but it does. And then I tried it with different problem sets and different - and it works. I'm not sure why it works.

Steve: No, I've had that. In fact there have been times when I've written a huge amount of new, like, SpinRite code, for example. And it assembles correctly, no syntax errors, and then I launch it, and it goes. And it's like, I'm not sure I trust it because...

Leo: It's too easy.

Steve: I have to pay a bigger price in order to have it all work correctly.

Leo: Absolutely. Absolutely.

Steve: Well, last week we noted Chrome's second zero-day of the year and titled the podcast, last week's podcast, "Targeted Exploitation," with information thanks to Google's Threat Analysis Group research, which documented the details of the exploitation of Chrome's first zero-day of the year. This week we switched to Apple, who last Thursday pushed out patches for its own fourth and fifth zero-days of the year. Last year Google had a total of 16 for Chrome, and Apple was at 12 for the year. So for Apple to be already at five by the end of the first quarter suggests a run rate that might break both of their 2021 totals. We'll see how that goes.

Anyway, the fourth and fifth patches which, again, zero-days, cover iPhones, iPad, and Macs. And these are true zero-day flaws since in their typically obscure way they said that the issues "may have been actively exploited." Uh-huh. May? Okay, so why the emergency? I mean, they, like, this was a "Get this out now" level patch. Anyway, in any event, we have an out-of-bounds write issue in the Intel graphics driver that allows apps to read kernel memory, and that's never what you want; and an out-of-bounds read issue in the Apple AVD, that's the Audio Video Media Decoder, that will enable apps to execute arbitrary code with kernel privileges. Both bugs were reported by anonymous researchers and resulted in iOS and iPad OS both moving to 15.4.1, and macOS Monterey to 12.3.1. The first flaw was fixed by improving input validation is all they said, and the second by improving bounds checking. So okay. We'll see how Apple goes with the rest of the year.

Another worrisome vulnerability in a Java framework has surfaced. The cybersecurity firm Praetorian said that the flaw impacts the "Spring Core" on the Java Development Kit (JDK) versions 9 and later. And this is odd, too. It's a bypass for another much, and I mean much older vulnerability from way back in 2010. That one was tracked as 2010-1622. And yes, only four digits. Those quaint days when we only needed four digits to number our CVEs.

Leo: There are only 10,000 security flaws.

Steve: Yeah. What happened to that? If exploited, this bypass enables an unauthenticated attacker to execute arbitrary code on the target system, making it, of course, a remote code execution, RCE. And unfortunately, a Chinese security researcher briefly posted a working proof-of-concept for this exploit to GitHub before deleting the account. But as we know, it doesn't take long. Nothing remains hidden on the Internet. So indeed, the proof-of-concept code was quickly shared in other repositories and tested by security researchers, who confirmed it was a legitimate exploit for this new, potentially severe, and previously unknown Java vulnerability.

Leo: So it had been patched in 2010, but there was another way to get to it or something.

Steve: Yes, exactly. So probably not fixed the way we would have wished.

Leo: Right.

Steve: Which is to say it would stay fixed. This was a bypass around the way it was fixed. So Spring is a software framework for building Java applications, including web apps on top of the Java EE, the Enterprise Edition platform. Researchers who've looked at it have said that: "In certain configurations, exploitation of this issue is straightforward, as it only requires an attacker to send a crafted HTTP" - you know, a standard web query to a vulnerable system. "However, exploitation of different configurations will require the attacker to do a little additional research to find payloads that will be effective."

So in that sense it does feel like the Log4j, which as we know turned out not to be like the end of the world because it wasn't just drop-dead simple for script kiddie weenies to massively exploit. It really required more expertise. The Spring framework's maintainers, Spring.io, which is a subsidiary of VMware, last Friday released emergency patches to fix this so-called, and it actually has been called "Spring4Shell." It's a zero-day RCE. Well, that's what they're calling it. And I was trying to decide whether this would really be a zero-day if it wasn't being actively exploited in the wild against victims. We know that Microsoft has their own definition for zero-day. Normally we reserve the term, and we're trying not to overuse it by making it overly broad, to be like, whoops, we learned about this because we saw it being used. That's clearly a zero-day.

So this is kind of a gray area because it's been - the exploit has been publicly disclosed. And I imagine that in the next week or two I'll be saying, oh, yeah, it went from public disclosure to weaponized. But anyway, I think that since a public proof-of-concept exploit exists before a patch is ready, and actually a patch just happened, but certainly the proof of concept has existed for some time, it probably qualifies as a zero-day.

There was also some confusion because two other related vulnerabilities in that same Spring framework were also disclosed last week. There was a DoS vulnerability, meaning you can crash the thing, and the Spring Cloud expression resource access vulnerability, and I didn't dig into those because they've been patched, and it's like, okay. And they're unrelated to this one.

So there' also been some questioning about just how bad this RCE really is. The concern is that it is in use by enterprises for all kinds of their own custom server things, which means that it's - who knows what's wrong with any particular enterprise's implementation? After an independent analysis, Flashpoint said: "Current information suggests in order to exploit the vulnerability, attackers will have to locate and identify web app instances that actually use the" - and here it is - "the DeserializationUtils, something already known by developers to be dangerous." Okay. But that doesn't mean developers aren't using them because they're there; right? It's an API. Oh, look, this does what I want.

So again, it's not like developers are nearly as security focused as we are, and the group listening to this podcast is. And we've talked a lot about the dangers of deserialization. Java is an object-oriented language, which means that an object is a complex, or at least can be, typically is a complex data structure. So how do you store such a thing? The way you store it is you serialize the object into a byte stream, you know, a blob, which you then store. And in order to later reconstitute the object into a form that Java can use it, you need to deserialize the blob. And a deserializer is inherently an interpreter of the byte stream. And as we know, naive interpreters are written to assume that they will only ever receive a valid deserialization stream to deserialize.

In fact, in an interesting twist, we're going to see that this Wyze cam authentication flaw is sort of like that. The guys that designed the handshake just assumed you'd be a valid handshaker. But no. Anyway, the security firm Rapid7 said that despite the public availability of proof-of-concept exploits, "it's currently unclear which real-world applications use the vulnerable functionality." Which is really just to say we don't yet. This just happened. So we need exploits in order to - we need actually to have some problems before we know. And they said: "Configuration and JRE version may also be significant factors in exploitability and the likelihood of widespread exploitation."

However, CERT CC's oft-quoted vulnerability analyst Will Dormann tweeted, he said: "The Spring4Shell exploit in the wild appears to work against the stock 'Handling Form Submission' sample code from Spring.io. If the sample code," he says, "is vulnerable," he said, "then I suspect there are indeed real-world apps out there that are also vulnerable to remote code execution." And I think is logic is exactly right. The developers, again, they're just going to take the sample code, which is obviously using the deserialize utils, and like tweak it, change the name to their company and whatever. So the flaw was assigned a CVE with a CVSS of 9.8. So that's meant to grab everyone's attention.

And yesterday, so Monday, VMware published security updates to remove the flaw from their Spring.io subsidiary's code. But as we also know, publishing the update is different from having it deployed on a server that's out in the field, and this is all just very fresh. So as I said, I expect in a couple weeks, much as we'll be talking about the exploitation of the browser and the browser flaw that was theoretical two weeks ago, not anymore. So I think the same thing will probably be happening here.

Between the initial discovery of the vulnerability and yesterday's patch publication, exploitation of the vulnerability where possible appears to have taken off at least as much for the CVE to get a 9.8. So it's considered to be of critical importance to anyone using this Spring framework. If you're responsible for it, or you know that your organization uses it, definitely go get VMware's update and fix this. The framework is of the MVC style, the Model View Controller approach, and also Spring WebFlux apps running on JDK 9 and later are vulnerable. So definitely worth doing.

Okay. We recently talked about the denial-of-service bug that Tavis Ormandy and Chrome's TLS guy together worked to discover in the OpenSSL library. Remember that that's the one which results in an infinite loop and essentially processor capture when processing client certs that have been deliberately manipulated to use specific elliptic curve crypto parameters.

Among the many companies whose products can likely be hung when they receive such a maliciously crafted client cert is the Taiwanese company QNAP, which last week revealed that a selected number of its network-attached storage appliances were in fact vulnerable to this OpenSSL problem. Last Tuesday their advisory said: "An infinite loop vulnerability in OpenSSL has been reported to affect certain QNAP NAS. If exploited, the vulnerability allows attackers to conduct denial-of-service attacks." So, you know, that's not the worst of all possible outcomes. Yes, your NAS goes down. But it doesn't go down in a way that lets any bad guys get in. So okay. It sort of sacrifices itself.

I have a list of the affected QTS and QuTS versions, but QNAP doesn't yet have any patches available anyway. The good news is, as I said, the only thing an attacker can do is to hang your NAS, which you then reboot, and it's back up until they hang it again. And they can only do that if TLS connections are being accepted from random IPs out on the public Internet. And everyone knows that's never a good idea; right?

QNAP keeps being somewhat of a mixed blessing. As I've looked through some of the back-and-forth conversations that surround its various problems, I see that its users generally love their devices, despite them having had more than their share of security troubles even just this year. I mean, we've been talking about QNAP vulnerabilities for years. QNAP is currently working to catch up to the OpenSSL DoS flaw which is only a couple weeks old, but they're also still working to patch that recent "Dirty Pipe" Linux kernel flaw from earlier in March, which also currently has no mitigation on QNAP's NAS devices.

The good news there is that at least it's only a local privilege escalation vulnerability. On the other hand, if you're in a big enterprise, and you're not able to trust all the people on the inside of your network, the fact that it's only local doesn't provide much comfort. And it's not QNAP's fault about the OpenSSL side, at least. Well, actually even this Dirty Pipe because both of these problems arise from the Linux kernel that it's built on. So everybody who was using the Linux kernel with those problems would have been subjected to these potential vulnerabilities.

But more than that, attackers have been pummeling QNAP devices all year with both ransomware and brute-force attacks to the point that the brute-force attacks prompted QNAP to urge its own customers to remove their Internet-exposed NAS devices from the Internet. In late January, QNAP forced out, pushed an unexpected and not entirely welcome update to its customers' NAS devices after warning them that the DeadBolt ransomware was mounting an offensive aimed at QNAP's users. So on top of everything else these users are targets. And just two weeks ago reports surfaced that DeadBolt, the DeadBolt ransomware, was at it again in a new wave of attacks against QNAP.

And last August, two vulnerabilities that could result in remote code execution and denial-of-service respectively prompted emergency patches by QNAP. Now, interestingly, this broader topic, and I think this is probably what made me think of it actually, of today's topic, the broader topic of the dangers of public port exposure, which QNAP perfectly evidences, serves as a perfect lead-in to today's discussion of Port Knocking, the idea being there is a way, and it's hosted on Linux, so QNAP could use it, of completely blinding the public Internet to the presence of open QNAP services, except to people who know the secret knock, which we'll be talking about later, both pros and cons, because port knocking has had some people saying, eh, it's just security through obscurity. I'm not sure that I buy that.

Okay. Sophos has a 9.8. Last week the cybersecurity firm Sophos warned that a recently patched critical security vulnerability - and that's the way you want to start these notices about a 9.8, it's recently patched, good, better than we don't have any fix for it yet - recently patched critical security vulnerability in its firewall product was now being actively exploited in real-world attacks. Now, that's not what you want to say; but that flaw CVE-2022-1040 sports, as I said, the attention-grabbing CVSS of 9.8 and impacts Sophos Firewall versions 18.5.3, also known as 18.5 MR3, and earlier.

And what you never want to hear is it's an authentication bypass vulnerability in the user portal and web admin interface. You could argue that the user portal needs to be open and running on the public side. Web admin, eh, I have to be convinced. But once again, there's a way to protect that. And still worse, when it's exploited, this particular 9.8, thus the number, allows a remote attacker to execute code of their choosing. And obviously, since it's under exploitation, the bad guys are aware of it. Sophos's security advisory said: "Sophos has observed this vulnerability being used to target a small set of specific organizations primarily in the South Asia region. We have informed each of these organizations directly." So clearly they've got some telemetry with their product which has allowed them to determine, whoops, this is happening, and nice of them to let their customers know.

And, you know, as for "doing it right," the flaw was addressed first in a hotfix that is automatically installed for customers who have the "Allow automatic installation of hotfixes" setting enabled. And I recognize we've talked about this a lot, that the whole issue of automatically pushing updates of security vulnerabilities is a bit controversial. I would argue it's becoming less controversial with time. But at least in this instance, having Sophos taking responsibility for and maintaining their firewalls to me sure seems like a good idea. Our operating systems are doing it now. Our mobile device platforms are doing it now. Pushing that out one level or layer to the firewall that is in front of the operating systems, that seems like a good idea, and especially when it's also in front of potentially a whole organization.

If it were mine, I'd be inclined to let Sophos autonomously maintain the firewall whose code they created in the first place. And I'm sure it's signed and authenticated, and there's no way for it to be spoofed. And yes, it's true. If they suffered a break-in, bad guys could potentially poison the source of the updates and push that out. So there's the downside of that. But on balance, probably a good idea. And not surprisingly, until the firewall is updated one way or the other, if a user goes and gets it themselves, they recommend that their users disable WAN access to the User Portal and the Web Admin interfaces. And I would wonder why. Certainly those should not be enabled unless they're absolutely needed.

And in a statement about both their integrity, their commitment to doing things right, and the severity of the issue, they have also provided updates to many earlier past-end-of-life versions of their firewalls and firmware. Sophos said: "Users of older versions of Sophos Firewall are required to upgrade to receive the latest protections and this fix." These things were past end-of-life, and they thought, okay, this is bad enough, we're just going to fix those. And as we'll be seeing, that's a choice that Wyze did not make with their cameras.

Okay. So last Thursday the U.S. CISA ordered federal civilian agencies to patch, not only that critical Sophos firewall bug we were just talking about, presumably if they don't already have automated update or auto update enabled, but in addition seven other vulnerabilities. And the federal civilian agencies have three weeks, or until April 21st, to get them all patched. And CISA says all eight of these vulnerabilities are under active exploitation. We know that the Sophos one is.

In addition to the Sophos problem, the CISA also ordered federal agencies to patch a high-severity arbitrary file upload - that doesn't sound good - vulnerability in the Trend Micro Apex Central product management console that can similarly be abused in remote code execution attacks. Two days earlier, Trend Micro said that it had observed "at least one active attempt of potential exploitation," which sounds a little bit like they're hedging, but I'm sure they know that this is actually happening. And the list of eight total "you must patch these" commandments include the need to patch a different QNAP NAS problem than the one we were just talking about, an improper authorization vulnerability which has been reported to affect QNAP NAS running HBS 3, which is their Hybrid Backup Sync. When exploited, that vulnerability allows remote attackers to log into a device. So I suppose it's no surprise that QNAP is being targeted as much as they are.

Anyway, when I was looking over this list, I did a double take, since two of the eight CVEs which CISA says are currently under active attack are dated 2018, and one is back from 2014. The two from 2018 are for Dasan GPON Routers. And I'm sure we talked about this back in 2018 because I remember the GPON stands for Gigabit Passive Optical Network routers.

Leo: Oh, GPON.

Steve: GPON. And they suffer from, again, a long-since-patched command injection vulnerability and authentication bypass vulnerability, two different problems, currently under attack, that have long since patched. But again, it's a router. So you can kind of see it like in a forgotten closet somewhere, just sitting there, doing its job, hosting all kinds of crypto currency miners and who knows what else.

Okay. The CVE from 2014 is a bit startling. 2014. It's CVE-2014-6324.

Leo: So the year gets assigned when it's discovered. Right? So it's been eight years.

Steve: Yes, eight years, this thing. And it's in use now. It's being, like...

Leo: Sure it is.

Steve: They're seeing people scanning for this thing.

Leo: It's an oldie but goodie, yeah.

Steve: Oh, boy. Its description in the National Vulnerabilities Database says: "The Kerberos Key Distribution Center (KDC) in Microsoft Windows Server 2003 SP2, Windows Vista SP2, Windows Server 2008 SP2 and 2008 R2 SP1, Windows 7 SP1, Windows 8, Windows 8.1, and Windows Server 2012 Gold and R2." It's there. It's in all those. "Allows remote authenticated domain users to obtain domain admin privileges using a forged signature in a ticket." And that sounds familiar to me. I'm sure we talked about it back then. And it's been exploited in the wild since November of 2014. It's known as the Kerberos Checksum Vulnerability."

So here we are, eight years later, and the U.S. CISA feels the need to explicitly tell federal civilian agencies that they now - eight years, but no, no. Now you have three weeks to get that patched. I'm sure all they needed was a gentle reminder. And oh, yeah, get right on that. Meant to do it yesterday, but thanks for the reminder. Wow.

Okay. Two weeks ago, when we talked about the browser-in-the-browser attack during podcast #863, it was all just theoretical. Remember that its penetration testing developer, "Mr.d0x," had simply produced a very convincing proof of concept. We showed in the show notes side by side a real OAuth popup authentication and his faked one. And they were the same. The same domain name which was spoofed, everything. So he demonstrated that using just HTML, CSS, and JavaScript, you could produce an actual lookalike multifactor authentication popup. And in today's world it took less than 10 days for that "Hey, that's a great idea" concept to become fully weaponized.

Last Thursday, a Belarusian threat actor known as Ghostwriter, also known as UNC1151, had been spotted leveraging this recently disclosed browser-in-the-browser technique as part of their credential phishing campaigns which are simultaneously exploiting the ongoing Russian invasion of Ukraine. As Mr.d0x demonstrated, this technique allows a legitimate domain and popup to be shown to an unsuspecting user.

Google's TAG team, their Threat Analysis Group, wrote in a posting last Wednesday that Ghostwriter was using Mr.d0x's browser-in-the-browser to siphon credentials entered by unsuspecting victims.

Leo: Well, of course. Who doesn't love Mr.d0x?

Steve: What could possibly go wrong? Let's just show this to the world because maybe the Belarusians are out of fresh ideas for how to phish people. So the TAG team said: "In early March, Google's Threat Analysis Group published an update on the cyber activity it was tracking with regard to the war in Ukraine. Since our last update, TAG has observed a continuously growing number of threat actors using the war as a lure in phishing and malware campaigns.

Leo: Oh, of course. God.

Steve: Of course. It always happens. Government-backed actors from China, Iran, North Korea, and Russia, as well as various unattributed groups, have used various Ukraine war-related themes in an effort to get targets to open malicious emails or click on malicious links. Financially motivated and criminal actors are also using current events as the means for targeting users. For example, they wrote: "One actor is impersonating military personnel to extort money for rescuing relatives in Ukraine." Wow.

Leo: That's just low. God.

Steve: Hey, would you like your mother to be rescued? I'm a Russian, and I found your Mom, and send me some money, and I'll bring her home. Wow.

Leo: Ugh.

Steve: TAG has also continued to observe multiple ransomware brokers continuing to operate in a business-as-usual sense. So it should be no surprise. Anytime anything of importance happens anywhere, the scum surface in an attempt to leverage the event to their advantage, whatever it might be.

In this case Google's group wrote that: "Ghostwriter actors have quickly adopted this new technique, combining it with a previously observed technique, hosting credential phishing landing pages on compromised sites." So here we have an example of a purely theoretical proof of concept being picked up within days of its publication and quickly being leveraged to significantly increase the effectiveness of a traditional phishing and logon campaign. And as we discussed at the time, Leo, two weeks ago, you look at it, and it's like, yeah, this says PayPal, very clear, P-A-Y-P-A-L. No typo. It's not Popal or anything else. You know, looks great. Click that link. What could possibly go wrong?

Leo: Yeah, yeah.

Steve: When we first talked about NPM supply chain attacks last week, the security firm JFrog had identified at the time a total of 218 malicious packages which were using a form of name collision to replace packages in the @azure namespace. By naming their malicious packages without any namespace designation, their packages might be obtained if a developer had not explicitly specified the @azure namespace as their target for their dependency.

At the time, and it turned out that was true, it wasn't a massive effect, but it was worrisome. And at the time JFrog had not identified the threat actor behind this NPM repository attack. Now, a week later, we know more. The threat actor is named "RED-LILI," R-E-D hyphen L-I-L-I. They've been linked to this ongoing large-scale supply chain campaign targeting the NPM repository, and have published nearly 800 malicious modules.

The Israeli security company Checkmarx said: "Customarily, attackers use an anonymous disposable NPM account from which they launch their attacks, as in one account. But this time, they said, the attacker has fully automated the process of NPM account creation and has opened dedicated accounts, one new account per package, thus making this new malicious package much more difficult" - these new malicious packages - "much more difficult to spot."

Checkmarx's findings build upon recent reports from, as we know, JFrog, but also Sonatype, which detailed hundreds of NPM packages which leveraged the dependency confusion typosquatting-style package replacement to target not only Azure, but also Uber and Airbnb developers.

According to a detailed analysis of RED-LILI's modus operandi, earliest evidence of anomalous activity was found to have occurred on February 23, with the cluster of malicious packages being published in "bursts" over a span of a week. Specifically, the automation process for uploading the rogue libraries to NPM, which Checkmarx described as now being a "factory," involves using a combination of custom Python code and web testing tools like Selenium to simulate user actions required for replicating the user creation process in the registry.

So in other words, an actual user goes through some processes to sign up for and acquire an NPM account. Well, nothing prevents all of that from being automated. This reminds me of the problems we were having initially over on our web forums, right, because you could have bots or actual users, and we have seen actual users creating accounts. So it's true they're not a robot when they click "I'm Not a Robot." They're telling the truth.

Bypassing the one-time password verification barrier put in place by NPM is no problem since NPM sends a one-time password to the email address the attacker's bot registers with. And what I've seen firsthand from bots registering on our web forums, they just create Gmail email accounts like there is no tomorrow. Typically a normal first name and then six or seven digits, which they just make up at random. Probably doesn't exist. Create the account, looks, I mean, it is a valid Gmail account. They then register under that account. The one-time verify your email goes there. They pick it up from there, plunk it into the web page. The whole thing is now automated. So one malicious package per account, thanks to automation.

Checkmarx researchers said: "As supply chain attackers improve their skills and make life harder for their defenders, this attack marks another milestone in their progress. By distributing the packages across multiple usernames, the attacker makes it harder for defenders to correlate and take them all down with 'one stroke' as had traditionally been possible."

So I read this as sort of the chickens coming home to roost. The NPM system never had super-tight security. And that was fine for a long time. But now it's not. And this is sort of, if we were to - if there was a recent theme, it would be things that were okay for a long time are no longer so. The lowest of the low-hanging fruit has been picked. Now attackers are looking around for other targets, and they're finding them. They're finding things that were not really deeply secured and going after those.

On the NPM side, its lack of tight security is finally becoming a problem for it. And the only way to combat this would be to impose much more stringent strictures on account creation and content publication. And like making somebody be a registered user for some length of time. The problem is I've seen that being bypassed. I've had over on GRC prior to us locking things down to a much greater degree, which we finally have, when I was getting rid of old accounts, there were all these bogus accounts that had been created that had never posted anything, presumably waiting for a time when they would come back later. And if there was some sort of a time, a minimal time somebody has to be a member before they're allowed to create content, they were just letting those clocks tick, waiting for the time that they would start posting spam under those accounts. So I don't know how you solve this problem. But it really is one.

FinFisher has been lurking around for years as one of the more successful and prevalent commercial spyware purveyors. Their product is called FinSpy. And the good news is that this Munich, Germany-based spyware company formally declared its insolvency last month amid an ongoing - not only "amid," but due to an ongoing and certainly unwelcome - to them, welcome to the rest of us - investigation into its business dealings.

They made the mistake, FinFisher did, of selling their premiere spyware product FinSpy to the Turkish government without having the legal documentation required to do so, after which their FinSpy system was used in a Turkish operation that preyed upon anti-government protestors. We talked about this at the time. Legal complaints filed by Reporters Without Borders, Netzpolitik, the Society for Civil Rights, and the European Center for Constitutional and Human Rights, all accused FinFisher of failing to abide by European export regulations including the requirement to obtain a permit granting trade to non-EU countries by the Federal Office of Economics and Export Control. FinSpy was created back in 2016 and has been linked to customers including the governments of Egypt, Bahrain, Bangladesh, Ethiopia, Oman, Saudi Arabia, and Venezuela.

According to the NGO's investigation, they said: "There are urgent indications that the Munich-based company conglomerate sold the spy software FinSpy to the Turkish government without the approval of the federal government, and thus contributed to the surveillance of opposition figures and journalists in Turkey." So about a year and a half ago, in October 2020, German authorities raided FinFisher's corporate offices, two associated businesses, and the residences of directors and executives, leading to the recent announcement that FinFisher accounts were seized and operations halted. So it's very likely the end of that operation. Not the end of all mobile spyware sales, unfortunately, but at least one fewer.

Recall that three weeks ago, during our QWACs On, QWACs Off episode, we mentioned the attack on Nvidia's networks and that the attackers subsequently exfiltrated about a terabyte of Nvidia's data, which paradoxically included some expired Nvidia driver signing certificates. Those certificates were then immediately used to sign malware, and I was puzzled at the time over how and why Windows would choose to honor drivers signed by certificates that were expired at the time of their signing. That still remains a mystery. The mystery that no longer remains regards a couple of the perpetrators behind that and apparently many other recent very high-profile attacks including the likes of Microsoft, Nvidia, Samsung, Okta, and Ubisoft, with many of them resulting in massive data leaks.

The group calls itself Lapsus$ spelled L-A-P-S-U-S with an additional trailing dollar sign appended. And despite the trailing dollar sign and their high-profile victim list, most Lapsus$ members are believed to be teenagers driven mainly by their goal - actually I don't think we can say "most" because we don't know how big the group is. But we've found...

Leo: Seven teenagers were arrested, yeah.

Steve: Yes, in addition to these most recent two, who were aged 16 and 17. They made the news last week when they appeared at the Highbury Corner youth court in London, charged with a number of cyber offenses. The names of both men, being minors, are being kept private, and both were released on bail. They've both been charged with three counts of unauthorized access with intent to impair operation of, or hinder access to, a computer; and two counts of fraud by false representation. Additionally, the 16 year old has also been charged with one count of causing a computer to perform a function to secure unauthorized access to a program. Which, you know, is gobbledy-gook for they're hackers, or legalese for they're hackers.

And the pair appears to be part of a larger group because also last week, as you said, Leo, the City of London police, which is leading the international investigation into Lapsus$, announced that it had arrested seven people, all between the ages of 16 and 21, in the U.K. alone.

In the U.S., our FBI is looking into the group's illegal activities and is seeking information concerning the Lapsus$ members involved in the compromise of computer networks belonging to multiple U.S.-based companies. The FBI said: "These unidentified individuals took credit for both the theft and dissemination of proprietary data that they claim to have illegally obtained. The FBI is seeking information regarding the identities of the individuals responsible for these cyber intrusions." So we've got an interagency issue. In the U.K., they're not disclosing the names of these individuals. The FBI is saying, well, thank you, we understand that, but we need to know. So I imagine that'll happen.

While it's still unclear how many active members the gang has and what roles each of them play, based on Telegram chats it's believed that they at least have affiliates, if not core members, located all over the world, speaking multiple languages including English, Russian, Turkish, German, and Portuguese.

Their bail and release was said to have "conditions." And I would bet that one of those conditions is an utter and total parental-enforced ban from any use of any Internet-connected devices while this case moves through the courts. If so, that might explain why around the time of the news of the arrests, Lapsus$ told its nearly 58,000 Telegram followers, among whom I'm sure is the FBI, that some of its members would be "taking a vacation."

But those recent arrests haven't put a damper on the larger group's activities because last week 70, seven zero, gigabytes of data belonging to the software services giant Globant were leaked on March 30th. Globant, whose headquarters are in Luxembourg, said they're currently conducting an exhaustive investigation and that it's "taking strict measures to prevent further incidents." I bet they are. Too bad they didn't do that beforehand.

Okay. I titled this "Not So Wyze." One week ago, last Tuesday, Bitdefender published the results of their close examination of the very popular Wyze family of security and surveillance-oriented Internet-connected webcams. And it will surprise no one to learn that they found problems, nor that the problems were extremely critical given the application these webcams are typically deployed for. Right? I mean, they're being sold as let's use this for security. And as I said at the top of the show, I utterly love the details, and our listeners will, too, and you will, Leo, of the authentication bypass that Bitdefender found, which I'll describe in a minute.

The most distressing part of the story, well, the equally distressing part of the story, is the fact that the Bitdefender group has been working with Wyze, or perhaps better stated "attempting to work with Wyze" for three years to get these three critical problems which they uncovered resolved. Back on March 6th of 2019, Bitdefender made first contact with Wyze and asked for a PGP key via their support form. You know, and as we know, that's standard practice now. You ask a vendor for a PGP key which will allow you to securely communicate with them, which involves the disclosure of potentially extremely sensitive details that they don't want exposed any more than the discoverer wants them exposed. No response.

They waited a week. On March 15th, 2019, three years ago, a little more now, Bitdefender made a second attempt at getting in touch with the vendor, still no response. Apparently unrelated, on April 22nd, Wyze released an update for Wyze Cam v2 to v4.9.4.37, which reduced the risk for unauthenticated access to the contents of the SD card that the camera might have. But still no contact with Bitdefender's research team. So this looks like it was just coincidental.

The next day, 4.10.3.50 was released for Wyze Cam Pan v1 with the same risk reduction for unauthenticated access to the contents of the SD card. So that looked like they did the same firmware update to a different product. That was April 23rd. A month goes by, and Bitdefender thinks, well, okay, let's reserve some CVE numbers for what we will eventually be publishing. So they did that. So that's May.

June, July, August, September, four months. And Wyze released Wyze Cam v2 that happened to fix one of the three CVEs that had been issued, but not the most critical one. So that was September 24th, 2019. Now we move to November 9th, 2020. And the vendor fixed a different one of the CVEs through an app update. The next day, finally, Wyze acknowledges the reception from a year and a half before and assigns an internal contact at Wyze to deal with Bitdefender. Two days later, Bitdefender sends the advisory to them and a proof of concept. Nine months pass. Silence.

On August 31st, 2021, Bitdefender follows up on patch progress. Hello. Is anybody there? September 13th, 2021, so two weeks from August 31st to September 13th. Bitdefender notifies the vendor. Oh, it actually probably was exactly two weeks. They waited. Nothing happened. So they said, okay, we're going to publish. Four and a half months pass, which brings us to January 29th, 2022. Wyze released firmware to fix the unauthenticated access to the contents of the SD card issue, which is one of the biggest problems. Okay. So that was on January 29th. Being again ridiculously responsible, Bitdefender waited 60 days from January 29th to March 29th. On March 29th, they published their report.

I've said it before, and I'm sure this won't be the last time I say it again: There is something fundamentally wrong with the idea, the way we have everything set up today, that an independent security research group must expend this level of effort to not only first reverse engineer and examine a product whose security is critically important to its users, but to then face an utterly unresponsive product publisher and attempt for three years to get them to fix critical flaws in the operation of their surveillance interconnected webcams.

And look at the Catch-22 that Bitdefender is then in. The only way to leverage responsibility from Wyze, to get Wyze to get off the dime, would be to go public with the news and the details of the flaws. But doing so would immediately place all of Wyze's gazillion webcam users at significant risk. And even if details were withheld, from like a partial disclosure by Bitdefender, we've all seen many instances where just telling the bad guys where to look for vulnerabilities is all that's necessary. Those wearing black hats could certainly follow in Bitdefender's footsteps. So Bitdefender had little true choice other than to wait and push and poke and prod and hope that Wyze would eventually open a responsible dialog. Again, they couldn't risk drawing any attention to the Wyze cams because other people could figure out how to exploit them. And the problems were really bad.

And what I loved, it's just rich that Wyze's cybersecurity team, like they have one, finally said they appreciated the responsible disclosure provided by Bitdefender on the vulnerabilities. Yeah, I bet they did. Three years Bitdefender patiently waited because of Bitdefender's ethics. Essentially Wyze had Bitdefender over a barrel.

Okay. So get a load of this truly amazing classic remote connection authentication bypass. It's just the best thing ever. When connecting remotely, a client is required to log onto the camera; right? The camera running a service, so we'll consider it to be the server. The client being a user on a web page or whatever. A client is required to log onto the device. Of course, because you don't want everyone to have access to your webcam, by definition. The client and the webcam share a 128-bit secret key. Okay, that's good security. Webcam has a 128-bit secret key burned into it. The client is required to know it, a pre-shared key. Good security. No problem there.

So the client initiates its connection by sending an IOCTL, an IO Control command, with the ID of its hex, 2710. Upon receiving, the cam will accept a TCP connection. Then the Wyze cam receives this packet with the ID 2710, which induces it to generate a random nonce value which it encrypts with its 128-bit shared secret key. Okay, that's great. It sends the encrypted blob to the client. By the design of this simple protocol, the client must have that same 128-bit shared secret key, which it uses to decrypt the camera's randomly chosen nonce value, which it had then encrypted, to authenticate itself to the camera, which it does by returning the properly decrypted camera nonce using an IOCTL command with the ID 2712 instead of 2710.

So 2710 initiates the handshake, asks the camera to generate a nonce, which encrypts the 128-bit shared secret, sends the encrypted blob back to the client. Client that has the same 128-bit shared secret key decrypts it and then returns it to the camera under the command 2712. The camera receiving the 2712 IOCTL compares the nonce that was hopefully decrypted by the connecting client to the value that it stored locally. And only if they match will the authentication succeed and the connection be accepted. And after that the client is free to do whatever it wishes with the camera. Right? No problem. Simple. Shared secret. Workable protocol.

Here's what the Bitdefender guys found. The way the Wyze firmware works is that upon receiving that initial 2710 command, it generates and stores the nonce for subsequent comparison. And it then encrypts it and sends it to the client. But if the client never sends the 2710 command in the first place, the nonce's value stored in RAM remains set to all zeroes. I just love this.

So all any attacker needs to do to gain full access to any original or only just patched just, what, earlier last month, or any unpatched cam, is to connect and skip issuing the first 2710 command which asks the camera to begin the authentication handshake. Instead, an attacker simply first sends the second 2712 command with an all-zeroes authentication. Since that will always match the camera's default null nonce, anyone can log into anyone's Wyze cam. You can see why Bitdefender said "Holy crap" three years ago.

Leo: Anyone would. But you do have to physical access. You have to be on the WiFi; right? You can't do this from the Internet. Or can you?

Steve: No. No no no no. It is a network attack.

Leo: Okay.

Steve: So the camera needs to be exposed. You have to be able to connect to the camera.

Leo: Right.

Steve: So if it's behind a NAT router, you wouldn't be able to. But if there were some reason that somebody had put their Wyze cam on the Internet, then anybody can access it.

Leo: Wow. Okay.

Steve: Bitdefender wrote: "After authentication, we can fully control the device, including motion control, pan and tilt; disabling recording to SD; turning the camera on or off, among other things. We cannot view the live audio and video feed, though" - get this - "because it is encrypted under that same still unknown to a remote attacker shared private key. However," they wrote, "we can bypass this restriction by daisy-chaining a stack buffer overflow which leads to remote code execution as detailed in Part 2."

They said: "For the stack buffer overflow, when processing IOCTL with ID 2776, the device does not check whether the" - you're not going to believe this, Leo - "whether the destination buffer is long enough before copying the contents onto the stack."

Leo: Well, there you go.

Steve: Uh-huh. "Exploiting this vulnerability is straight-forward. Through the IOCTL with ID 2776, we can set which servers to use to connect to the cloud. This seems to be a debugging function that allows the selection of production, beta, or internal API servers. When sending a request, we specify the length of the buffer in the first byte, then the buffer itself." They said: "This content is then copied onto the stack into a fixed 40 hex, which is 64 bytes, length buffer. Even though the specified size in the first byte is taken as a signed INT" - okay, now a signed INT that is a byte will have a maximum size of 7F because the signed bit that we were talking about a few weeks ago, that's got to be off for the signed INT to have a positive value. Still, 7F is 127, so that's enough to overwrite a 64-byte buffer and allow the stack to be overwritten and run the attacker-provided code.

The third and final flaw they found is unauthenticated access to the contents of the SD card: "When inserting an SD card into the camera," they said, "the contents of the SD card, including the recordings, can be accessed via the web server listening on port 80 without authentication. This is enabled by the fact that, after an SD card is inserted, a symlink to the card mount directory is automatically created in the www directory, which is served by the web server. The card contents can be viewed through the hello.cgi functionality located at /cgi-bin/hello.cgi; then the files can be downloaded through the /SDPath/path.

The SD card also holds the camera's log files. Before writing them to the card, the device XORs the content with a hex 90" - like why? - "not very strong protection. These log files can contain sensitive info such as the unique ID and the shared private key, which can then be used to connect remotely," and view the stream in real-time because now we're able to decrypt the stream, having obtained the shared private key.

The good news here, such as it is, and it's not much, is that the second- and third-generation Wyze cams can be updated to cure these various problems. The bad news is that the first-generation cameras have been abandoned by Wyze, and Wyze has said that they do not plan to support or update them in the future. The only thing we can hope for,

for anyone who has early first-gen Wyze cameras, is that maybe the press that this is now, finally, after three years, generating with the negative publicity of having critically broken and trivial-to-hack first-generation webcams might cause Wyze to change their minds. I mean, unless it's burned into actual ROM, there is no reason, or maybe they have no - they didn't provide a means for updating the firmware. I can't explain. They're just...

Leo: They say there's not enough RAM to update the firmware, not enough memory. That seems hard to believe.

Steve: Okay. It does. It seems more likely that they're just like, well, those are too old.

Leo: Yeah. They were only 20 bucks; right? I mean, you know...

Steve: It was an amazing little camera for the price.

Leo: Right.

Steve: But Wyze, you know, and this is what we see, right, with bottom of the barrel IoT vendors that just want to sell their stuff and not be bothered with security. Unfortunately, they tried to have it secure, but nobody audited their stuff; right? It's all proprietary. It's just trust us, you know, we got this. It's, oh, 128-bit encryption. Military grade. Can't get in there. Yes. Just drop the first half of the handshake and do the latter half, and handshake with all zeroes, and you're in.

Leo: Wow.

Steve: Yeah. Two tweets from listeners, from someone whose name is The Nargles. His Twitter handle is @WithTheNargles. He said: "Thanks for your recommendation. I just finished the first Bobiverse book. All I can say is, thanks for the recommendation. It was a lot of fun." He said: "And Ray Porter's narration," he said, "particularly of GUPPI, is spot-on."

Leo: Yeah, you're missing out on that. It's pretty good. He does GUPPI as Admiral Akbar. So he's always talking like this. It's great. It's really funny.

Steve: That would be great.

Leo: Yeah, Ray Porter's really a talent, and he kind of brings a Martian, Andy Weir's "The Martian" style to it, which is great.

Steve: Nice. Scott Cleveland tweeted: "@SGgrc A few weeks ago you and Leo were talking about the Bobiverse 'We Are Legion (We Are Bob)' books." And he said: "Thank you!" He said: "It's so hard to narrow down when scrolling through Audible what I will like. Your suggestions are pretty much a spot-on automatic win for me." So thank you, Scott. And then it occurred to me, if the quality of my recommendations is to hold, I should probably note that Book No. 4 is noticeably dragging.

Leo: Yeah.

Steve: It's still okay, but Dennis appears to be running out of new ideas for his Bobs. He's reusing the ideas he already has, and I have to say he's built an extremely interesting and clever Bobiverse using subspace com links and virtual reality in clever and, given a suspension of disbelief, feasible ways within this universe. But perhaps it should have been left at a trilogy.

Leo: Yeah. I'm halfway through Book 2. Maybe I'll just stop at 3.

Steve: Yeah, I think you should. But, you know, still there is more fun stuff happening. Oh, have you run across the Others yet?

Leo: Yes, yes. The Others exist. In fact he's just now encountering them for real. So I'm excited. It's getting exciting.

Steve: Yes. They are really, really, really, I mean, they're, I mean, bad.

Leo: Yeah. Yeah.

Steve: And seeing how they get taken care of is worth finishing the trilogy.

Leo: Peter F. Hamilton's aliens from, is it "Fallen Dragon," I can't remember which one.

Steve: "Pandora's Star."

Leo: Maybe it's "Pandora's Star," the ones that just kind of, yeah, they're great. They look like little...

Steve: The motiles.

Leo: The motiles, yeah, yeah. That's my favorite evil alien.

Steve: Oh, and they were non-biological. And they were quadrilaterally symmetrical; right? So they had like four feet, and they were sort of - kind of like Daleks except bad.

Leo: Quick follow-up from the chat room. Would it be safe to use a Wyze cam v1 behind a firewall?

Steve: I think so, yeah.

Leo: Yeah, because it needs access from the outside world.

Steve: Yes. The threat model is that you might have mapped a port through it so that you had access to the camera directly, remotely. And unfortunately that means other people could, too.

Leo: Thanks.

Steve: And of course that's another reason why our topic for the day, port knocking, will be of I think great interest to some of our listeners.

Leo: Yes, indeed.

Steve: David Lemire, he said: "Hi, Steve. I recently bumped into the author of NoScript on Twitter," he said, "when I mentioned I'd abandoned it long ago. He encouraged me to take a fresh look. So I found the website." He said: "Looking at the usage page, it does appear that the program has been updated/adjusted to the realities of the modern web." And he said: "I know you also gave up on it long ago, but this was interesting enough I thought maybe it was time for a revisit of NoScript." He said: "And no, I haven't actually tried playing with it myself yet, or I'd include my experience here."

Okay. So David, thank you. And I wanted to share that little tidbit with our listeners. I appreciate knowing that NoScript hasn't thrown in the towel yet, despite its name. And it occurred to me that the author is probably suffering the same dilemma I am with SpinRite. Renaming his program "SomeScript" really doesn't pack the same punch.

Leo: No. Well, it's some scripts.

Steve: It's SomeScript because you need some script. But you can't do NoScript. Our development group's discoveries with SpinRite strongly indicate that SpinRite's future with solid-state mass storage is guaranteed, perhaps maybe even more so than it ever was with spinning media. But I can't change the name, even though someday nothing will be spinning anymore. It's still going to be SpinRite.

Leo: On we go with a little knocking, with Steve Gibson.

Steve: Don't knock it. Don't knock it. Okay. So our listeners have heard me over and over and over lament the dangers of having exposed ports on the public Internet. There's the problem of a recognizable server or service that is in some way protected, but with a password which can be brute-forced in the background over time. Or, I mean, equally problematic is a service that has strong authentication, but a bug in the service itself. A perfect example is this OpenSSL bug. There's nothing that you need to authenticate about establishing a TLS connection. The problem there is a bug in the underlying service itself. And if the access to the port is completely unrestricted, then that means an incoming packet from any of 4.3 billion IPs is treated just like any other.

So the solution to this is firewall rules. And I have three locations, and I've got static links running in a triangle configuration between all three using strong firewall rules. I have the advantage that GRC is a set of fixed IPs. They will never change. But a cable modem rarely changes. I mean, you have to be offline or unplugged for day. And then when you reconnect you may get a different IP. You probably will. But, I mean, I'll go years often with no IP change. And so the key is that every endpoint knows its IP and knows the IPs of the other endpoints it trusts and selectively allows packets on specific ports only from those IPs. And when they're TCP connections, because of the need for a three-way handshake, IPs cannot be spoofed, as we know. UDP spoofable; TCP not.

And of course we've talked about being stealth, you know, GRC's ShieldsUP! service likes the idea that your firewall, your router is not even saying to a requested connection no, thank you, there's no port here, go away. Instead, it just drops the packet. In this day and age, technically by the original formal protocol rules of the Internet, you should respond by saying hi, I got your packet, but you should know there's no service running here. Well, unfortunately that provides information out of a sea of IPs that there may not be a service there, that service. But there's something there, maybe at a different port. So better just to let the packet die in a modern Internet.

So imagine that you want to make a service available or services available to an IP which is not static and not previously knowable, but could be anything, yet you simultaneously want all other services or that service at all other IPs not to be available. Well, since the way you enforce allowing a specific IP to connect to a specific port is with a firewall rule that permits packets in identifying themselves with a source IP and a destination port which is specified, what you want is essentially a means of on-the-fly changing a firewall rule to permit a specific client anywhere on the public Internet to get in.

And there are several ways to do this. Generically those are known as port knocking. And the original old-school port knocking was very clever. The idea is that in the machine with the firewall which is publicly exposed to the Internet, you run a service. And in Linux it's known as "knockd," K-N-O-C-K-D. And it's available. Linuxes have it. It's not that widely used, which is one of the reasons I wanted to talk about it today. It's there, and it is cool. It runs monitoring the interface itself below the level of the TCP/IP stack. And you have to have libpcap installed in order to allow it to open a connection to the raw interface. This service is script driven. It takes a config file that tells it what to do when it sees different things.

So the idea is that when packets come down the wire to your IP and hit the machine, if they're unsolicited from some random IP on some port that you don't have open, they just die. They hit the machine, they die. But the point is they cross the NIC to the guts of the computer where libpcap and this knockd daemon are able to see them. So imagine if this firewall, this machine, had a secret knock sequence, which is to say send a packet to port 10192. Then send a packet to 10234. Then send a packet to 32769. Then send a packet to 50743.

The point is you can create an arbitrary long sequence which has to be specified in the proper sequence to create an unlocking sequence which the knockd daemon will recognize because it's watching all the incoming traffic to your IP. And if it sees a sequence of the proper packets all coming from one specific public IP, it then, using its config file, emits a command to your firewall, IP tables or whatever, it supports all of the different firewalls, to selectively open a port to the IP that generated this correct knocking sequence.

And what you now have is a means of having services which are publicly available, but absolutely non-existent. It's also the case that your use of port knocking is invisible, unseen, unknowable. Unless you tell people, there's no way for anyone to know that you're running a port knocker on your side, and that by sending a specific sequence of packets would have any effect. And the good news is it's not very common. So people aren't expecting you to do it.

The other piece of good news is this is actually pretty strong security. There are some problems. We'll talk about that. But on the pro side of this, we know that port numbers are 16 bits. So that means that a randomly chosen port carries 16 bits of entropy, essentially. Think of it as 16 bits of password. That means that four randomly selected ports, each carrying 16 bits, will give you 64. Or eight randomly selected ports gives you 128 bits. Okay. Eight, and 128 bits. There are people on the Internet who say, well, port knocking is security through obscurity. I would disagree. So is a password. Nobody knows what the password is. It's obscure. Right, it's a secret. Well, so is the proper port knocking sequence a secret. The biggest problem with it has been solved in its evolution.

But I like this just for its clarity and its simplicity. The biggest problem is, if somebody were able to somehow arrange to sniff the traffic between the client sending the packets and the server receiving them, so at either end or somewhere in between, there is no prevention for a replay attack. So standard old-school port knocking is not safe against replay attacks. On the other hand, I'm not suggesting that this be the only security that your system would have. For example, I'm not suggesting that after providing the knocking sequence, the server you connect to doesn't have still its own security. Again, multi-layers of security are good.

This is another really intriguing and useful layer because it is able to hide the fact that there is a server accepting TCP connections. Without port knocking, that server will accept a TCP connection from anyone because it might have to accept a connection from anyone. That tells the bad guys there's a server there listening on that port, and they can go to town. If you put up another layer around your system, a port knocker, you look like every single port on your machine is stealth. You know different.

The other kind of cool thing about port knocking is that it doesn't take a sophisticated client to be able to generate these. I've seen an example, for example, where telnet, trying to initiate a connection, will send three SYN packets to an IP that doesn't respond. So you could set up the knocker daemon to look for three SYN packets on the first port, then three SYN packets on a second port, and so on. That would mean that you could just use a brain-dead telnet which you ask to connect to eight successive ports which it will ultimately fail. You know that each of those generated three connection attempts. So you've ended up sending a total of 24 packets.

And now the knocker daemon is satisfied. It sends a command to IP tables, opens up a rule, adds a rule to only accept incoming connections to the destination that you specified for that knocking sequence from whatever IP you're at. So even with the knock completed, bad guys still can't see that you have anything open because it is only open for the IP which was the source of the knock packets. Anyway, my point is it is such a clever and cool idea that I wanted to share the concept with everyone. And there are knock generators, knocking clients which will do a much more clean job of establishing connections, depending upon what kind of client you have. Okay. But I said that the problem was replay attacks.

Leo: Right.

Steve: The evolution of this which has occurred is known as single-packet authorization. And what we're missing from port knocking is that all we're taking advantage of is the fact of a packet hitting the firewall, not its contents. Which means we're missing a huge opportunity. The cleverness of it is that it uses just the fact of the packet's arrival. But if we want to step up our game, we do it with what's known as single-packet authorization. There is a tool, FWKNOP. And that stands for Fire Wall Knock Operator. The guy behind it took this to the next level. And again, it's on GitHub. All of this is free. All of it's open source. And it has had, over time, it's been scrutinized to death.

Single packet authorization takes the IP of the source, that is, the IP of the client. It encrypts it with a public key which the user knows, or private key, or both, or symmetric key, and uses an HMAC in order to authenticate the result. And it sends one packet to a pre-determined closed port on the destination. The agent which is listening there gets the packet, uses its matching secret or its private key. If you want to use asymmetric encryption, public key encryption is also supported. It authenticates the packet. It decrypts the packet. It verifies that the IP that was contained in that envelope is the source IP from which the packet came. And only if all of that works, it then does whatever its been configured to do, which could be anything. So now we have fully stealth cryptographically secure single-packet authentication which can be used to do lots of things.

Oh, I forgot to mention that some of the cool things that the behind-the-scenes scripts can do is, for example, it could open a port, and also send a Wake-on-LAN packet to a server on the LAN, causing that machine to power up like by command in order to then provide services for whatever the port was that was opened. So it's all configurable.

This FWKNOP, I've got links in the show notes. Its founder is at Cipherdyne, C-I-P-H-E-R-D-Y-N-E, dot org, Cipherdyne.org. And to go directly to the page is /fwknop. There are clients for Fedora, Red Hat Linux, CentOS, Debian, Ubuntu, OpenWRT, FreeBSD, macOS, OpenBSD, iPhone, Android, Cygwin, and Windows; servers for all of the OS platforms and the desktop platforms except for the mobile clients. Doesn't really make sense for obviously a mobile client to have a server. GnuPG support, HMAC support, client NAT penetration support, server-side NAT support.

Anyway, it is a beautiful complete win for anyone having a need that this particular approach solves. And again, in this day and age, where we've got people brute-forcing servers that are sitting exposed to all IPs when they don't need to be, I kind of wanted to remind everybody, this thing's been around since the early 2000s. I mean, the concept has. It's just it's very clever in terms of allowing authenticated otherwise stealth access to servers operating at arbitrary ports. And that's port knocking.

Leo: Neat. Very neat. You can have a rolling port knock, I guess, like a TOTP, sort of.

Steve: I don't think you really need one.

Leo: You don't need it. I'm just thinking of the replay attack issue.

Steve: Yes, very good point. And the way this guy solves the problem, he has solved it, is it records a log of all the previous packets that have authenticated, and it will never allow the same one to be used a second time.

Leo: Perfect, perfect, yeah.

Steve: And it would have to be the same IP, after all.

Leo: Right.

Steve: So but he actually did think about the replay problem, and he does it just by logging successes. He logs a small hash of a success, so the log isn't big. And if the packet, it first has to match all the other proper criteria, which means it could only be a replay. And if it is, if it matches all the criteria, then it checks to make sure it's never seen that before, so you don't end up with a big log in any event.

Leo: Steve, you've done it again. Another great, thrilling, gripping edition of Security Now! for all of our listeners. Steve lives at GRC.com, the Gibson Research Corporation. That's where you'll find SpinRite, the world's finest mass storage maintenance and recovery utility. Version 6 is current; 6.1 is imminent. If you buy 6 now, you'll get 6.1 automatically for free. You can also participate in the development, if you want. GRC.com.

While you're there, check out all the freebies, the forums, ShieldsUP!, all the utilities Steve writes, like InControl. It's all there, GRC.com, along with this show. He hosts a couple of unique versions of the show on his website, a 16Kb audio version for the bandwidth-impaired, and he also has a transcript carefully crafted by humans, well, a human named Elaine.

Steve: A loving human.

Leo: A lovely human. And you can read along as you listen or search to find parts of the show and so forth. That's all at GRC.com. He also has a 64Kb audio version, as we do. We have video as well at TWiT.tv/sn. There's a YouTube channel dedicated to Security Now!. You can watch every show there, all 865 of them. And, well, they're not all video. So maybe all 810 of them or whatever it is. I don't remember when we started video. And then of course you can subscribe in your favorite podcast client. You'll get it automatically that way. And if your client allows reviews, please leave us a five-star review. Share the good news about Security Now!.

If you want to watch us do the show live, we do it Tuesdays right after MacBreak Weekly. That's usually sometime between 1:30 and 2:00 p.m. Pacific, 4:30 and 5:00 p.m. Eastern, 20:30 UTC at live.twit.tv. You can chat with us live. We still use IRC, yes, we do, at irc.twit.tv. But if you're a more modern type, and you kind of prefer to do the Discord thing, Club TWiT members get access to a wonderful Discord, which is not just about the show, but about every other aspect of geek life, including coding and beer and wine and cocktails and ham radio.