A BGP Routing Attack

Description: This week we talk about another WordPress plug-in mess, this one so bad that WordPress themselves force-installed updates on more than three million sites. We look at the new Xenomorph Android malware and at a mistake made by a new and prominent ransomware service. We examine why blurring or pixelating text for redaction was never a good idea, and what can go wrong with a plan to shut off one's teenagers' Internet access at home.

High quality (64 kbps) mp3 audio file URL: http://media.GRC.com/sn/SN-859.mp3
Quarter size (16 kbps) mp3 audio file URL: http://media.GRC.com/sn/sn-859-lq.mp3

We unfortunately need to revisit the supercritical Magento/Adobe Commerce platform patch which didn't quite work completely the first time, and we consider the implications of the technology behind last week's denial-of-service attacks on some of Ukraine's critical infrastructure. Then, after quick sci-fi and SpinRite updates, we'll take a look at an effective and lucrative attack that was perpetrated by deliberately abusing the still-too-trusting Border Gateway Protocol.

SHOW TEASE: It's time for Security Now!. Steve Gibson is here. Yes, another nightmarish flaw in a WordPress plugin. Steve will give you the ins and outs on that. A new alien life form, Xenomorph. Well, it comes from hackers. And we'll talk about a BGP routing attack, this one not an error, a malicious attack, intentionally malicious attack. Plus the ransomware gang that couldn't shoot straight. Well, at least it couldn't encrypt straight. It's all coming up next on Security Now!.

Leo Laporte: This is Security Now! with Steve Gibson, Episode 859, recorded Tuesday, February 22nd, 2022: A BGP Routing Attack.

It's time for Security Now! for February 22nd, 2022. Here's Steve Gibson, our man in charge. Hi, Steve.

Steve Gibson: Hey, Leo. Great to be with you for this monumental moment in time. What is it, about a little less than half an hour from now it will be 2:22 on 2-22-22.

Leo: 22222. Wow.

Steve: Which, yeah. So...

Leo: So there.

Steve: So there.

Leo: So there.

Steve: We're going to talk about what I promised last week we would talk about because there's actually all kinds of fun stuff to talk about this week. But this BGP routing attack is so interesting and sobering that - and perfect for this podcast because it's going to be a little bit of a, I would say a stem winder, but more like a prop winder for our propeller beanies.

But first we're going to talk about another WordPress plugin mess, this one so bad that WordPress themselves force-installed updates on more than three million sites which were affected. We're going to look at the new Xenomorph Android malware and at a mistake made by a new and prominent ransomware service. We examine why blurring or pixelating text for redaction was never a good idea. And if you wanted to cheat, and you're listening live, you could go - that's the shortcut of the week, grc.sc/859. That'll give you a little teaser to where we're headed.

We're also going to look at what can go wrong with a plan to shut off one's teenager's Internet access at home, how not to do it. We unfortunately need to revisit the supercritical Magento/Adobe commerce platform patch which didn't quite work completely, unfortunately, the first time. And we consider the implications of the technology behind last week's denial of service attacks on some of Ukraine's critical infrastructure. Then, after a quick sci-fi and SpinRite update - actually, fortunately, those are separate topics, SpinRite will not be science fiction for much longer - we'll take a look at an effective and lucrative attack that was perpetrated by deliberately abusing the unfortunately still-too-trusting Border Gateway Protocol.

Leo: All right.

Steve: So there.

Leo: Picture of the Week coming up with the very first rule of programming. Wait a minute, that was last week. I'm still looking at Inept Panda. No wonder I'm confused. Trying to find all this stuff. I've got the wrong show notes. I'll get the right ones in a second. But first...

Steve: Ah, that's the problem.

Leo: That's the problem.

Steve: I need to acknowledge all of our listeners who have sent me the cartoon that will be seen next week. I already had this one set up and in the show notes and canned and done when everyone, because it's assembly language themed...

Leo: Oh, nice.

Steve: ...sent me the same cartoon. So I thank you. I wanted to recognize - and for all of you through the next week who would be doing so, thank you in advance. I got the message.

Leo: Don't send me the cartoon.

Steve: I've already captured the cartoon. We'll all be laughing at it next week.

Leo: This is funny. Your reputation, you are known as THE assembly language user. I mean, you're the last one. It's hysterical.

Steve: Yeah, yeah.

Leo: Yeah. This one is a Jurassic Park cartoon.

Steve: This is. And we have two frames. And I've had this for a while. I just thought it was kind of cute. In the first frame we see the character who plays John Hammond, who of course is the millionaire, multimillionaire founder and evangelist of Jurassic Park. And it's one of my favorite movies. Anyone who's seen it knows how enthusiastic John is about this creation of his. And he's, like, they're tasting the ice cream, and someone says, "Oh, this is really good ice cream." And John says, "We spared no expense." And so this "spared no expense" thing is like a theme of his throughout, like oh, these cushions are really soft that we're sitting on. "Spared no expense."

Anyway, the second frame of this has this fabulous programmer guy who's got his Sun workstation, and his workstation unfortunately looks like mine, littered with stuff and candy bar wrappers and multiple old-school mechanical clanky keyboards. And so the first frame, "Spared no expense." The second frame is the response to that, saying "Hires only one computer programmer to run the whole park full of man-eating dinosaurs."

Leo: Newman? He is, by the way, he is so good in that movie. It's so funny.

Steve: Yeah, he's just perfect.

Leo: Such a weaselly fellow, yeah.

Steve: Yeah, exactly. Yeah, you do not want your park under his thumb.

Leo: Run by him, no.

Steve: No. Okay. So a three-year-old flaw has been found in the UpdraftPlus WordPress plugin, which is in active use at more than three million sites, including those at

organizations including Microsoft, Cisco, and NASA. It's been assigned, that is, the flaw, assigned CVE-2022-0633 and given a high common vulnerability severity score of 8.5. It can be readily weaponized to download a site's private data using an account on the vulnerable sites. So the good news is you have to have an account of some form on the site. So sites that allow people to create accounts, that's going to be a problem. It was mostly seen as an insider threat because what this allowed you to do, when you talk about the site's private data, it's the database that backs up the site, so not just the way the pages look, but if it's an ecommerce site, all the credit card data, all the personal identifiable information, I mean, blah blah blah.

Anyway, this upset, as I said, WordPress enough that they didn't waste any time. This UpdraftPlus is a backup and restoration solution that's capable of performing full manual or scheduled backups for WordPress files, the databases, the plugins, and themes, which can then be reinstated through the WordPress admin dashboard that you're able to roll back to a previous saved state. But as a consequence of this defect that was found by a security researcher at Automattic, which of course is WordPress's parent company, as I said, any logged-in user to a WordPress installation that has UpdraftPlus installed would have been able to, until this was fixed, obtain the privilege of downloading an existing backup, which should have been restricted to admin users only.

So basically this was an admin rights bypass. And since backups can contain all of a site's data, passwords and other confidential data could be obtained. Moreover, in some cases, site takeover is possible if an attacker is able to obtain database credentials from a configuration file and successfully access the site's underlying database.

The fundamental flaw in this case was the mechanism by which UpdraftPlus validated who was requesting backups. The attack starts by sending what's known as a "heartbeat request" containing a data parameter to obtain information about the most recent backup. Once the attacker has that information, they're able to trigger the "send backup via email" function after manipulating the endpoint request a bit. This function is the one that's supposed to be restricted strictly to administrators. But due to a missing permissions check, anyone with an account on the target site can access it without limits.

So the only good news here, as I said, is that an attacker would need to have an account of some kind on the site. So that does minimize the risk a bit. However, the popularity of UpdraftPlus, coupled with the simplicity of the attack, makes for a potent combination.

In fact, it was so potent that, immediately after learning of the trouble, last Wednesday WordPress themselves took the nearly unprecedented step of ignoring individual site admin preferences and force-updating all affected sites. A count of a log showed that 783,000 sites were updated by WordPress last Wednesday the 16th, and an additional 1.7 million force-updated the next day, on the 17th.

In an advisory just published, the maintainers of the plugin said: "All versions of UpdraftPlus from March 2019 onwards have contained a vulnerability caused by a missing permissions-level check, allowing untrusted users access to backups." The issue impacts UpdraftPlus versions from 1.16.7 to 1.22.2. So all three million-plus sites using UpdraftPlus are recommended to update to 1.22.3, or in the case of using the Premium version, 2.22.3, in order to mitigate this potential exploit. And actually since then another problem was fixed related to printing auto-backup options under PHP 8. So actually the latest version as of last Thursday was actually 1.22.4.

Anyway, if by any chance you are a user of Updraft Plus who isn't - Leo and I were talking about this a little bit before. There are some sites that WordPress may not have access to, may not be in the maintenance loop for. So you definitely want to make sure, you know, again, if nobody has an account, then okay. But why not update when you know that there's a potential flaw? That's the kind of thing, too, where maybe you

thought, oh, no problem, I'm the only accountholder on the site. And then this all, you know, some time passes, and there's some reason to give somebody else some rights on the site, and whoops, there's a flaw that could have been abused that could have been taken care of a long time ago. So yeah, just in general, stay up to date.

Okay, now, for this next one, I'll admit that this bit of news grabbed my attention only because of my love of science fiction, and the name given to the malware was Xenomorph. So the problem is, for us, is that talking about malware on the Android platform runs the same risk we felt in recent years if this podcast wanted to talk about ransomware. The question is, if we're going to start talking about it, where do we stop? And it's not clear to me that this is all Android's fault anyway, with Android-based handsets commanding 87% of the global market, versus Apple's remaining 13. There are nearly seven times more Android users than Apple users.

And I think the nature of an Android user, I mean, there are just so many of them, may be a little less cautious than Apple's users because a lot of these exploits do require someone who's not really focused on security. But in any event, seven times the opportunity to trick someone into downloading something that hasn't yet been flagged as malicious and removed from the Google Play Store. Now, that said, the increased level of freedom that Android's apps and users demand and enjoy over Apple's platform, of course that's what we always hear about Android people, oh, yeah, I'd like to be able to use whatever launcher I want, and I don't want all these constraints that iOS puts on me.

So yeah, with that freedom comes additional risk, inherently. And it creates an environment that is unfortunately more comfortable for malware. But since we're talking here, now, about Xenomorph, it's worth touching upon it just as a reminder to the Android-using contingent of our listeners. The point is, we're going to talk about Xenomorph, but this is sort of generically true of Android in general.

For the past 10 years, the Amsterdam, Netherlands-based security research company ThreatFabric has been focused upon banking malware. Yesterday, just Monday, yesterday they published the first report of their discovery of a new malware family they named "Xenomorph." And I'm going to share just the top of their report, and then I'll summarize the rest because they sort of start off giving us a good grounding.

They said: "In February of 2022" - meaning this month - "ThreatFabric came across a new Android banking trojan, which," they said, "we dubbed Xenomorph. The name comes from its clear ties with another infamous banking trojan, Alien, from which Xenomorph adopts class names and interesting strings." In other words, clearly there's some relationship between the two.

They said: "Based on the intelligence gathered, users of 56 different European banks are among the targets of this new Android malware trojan, distributed on the official Google Play Store, with more than 50,000 installations," of this one particular app that contains this trojan.

Anyway, I'll finish quoting them with them saying: "Just like the monster protagonist of the famous Ridley Scott franchise, this malware shares some aspects with its predecessor. However, despite its obvious ties to one of the most widespread malware of the last two years" - that is, the Alien predecessor - "Xenomorph is radically different from Alien in functionality. This fact, in addition to the presence of not-yet-implemented features and the large amount of logging present on the malware, may suggest that this malware might be the in-progress new project of either the actors responsible for the original and quite pervasive Alien malware, or at least of someone familiar with its code base. However," they said, "this is only speculation at the moment."

Okay. So Google has been doing what is, I am sure, their best to keep their Play Store clean and free of malware. But it's a sheer numbers game. The Google Play Store currently contains 3.48 million individual app titles - 3.48. Now, are a bunch of those 3.48 million malicious? You betcha. Every day an average of 3,739 new apps are added to the store, 3,739 per day. Are some of those malicious? Yeah. It's going to happen.

So a case in point which put ThreatFabric onto the malware they came to name Xenomorph was a Google Play Store offering called Fast Cleaner. It's a utility aimed at speeding up device by removing unused clutter and removing battery optimization blocks. The app itself seems successful, with more than 50,000 installations reported on Google Play. It shows it was updated, if memory serves, on February 6th. So it's not clear if it got malicious at that point. Some of these things use the update process to sneak their bad conduct in, where they start off being benign, but then don't stay that way. Or maybe that was the first time this thing appeared. In any event, 50,000 installations.

But upon their analysis, the ThreatFabric folks recognized that this application belonged to the so-called "Gymdrop," G-Y-M drop, malware dropper family. They discovered and named "Gymdrop" in November of 2021 when it was observed deploying a payload of Alien.A malware. And from the configuration downloaded by the dropper, they found lurking inside - I'm sorry, they found it lurking inside of that Fast Cleaner app, which was purporting to be just a generic Android make-your-phone-better utility. Which, you know, is a way a lot of these things get in.

ThreatFabric was able to confirm that this dropper family continues to infect with the Alien malware family as its payload. However, the server hosting the malicious code which gets dropped into the unsuspecting client also contained two other malware families, which might be returned instead of Alien, depending upon specific drop triggers. One of those two others they were already well familiar with, something that they refer to as ExobotCompact.D. But the other was brand new, this one. Thus it got the name Xenomorph.

So this is what they had to say about Xenomorph. They said: "This Android banking malware is heavily under development and mostly supports the minimum list of features required for a modern Android banking trojan. Its main attack vector is the use of the accessibility feature overlay attack" - which we'll talk about in a second, and they will - "to steal credentials. Combined with the use of SMS and notification interception to log and use potential two-factor authentication tokens. The accessibility engine powering this malware, together with the infrastructure and command-and-control protocol, are carefully designed to be scalable and updateable."

They wrote: "The information stored by the logging capability of this malware is very extensive, and if sent back to the command-and-control server, could be used to implement keylogging, as well as collecting behavioral data on victims and on installed applications, even if they are not part of the list of targets. Xenomorph seems to be in its infancy, based on the fact that many commands are present in the code of the malware, but are not implemented." So just, you know, stubs. "In addition to this, the large amount of logging used also suggests that this might be an in-progress malware project.

"Despite having an in-progress number of features, Xenomorph contains code to support much more. Its accessibility engine," they wrote, "is very detailed, and is designed with a modular approach in mind. It contains modules for each specific action required by the bot and can be easily extended to support more functionality. Like many other Android banking trojans, this trojan heavily relies on the overlay attack mechanism to fool its victims into revealing personally identifiable information which could then be used by criminals to perform fraud. If the malware obtains the accessibility services privileges,

which it insistently requests after being started, it will automatically grant itself all the required permissions and then silently execute on the device.

"The main attack vector for Xenomorph is the classic overlay attack powered by accessibility services privileges. Once the malware is up and running on a device, its background services receive accessibility events whenever something new happens on the device. If the application opened is part of the list of targets, then Xenomorph will trigger an overlay injection and show a WebView activity posing as the targeted package. In addition, the malware is able to abuse accessibility services to log everything that happens on the device. At this time, all the information gathered is only displayed on the local device logs. But in the future, a minor modification would be enough to add keylogging and accessibility logging capabilities to the malware.

"As a first step, the malware sends back the list of installed packages on the device. And based upon what targeted application is present on the device, it then downloads the corresponding overlays to inject. The list of overlay targets returned by Xenomorph includes targets from Spain, Portugal, Italy, and Belgium, as well as some general purpose applications like emailing services and cryptocurrency wallets."

So I thought it was worth taking a moment just to talk about this, to sort of, you know, this is the state of the art of Android trojans. They're just playing a numbers game. They're spraying the Play Store with apps that some number of people, you know, that are appetizing, that look good, sometimes they'll even go as far as upvoting their reviews, posting fake reviews, arranging to get a high star count. Yet the entire goal of this app is not to do anything good, it's to get this trojan planted. And the post goes on to substantiate their observation that Xenomorph and Alien are closely related.

So we have an instance of an automated profiling app. With more than 3,700 apps on average being added to the Google Play Store every day, some percentage of them are malicious. Google is looking for them, I'm sure is catching a lot of them that never go live. Here's an example of one that did. 50,000 people downloaded it before it was found by this ThreatFabric firm and then reported to Google so that Google could yank it. And in this sense it sort of represents the flipside of the highly targeted exploits of rare and valuable vulnerabilities which we often talk about in iOS, Chrome, and Windows.

Generic Android banking trojans are instead opportunistic shotgun malware. They don't know or care who anyone is. They're just hoping that someone in their target regions will download it, will allow it to get in control, and then they will do something on their Android device which this thing is able to intercept and take advantage of. And the people behind this just keep pumping out these freebie utilities which clearly tens to hundreds of thousands of people will download, and unfortunately some will be unlucky.

So really the takeaway here is just as an Android user, and certainly to some degree as an iOS user, but to a lesser degree, primarily as an Android user, I guess I would say maybe allow apps to age for a while so that if they're malicious, somebody else is going to download it and discover it. Google will get around to checking it and see that it's bad. So I would, in general, I would shy away from things that have just been updated or just been uploaded that don't have a huge following. It's not a guarantee obviously of safety. But it's a heuristic that you could use. And the better thing is just to beware of our own conduct. Just try not to fill your phone with stuff because statistically the chances are the more things you fill your phone with, the greater the chance that one of them is going to be bad, and you're not going to like the outcome.

We haven't spoken much about the Hive ransomware, due to me working, as I said earlier, not to make this the ransomware podcast, which it could easily become because there's just so much, just as I don't want it to be the Android malware podcast. But I thought it was working talking about this one in this case. But this week I want to talk

about the Hive ransomware because the folks behind its design made a significant boo-boo in assembling this piece of software.

To set the stage, back in mid December, our original ransomware tracking site and a great site for security news, BleepingComputer, introduced Hive by writing, they said: "The Hive ransomware gang is more active and aggressive than its leak site shows, with affiliates attacking an average of three companies every day since the operation became known in late June." Okay, so these guys appeared around the middle of last year. BleepingComputer wrote: "Security researchers, gleaning information straight from Hive's admin panel, found that affiliates had breached more than 350 organizations over four months. The gang's data leak site currently lists only 55 companies that did not pay the ransom, suggesting that a large number of Hive ransomware victims chose to pay the ransom. A conservative estimation places Hive ransomware gang's profits into millions of U.S. dollars between October and November alone.

"Hive ransomware emerged in late June targeting companies in various sectors. While most of the non-paying victims on their leak site are small- to medium-sized businesses, the gang also published files from larger companies with revenues estimated to be in the hundreds of millions." And they finish by saying: "Analysts at cybersecurity company Group-IB investigating the Hive ransomware-as-a-service (RaaS) operation discovered that the group is one of the most aggressive ones, its affiliates hitting at least 355 companies by October 16.

The first publicly known attack from this gang was on June 23rd against Canadian IT company Altus Group. At that time it was unclear if Hive was a RaaS operation open to other cybercriminals. Things became clear in early September when the group, through a user known as 'kkk,' replied in a thread about 'reputable' ransomware programs that they were looking for partners that already had access to company networks. The message also included details about splitting the ransom money, 80% to the affiliates, 20% to the developers. The same user also provided technical information about the file-encrypting malware in a self-destructing note captured by Group-IB researchers. Although 'kkk' did not name the ransomware as a service they were representing, the researchers say that the technical details provided made it clear that the actor was referring to the Hive ransomware."

Okay. So Hive is a recently emerged ransomware-as-a-service entity. This makes them one among many. Why are we talking about them today? As I mentioned, the folks behind its design made a mistake. Last Friday a team of South Korean researchers published an academic paper with the title "A Method for Decrypting Data Infected with Hive Ransomware."

When the first ransomware emerged, the question of its decryptability of the ransomware-encrypted files was, quite naturally, the first thing that occurred to anyone, especially those who were untrained in the ways of cryptography. And it may have been that the very first ransomware wasn't very well designed because there have been certainly some ransomware decryptors, as we know, that have been created after a reverse engineering of ransomware found that, yeah, sure enough, these guys didn't do it right.

But as for cryptography, as I've often said, this is a solved problem. Unlike in fiction, you cannot run a bypass or just crack the crypto because you want to. There's no such thing. Trying harder or wanting it more doesn't help. These days, aside from side-channel attacks and key recovery from RAM, or like some way of subverting, anything that's been properly encrypted will remain encrypted unless and until the proper decryption key is applied. So what did the guys who designed the Hives crypto do? They made the one classic mistake. They rolled their own crypto.

Leo: Oh, no. You know you're not allowed, no no no.

Steve: It's such a dumb thing to do that the only possible reason I can see for doing so would be speed. Speed does matter for ransomware. Mass storage is, after all, massive. And it's only getting more so. So increasing the performance of the bulk file encryption, the symmetric cipher speed, would be potentially a big deal. It could mean all the difference between getting everything encrypted before being discovered as opposed to being shut down. And we've seen previous ransomware boasting about its increased performance. In the ransomware-as-a-service world, where attracting affiliates away from other ransomware service providers could be beneficial, being known to have the fastest file encryption in town could be a significant competitive advantage.

It is possible to do extremely high-speed ultra-secure encryption correctly. But they didn't do it right. If I were tasked with designing the fastest possible fully secure cipher, of course for the purpose of doing good rather than evil, I'd use RC4. I've always been a huge fan of RC4. Its sheer simplicity and elegance has always appealed to me. And Leo, I recall doing one of my appearances on your Call for Help show in Toronto on the topic of RC4.

Leo: Really. So it's pretty old.

Steve: Yeah, oh, yeah. Well, yeah. It's an early, early Ron Rivest cipher. That's what RC stands for is Rivest Cipher 4.

Leo: Is this symmetric?

Steve: Yes.

Leo: Okay.

Steve: Is it symmetric. And I used that whiteboard that we always got out for me in Toronto in order to draw it. And, I mean, it is really simple. It's not complicated. But it has an undeserved bad reputation due to its misuse. It is a bit tricky to use correctly because the first 3K or so bytes that it generates should be discarded. It is a keyed pseudorandom byte generator with a sort of built-in entropy pool. So it makes sense that you need to allow time for the entropy pool to stir itself up and get itself fully randomized.

When used as a cipher, it generates a pseudorandom bitstream, which is XORed with plaintext to create the cipher text. Its main problem, as I said, is that it does take a while to warm up and begin generating the highest quality bitstream. But once it has, its bitstream is of the highest quality at an incredibly low cost per bit. Not giving it time to warm up is the mistake that the designers of the early WiFi encryption made when they chose RC4. They used the start of the stream, which is well known to contain a detectable influence from the key, which is a big no-no. And after mucking up their original implementation, the Wi-Fi Alliance who keeps demonstrating their lack of genius chose to abandon RC4 and switch to AES. Not that that was a bad choice. AES is wonderful. I'm a big fan of the Rijndael cipher. But it is far slower than RC4.

In any event, we've spoken many times about the surprising power of the exclusive or operation. XOR is simply a conditional bit flipper. Any bit you XOR with zero remains the same. Any bit you XOR with a one is inverted. And so it's counterintuitive that simply using a pseudorandom bitstream to choose which bits to flip in a plaintext could convert it to truly uncrackable cipher text. But it can, and it does. And we've talked about how tricky it can be to use XOR properly. The similar famous case of that is the one-time pad. If a one-time pad is used exactly one time, as its name urges, simple as it is, it is uncrackable without the key. But if it's ever used a second time, its security completely fails.

The XOR, for all its potential power, is similarly brittle. If it's ever possible to know a plaintext for the matching ciphertext, then those can be XORed to recover the key stream. And a somewhat more convoluted version of that mistake is what the South Korean researchers discovered. So here's a short relevant passage from their 23-page paper where they provide an overview of when they found and did.

They wrote: "Recently, many ransomware attacks have been found to use a hybrid encryption scheme that encrypts users' files with a symmetric cipher and stores the encryption keys used with an asymmetric cipher." You know, that's the model that we've talked about always. They said: "Most ransomware uses secure algorithms such as AES and RSA for encrypting files. Therefore, if an attacker's private key is not obtained, it is difficult" - well, yeah, they say "difficult," we're talking virtually impossible - "to decrypt the encrypted files. However, certain ransomware may use a self-developed encryption algorithm when encrypting files.

"If attackers cryptographically misconfigure the ransomware" - which is their polite way of saying if they design a bogus encryption system - "a cryptographic vulnerability can occur. The Hive ransomware encrypts a victim's file using an encryption algorithm developed by the Hive programmers. We analyzed Hive ransomware and discovered the detailed operation process of the Hive ransomware. Hive ransomware uses a hybrid encryption scheme, but uses its own symmetric cipher to encrypt files. We were able to recover the master key for generating the file encryption key without the attacker's private key by using a cryptographic vulnerability identified through analysis. As a result of our experiments, encrypted files were successfully decrypted using the recovered master key based on our mechanism. To the best of our knowledge," they wrote, "this is the first successful attempt at decrypting the Hive ransomware."

So then they have three numbered points. They said: "We experimentally demonstrated that more than 95% of the keys used for encryption could be recovered using the method we suggested. Our contributions are summarized as follows." So here's the three. First, they said: "We identified the way in which Hive ransomware generates and stores a master key for victim files. Hive ransomware generates 10Mb of random data which it uses as a master key. For each file to be encrypted, 1Mb and 1Kb of data are extracted from a specific offset of the master key and used as a keystream. The offset used at this time is stored in the encrypted file name of each file. Using the offset of the keystream stored in the filename, it's possible to extract the keystream used for encryption."

Okay. That all makes sense. So basically these guys, because they're clearly interested in speed, they come up with what they think is a super spiffy fast way of doing encryption. They're going to generate 10Mb of pseudorandom data one time. Then they're going to randomly choose an offset into that and grab one megabit for one purpose and a kilobit for another and use that as the encryption.

So they say, point two: "We analyzed the Hive ransomware to uncover its operation process and a newly developed encryption algorithm process. Hive ransomware encrypts files by XORing the data with a random keystream that is different for each file." Right.

Right? Because they chose a different offset into the single master keystream. They said: "We found that this random keystream was sufficiently guessable."

And finally, point three: "We suggested a method for decrypting encrypted files without the attacker's private key. We found that the Hive ransomware does not use all bytes of the master key encrypted with the public key. Using our proposed method, more than 95% of the master key used for generating the encryption keystream was recovered. Most of the infected files could be recovered by using the recovered master key. We present experimental results for the case of recovering files using our proposed method."

Okay. So from this description we can be pretty certain, as I said, that encryption performance was the goal. The improperly conceived ransomware generated a single static reusable and reused 10Mb XOR keystream and then chose various 1Mb chunks at a per-file offset into that keystream for its encryption. In taking this approach, tempting as it was, they broke the cardinal rule of XOR-based encryption, which is never reuse the same keystream. The bad news is software can be updated, and I'm sure that the Hive ransomware technology will be updated immediately, you know, probably already has been, or they're at work on it. So only those victims who have been encrypted by this first version of the Hive, but haven't yet paid the ransom, might be helped by this research.

Still, this was very nice work, and it demonstrates that we should not simply assume that any random ransomware was properly designed to be truly uncrackable. It is definitely worth going in, reversing the ransomware, figuring out how it works, and verifying that there isn't a simple way, or even less than simple, like these guys tackled, but still possible way of reversing the encryption. So bravo to them. Very cool. And another couple interesting lessons about XOR. If you use it right, it is the fastest thing there is. But you really need to be careful with the way you use it. And I need to be careful with my throat, Leo, which is getting dry and scratchy.

Leo: Deal. I'm playing with my new Samsung Galaxy S22 Ultra.

Steve: Look at all the lenses on that thing.

Leo: It's like a bee or something; right? All those lenses. Yeah. Pretty, though. This is the burgundy color.

Steve: Okay. So another fun story caught my eye this week. Dan Petro, the lead developer at the security firm Bishop Fox, titled his posting last week "Never, Ever, Ever Use Pixelation for Redacting Text." So the issue here is the somewhat common practice, and we've all seen it, of redacting text to make sections of it unreadable by pixelating the underlying image into blocky blocks of varying shades of gray. Dan intuitively understood, although he didn't describe it in information theoretic terms, that using this approach for true secrecy was a bad idea.

In his posting, he said: "So there's an existing tool called Depix that tries to do exactly this through a really clever process of looking up what permutations of pixels could have resulted in certain pixelated blocks," he says, "given a de Bruijn sequence of the correct font." He says: "I like the theory of this tool a lot, but a researcher at Jumpsec pointed out that perhaps it doesn't work as well in practice as you'd like. In real-world examples you're likely to get minor variations and noise that throws a wrench into the gears. They then issued a challenge to anyone, offering a prize if you could unredact the following image."

And I have an image in the show notes for those who eventually look at the show notes. It just shows your typical pixelated gobbledygook. It's completely illegible. Whatever it used to say, it's like it's four blocks high, and I don't know, like maybe 60 blocks long. And no idea what that is. And so, and Dan then asks rhetorically, "How could I refuse such a challenge?"

Okay. So work on unredacting blurred or pixelated text has been done before. Dan referred to one, that Depix. But Dan's solution was clever. He solved the challenge by writing a simple search algorithm. He started with the first character or two, pixelated it, and compared his pixelation with that much of the goal. He created a metric describing how closely the two matched, then guessed again. After he'd run through all possible combinations, so in that sense it's sort of a brute-force cracking, right, except he's able to do it in the science fiction way that we see on TV where all the digits are spinning, and then they sort of lock in one at a time. So after he'd run through all of those combinations of what the first character or two might be, he then chose the next one which produced the most closely matching pixelation, and then started working on the next character, and so on.

He provides a demo of his algorithm showing us it working. And I found an animated GIF image. It is, as I said at the top of the show, this week's picture of the week, grc.sc/859. It does a beautiful job of - he has a target blurred image, and then you can see in, again, grc.sc/859, you can see him guessing characters, pixelating them, and then his algorithm checks to see how close his pixelations match the target pixelations, for which he gets a metric of essentially the error, the difference between them. So lower is better, meaning less error. And then after running through them all, he chooses the one that had the lowest error and goes to the next one.

Okay. So the takeaway is, if you ever seriously want to redact text, use a simple, if not inspiring, and not nearly as fun, solid black bar to do so. Never get cute with pixelation or blurring or anything else that allows some of the underlying image to survive in any form. If it's possible to duplicate the algorithm that was used to perform the original obscuring, a search strategy such as Dan's can always be used to find the original text that was obscured by simply guessing, duplicating, and comparing successively. So anyway, I just thought that was a cool little bit of tech, and clearly the right way to do this. And what's a bit sobering about it is that it is so universally applicable. I mean, it's just - it will solve the problem. You do need to be able to have your guess text blurred, fuzzed, defocused or whatever, using the same algorithm that was used. But the pixelation algorithm is super trivial. So Dan was able just to guess what it was.

Okay. I titled this one "No Internet for You!" A small town in France was facing a mystery. From midnight until 3:00 a.m. in the morning every day of the week, the town's cellular and Internet services mysteriously stopped working. They just went dead. Finally, a mobile carrier reported the mysterious issue to the public agency responsible in France for managing the RF spectrum there, and an investigation ensued.

It was soon discovered that a homeowner had purchased an illegal RF signal jammer as the only means he could find to get his social media-addicted teenagers to go to sleep at night. He had to force their cell phones to stop working. The father explained that, after consulting forums on the Internet, he came to the conclusion that a jammer was the best solution to put an end to his teenager's excesses.

Although it was certainly not the father's intention to bring down the entire town's Internet - and just as an aside, you've got to wonder, like, what? A jammer in the attic can bring down an entire town's Internet? That's got to be a very small town, or a very fragile Internet, or maybe both. Anyway, it is illegal to do this, just so that everyone is on notice, not only in France but in the U.S. In France it carries a penalty of 30,000 euros and up to six months in jail. And it's also a no-no in the U.S.

And, you know, in this case it's unclear what's going to happen to Dad, but some other solution will need to be found to keep the kids off of the Internet. You could, I guess, since they're phones, you can't turn off the household WiFi because they'll just switch to cellular. But how about just handing them over to Mom and Dad in the evening, and get the homework done, and then you could have them in the morning. So anyway, thought it was interesting that it's a little overpowered jammer that Dad got.

It turns out that perhaps Adobe's - and I'm being so careful with this word now. I'm, like, I don't want to say Magneto, I want to say Magento, the Adobe - or actually it's the Magento Adobe Commerce Platform developers may actually have had the game on in the background and weren't paying as close attention as they should have been while they were working on and rushing out that five-alarm emergency-drop-everything Super Bowl Sunday update because they didn't fix the whole problem. Very early, at 1:17 a.m. last Thursday morning, the guys at Positive Technologies tweeted that they had managed to reproduce the Improper Input Validation vulnerability in Magento Open Source and Adobe Commerce which was supposedly fixed with that previous Sunday's out-of-cycle patch. It wasn't.

Adobe replied: "We have discovered" - uh-huh, yeah. "We have discovered..."

Leo: You know what? I'm thinking they patched Magneto, and they forgot to patch Magento.

Steve: That's it.

Leo: That's it.

Steve: They patched Magneto.

Leo: It's safe now, by the way. You can use Magneto.

Steve: You could crank that Magneto all you want. That's right. Adobe replied: "We have discovered additional security protections necessary for CVE-2022-24086 and have released an update to address them as CVE-2022-24087. Adobe is not aware," they wrote, "of any exploits in the wild for the issue addressed in this update." Right.

Like the previous Sunday fix, this is also a pre-authentication remote code execution with a CVSS of 9.8. And remember that this is PHP source being updated, so what's changed will be no mystery. Hopefully, if you're responsible for maintaining Magento, or Magneto, well, actually Magneto you're safe from the Sunday patch, apparently. But if you're responsible for maintaining Magento, or Adobe's commerce platform, this is already old news to you. If not, this is one to fix immediately because, I mean, we know that the previous one was already being used in attacks. So, yeah.

Last Tuesday, in what had to be I guess a plain and stark run-up to Russia's designs on Ukraine, which of course there was some news about last night, multiple institutions which are critical to Ukraine's military and economy were hit with denial-of-service attacks. The attacks triggered, just because everyone's watching, I suppose, a great deal of global news and speculation despite the fact that the impact of the attacks themselves was somewhat limited. The trouble is, the ramifications of such attacks are not.

The targets of the attacks were the Internet presence of the Armed Forces of Ukraine, the Ministry of Defense, and the country's largest two commercial banks, which was considered to be systemically important to Ukraine's financial functioning. And these could very well have been just some target scaling. We know that that's the nature of denial-of-service attacks.

Like most of the world, I'm quite worried about what's going on over there. But as always, what interests us here is technology. Adam Meyers, the Senior VP of Intelligence at CrowdStrike, said in an email to reporters at Threatpost that the attacks consisted of "a large volume of traffic, three orders of magnitude more than regularly observed traffic, with 99% of this traffic consisting of HTTPS requests." That caught my attention because there are many implications there.

First of all, HTTPS rides on top of TCP. TCP connections are established with a three-way handshake whose purpose is to verify and establish a functioning two-way connection between the endpoints. As we know, both sides exchange randomly chosen 32-bit synchronization packets which will be used to number the data, the number of the data bytes that they subsequently exchange. And it's this three-way handshake that prevents IP address spoofing of TCP connections. Unlike with a UDP query, which is just a single packet, it's not possible to spoof the IP of an HTTPS web query which runs over TCP.

And the second part of this is the HTTPS web query itself. Attacks are typically far more effective if they're able to exploit large asymmetries between the attacker and their target. In the case of UDP traffic, we've talked about the bandwidth amplification that can be created by many Internet services. The simplest example is probably querying a DNS server where the query is small but the reply is large. Size asymmetries like this are quite common on the Internet. If attackers can ask a short question of a publicly available Internet service using the UDP protocol, which only requires a single packet, they can spoof the source IP carried by that query packet to cause the reply to be sent to their targeted victim. In this way a little bit of traffic generates a much larger attack.

Of course, this form of asymmetry doesn't work for HTTPS, since the IP address of the endpoint generating the HTTPS query cannot be spoofed. But a far more damaging asymmetry does exist. A constant topic of this podcast is the fact that today's fancy websites are not built from simple static HTML pages previously written and stored on a mass storage device and then probably cached in RAM. Today's sites are not archives, they're applications. As such, incoming queries are examined and handled by software, typically Java, C#, PHP, Python, Node.js, or perhaps Ruby on Rails. Several of those languages are neither multithreaded nor particularly high performance. Many of them are interpretive.

As a result, they don't scale well. To make matters worse, they typically read from and need to interpret page templates which direct them in the formation of the page to assemble and return, which almost always necessitates multiple if not many queries to a back-end database.

As an industry, we've built extremely complex and capable data-driven web applications. We've kept them understandable and maintainable by using templates and databases. But all of this elegant design has come at the high price of efficiency. As a result of many levels of interpretation and lookup, today's websites have extremely high computational and mass storage lookup costs per page. And this is the asymmetry that modern HTTPS query flooding attacks are leveraging.

A botnet is needed to generate such attacks, since the attacker's IP addresses cannot be hidden. And there must be many tens of thousands of attacking IPs, otherwise the unspoofable IPs of a few attackers can be trivially blocked by a firewall filter. But as we know, botnets consisting of many tens of thousands of previously compromised devices

do exist. Many, in fact. And all of the flawed design IoT devices, which we also often talk about here, which continue to be casually connected to the Internet every day, just adds to that potential inventory. As we've noted before, the only real way to know how strong a botnet is, is to fire a test shot. What the world saw a week ago in Ukraine was an HTTPS-query-capable botnet flexing its muscles. We may be seeing more of that in the future. And nothing prohibits it from being aimed at any website where there is a high cost per page.

I know that - I'm blanking on the name - Cloudflare has built some specialty around protecting that kind of site. Sometimes when I'm checking onto security sites, since security sites seem to often be a DDoS attack target, I'll get kind of this weird intercept page from Cloudflare saying that they're verifying my browser or something, which is clearly like a first-stage intercept to say, okay, we're going to let this guy through, or not. So anyway, I have every expectation that, I mean, everyone keeps saying that cyberwarfare is the next big thing. And if so, it's going to be botnets of this sort not doing the old-style SYN flooding, but probably bringing our websites down because it is so expensive now to produce a single page to a visitor. If you're able to produce three orders of magnitude greater traffic than a given site has already been scaled to handle, it's gone. It's off the Internet.

Yesterday I pushed to finish the third of the so-called "Bobiverse trilogy" so that I could report today. And I can now recommend it without hesitation.

Leo: Oh, good. I just downloaded the first one. "We Are Legion (We Are Bob)."

Steve: Yes. All of our listeners who were telling me that I would love it were right.

Leo: Nice.

Steve: As I had mentioned before, I was initially somewhat cooler and tentative about it. The author's writing style is more, I guess, expositive or casual than either Ryk Brown's or Peter Hamilton's. Both Ryk and Peter spend an inordinate amount of time on character development. That's a big part of what makes their universes so rich and believable, but at the cost of their novels being nearly interminable. Boy. By comparison, Dennis Taylor just jumps right in and starts telling his story. So while his style was initially a bit jarring to me because it's so much different from what I had been reading, I wound up truly loving his story.

It's also a lot of fun that Bob, the primary characters in the books, was himself an avid sci-fi fan, having read all the same books and watched all the same TV and movies that we have, as the readers of the story. So the continual references, scattered throughout the trilogy, to things that only a true fan of science fiction would pick up on, like naming a vessel the Bellerophon, made the journey through the trilogy extra fun.

So there is a fourth novel that's about twice the length of the largest of the previous three. I think the larger of the previous three was 300 and some pages. This fourth one, "Heaven's River," is 600 and some pages. So it's also available through Kindle Unlimited and Audible, as are all of them. So that's what's next up for me while I await more from Ryk and Peter.

And Leo, I do think you're going to have fun. Apparently the reader is also a well-known and expert Audible reader.

Leo: Ray Porter, yeah. He's good. I've heard him before, yeah. I'm excited. Can't wait.

Steve: Yeah. Okay. InControl, which I announced last week, settled down quickly following its announcement last week. I finished the technical documentation of the Registry keys it manages, added version tracking and an FAQ, then got back to work on SpinRite. I've accomplished so much since SpinRite's last major development release that it's time for another. I was actually hoping to get that done on Sunday, but I ran out of time. But it'll be happening within a couple days.

I mentioned previously that I had discovered that in some cases the private information that a third-party adapter has about its own hardware, and actually in some cases even the motherboard, gives it a special advantage and keeps SpinRite from outperforming that adapter's own firmware. Since SpinRite's total scanning time is crucial to the feasibility of its use, and since performing an occasional media scan is really the only way to spot, catch, and repair trouble before it becomes irreparable thus SpinRite's long-term maintenance aspect nothing is more important than minimizing that scan time.

Therefore, I realized that I needed to have SpinRite perform mini-benchmarks on those drives where it could choose which way to access the drive during its bulk surface scanning work. So SpinRite now incorporates an integrated mini-benchmark that takes very little time, it just flashes by, but it allows SpinRite to quickly choose the fastest bulk data access method.

And as I think I mentioned before, whatever I'm doing with the AHCI controller is blowing everybody away, like by a factor of two or more. SpinRite just saturates the SATA bus on AHCI controllers. But that's not so much the case on the older ATA bus mastering controllers. Anyway, I realized after having done that, that I could utilize the results from the mini-benchmark to estimate the entire drive scan time and show that in the UI in the same way that I had been showing it only after the user and only if the user had first deliberately performed a full-scale benchmark, but without needing the full benchmark.

And I saw that, I got that done, and I saw that result on Sunday. And it was somewhat astonishing to me to see what a huge difference just that little change makes. You start SpinRite. It finds all the drives attached to the machine. And then you go to the screen where you select which drive you want to run it on. And just as it always has, it shows you the size of the drive. But now, listed there beside each drive is the estimated time to scan that drive. It's just a small new little feature. But it turns out, since nothing is more important than how long it's going to take to do this, it's a real cool little new feature.

So anyway, just for any of our listeners who are working with me, do check into the newsgroups in the next day or two. The pre-release number nine will be there. And I can't wait to confirm all the known issues that we fixed. And we have 15 that have been outstanding, and more than half of them have been fixed for a long time. But I've been working on this other stuff, wanting to get as much done as I could before the next drop. But it's time to do that. And that'll allow us to get resynchronized and find out what issues we still have. So we're getting there.

Leo: Once again for Mr. Steve Gibson and the meat of the matter, the heart of the show today, BGP.

Steve: So this is tricky, but I think I'm going to be able to explain it to everybody. We'll also regress a few times to talk about some of the underlying technologies that we need to understand what happened.

Earlier this month, on February 3rd, a Thursday, customers of a South Korean cryptocurrency platform known as KLAYswap lost a total of 2.2 billion Korean won, which is equivalent to around $1.9 million U.S., so no small amount, as a result of 407 fraudulent, intercepted, and altered cryptocurrency transactions across a total of 325 KLAYswap customer wallets. So KLAYswap is a cryptocurrency exchange. It's got customers. Something bad happened. And of course, as we know, such things have happened before. But the way this was engineered to happen was diabolical and somewhat horrifying.

Okay. So as we know, the success of most cryptocurrency heists can be traced back to some security failure on the part of the cryptocurrency exchange itself. Attackers might compromise the account of an employee who's using a weak password and no additional authentication factors. Or bad guys might find a flaw in the platform's online server code, use that to steal funds from the platform's customers' accounts. Or maybe use a flaw in the operating system software and get in and then do that. None of that happened here.

When any of us visit any sort of modern website, whether it's Amazon, any Google property, Facebook, PayPal, our bank or whatever, the first thing that happens is that the remote site moves a chunk of its application code into our browsers in the form of JavaScript. As we've all learned, the days when much can get done on the web without any browser-side scripting, or with browser scripting deliberately disabled, are pretty much past. As we all know, I was using NoScript as long as I possibly could, as were many of us, until I found myself needing to disable it more than not because more and more sites were just assuming that the browser I was running knew how to do JavaScript. So I finally gave up and switched to Gorhill's uBlock Origin. And I do sure hope I never have to give that one up.

In any event, accepting a huge blob of code from any site we're visiting is the way the web has evolved. And that was also true when KLAYswap's customers visited the KLAYswap site. Their browser would grab the client-side support code, over HTTPS as everything needs to be these days, from the domain developers.kakao.com, which was part of the KakaoTalk's developer infrastructure where KLAYswap hosted its official SDK which allowed third-party apps to integrate with its services. This was all fine. Client browsers go to KLAYswap, obtain their own version of JavaScript, which allows them to integrate, that is, their browsers to integrate, with KLAYswap's back end services. All good.

Now, we've talked many times in the past about how Internet routing works or, more specifically, about how routing tables function. In the IPv4 space, which is actually where this occurred, we start off with a 32-bit address. Part of the original genius of the designers of the Internet was to realize that, rather than just assigning addresses totally at random, they realized that if the Internet supported a 32-bit machine address space, and all of the machines associated with, for example, a single organization or a network could have their addresses grouped together, then their addresses would all have identical most significant bits.

In other words, when viewed as 32-bit binary addresses, all of the bits at the left end of the address, of all of their addresses, moving right to some degree, would be the same. All the left side of some number of bits would be identical. The original designers called this common set of bits the "network," or more formally, the "network prefix." And they realized that this could be used globally to route packets whose leading, you know, the left end bits were such-and-such to that entity's router for then further distribution to the individual machines there. So that was like the origin of the concept of having a network

where the network prefix numbered that network, and then you had lots of little machines within that network.

Next they realized that multiple organizations near each other and being served by the same larger Internet service provider, might differ in only another couple of bits further to the left. This meant that all packets having slightly fewer identical left side bits could simply be sent to the ISP's network, which included all of its customers' subnetworks. Then the ISP's routers could further subdivide its larger all-inclusive network into its individual customers' subnetworks. In other words, the Internet, being a network of networks, can be viewed as a hierarchy of networks. A few big mega networks, the so-called Tier 1 providers or networks, the backbone providers, whose networks are successively subdivided and subdivided and subdivided until they get down to an individual customer.

I wanted to refresh this in everyone's mind because we need to understand something that naturally falls out of this sort of hierarchy. The way routing tables work is that, being a table, they contain a large list of these network prefixes. And to determine where any given Internet packet should be sent, the left-most bits of each packet's address is compared to each entry in the router's routing table. It's looking for the longest network prefix in the table that matches the packet's destination IP. Once the longest prefix match, as it's called, the longest prefix match is found, the packet is routed toward the network having that longest matching prefix.

And finally, with Internet routers scattered all over the Internet, the last piece of technology we need is a means for maintaining all of these separate routing tables. That's what the Border Gateway Protocol provides. Routers that are connected to each other continually share their routing information with their peers. If, for example, some new and previously unused block of IPs were to be assigned to someone connected to a router, that router would need to tell its peers the equivalent of, "Hey! I've just become responsible for this block of IP addresses. So if you receive any packets for that new network, forward those to me."

BGP is what makes that possible. And when you stop to think about it, the news of that new network needs to be spread. The router that directly receives that announcement, or advertisement as it's formally called, first adjusts and updates its own routing table to incorporate the news of that new route. Then, having adjusted its own table, it sends out the announcement of its change to each of its peers in turn. And they do the same. Thus within a very few minutes the news of this new network will have propagated across the globe, and every router will know where to send packets, if they receive any, for that new network.

We've also talked about Autonomous Systems, or AS numbers. Any entity that has blocks of IP addresses assigned to it has an AS number. And it is said that the Autonomous System "owns" those IPs. This just means that any Internet traffic placed onto the Internet anywhere, whose network address prefix matches any of the prefixes owned by that Autonomous System, will have those packets routed to them.

Kakao.com, the Internet service provider, is Autonomous System number 38099, and that provider owns a block, actually owns many blocks. But one of the blocks of IP addresses, a block of 512 IP addresses at 121.53.104.0/23. The /23 indicates that the left-hand 23 bits of the IP address is the network prefix. So since we have a total of 32 bits in the address, and 23 is the network prefix, that means that the remaining nine bits at the right end of the IP address are the machines within that network. When routers encounter packets with the proper first 23 bits, that means that the packet is destined to AS38099. So that's where the router forwards the packet.

Kakao.com's server at developers.kakao.com is the domain serving the JavaScript for this cryptocurrency exchange. Developers.kakao.com lives at IP address 121.53.104.157, which is within those 512 IP addresses owned by AS38099. So all was well.

Until someone posing as AS9457 - which actually is a major player. Dreamline Co is a major Internet carrier in South Korea with control over, meaning owning, 1,793,723 IP addresses. Someone posing as AS9457 managed to sneak a fraudulent BGP routing table update into the global network. Now, additional technical details are available, but they're all written in Korean, which is not a language I'm able to read. It may have been that it was a compromise at Dreamline Co, or just a break- in to the inter-router BGP communications. Or maybe they actually did edit the routing table of a router belonging to Dreamline. We don't know. Both have occurred in the past.

But what happened in this case is someone managed to sneak a more specific route into the global routing tables. Whereas the actual route for the real network is a /23, the bad guys or, Leo, the miscreants, as we like to call them...

Leo: Yes.

Steve: ...snuck in a /24 route which specified one additional bit of the network prefix for the network which also contained the developers.kakao.com IP address and server. Since all of the world's routers saw that as being a more specific route because it specified a longer network prefix, the traffic that should have gone to the proper AS38099 was instead routed to a machine somewhere in AS9457. And the guys who set this all up were, of course, ready. They had a server ready to impersonate the original, the authentic "developers.kakao.com" machine, complete with a freshly minted authentic TLS certificate which they had obtained the instant they commandeered the network. If you own an IP address, you can quickly obtain a certificate for any machine that DNS points to there.

Once they had set up their clone of developers.kakao.com, they began serving malicious JavaScript from that host. The actual file was https://developers.kakao.com/sdk/js/kakao.min.js. So it was minified JavaScript. For a period of two hours, from 11:30 to 1:30 on February 3rd, KLAYswap's customers who visited KLAYswap's site to perform secure cryptocurrency transfers unwittingly received malicious JavaScript, despite the fact that their browsers were pulling code over HTTPS from the correct URL.

The malicious JavaScript had been modified to include additional code at the end of the file which would wait for the user to initiate a transaction on the KLAYswap website, such as an asset deposit, a swap, or withdrawal. When such an action was detected, the code would hijack the funds and send the user's assets to an attacker-controlled wallet. From there the funds were immediately laundered through Orbit Bridge and FixedFloat, two cross-blockchain conversion services, thus laundering them. Two hours after the attacks began, still undetected at that time, the attackers voluntarily removed the fraudulent routing table entry from BGP, and normal packet flow resumed.

When you consider all of the moving pieces that had to be in place, working and tested, this was a very impressive attack which had to have taken months of planning. These guys had a high-end knowledge and understanding of many facets of the operation of the Internet. And they had some access somehow because you have to have that, too. And they may have needed to create custom tools to make this happen.

There's been a lot of talk about securing BGP. It's a huge and glaring problem. But still today Internet routing is largely based upon trust. When that trust is undeserved, there's

very little that can be depended upon. It would not be difficult to make the case that the Internet is still not ready for primetime. Mostly it works, but in specific highly targeted instances like this, it really does seem that we come back to targeting being a problem with the way some of our systems work.

Anyway, I just thought it was fascinating that what we normally see as a mistake, where a chunk of the East Coast disappears because somebody entered something wrong, or all of the Internet gets routed through Belgrave for some reason and crashes it, in this case no. Just a selected little chunk, a little /24 got peeled away from a /23, sent somewhere else, and that's all it took because the guys were ready for what would happen when they had that access.

Leo: Just that easy. Wow.

Steve: Wow, yeah.

Leo: Yeah. Well, that's good to know because I always do blame the originators for BGP errors. Sometimes it's not your fault. Good to know. Steve Gibson, it's never his fault. He's always here to enlighten, and does that every Tuesday, 1:30 p.m. Pacific, 4:30 Eastern, 21:30 UTC. If you want to watch us do it live, you can at live.twit.tv. After the fact he's got 16Kb audio of the show, 64Kb audio, transcriptions, all at his website, GRC.com. We also have copies of 64Kb audio and video at our website, TWiT.tv/sn.

When you head over to GRC, pick up a copy of SpinRite, the world's best mass storage, maintenance, and recovery utility. Version 6 is out. As you can hear, we're getting close to 6.1. You can participate in the development, and you'll get a free copy of 6.1 if you buy now. SpinRite is available at GRC.com, along with ShieldsUP! and all the other many, many, many wonderful freebies Steve offers us, including his forums. You can leave a message for him there at GRC.com/feedback, or on his Twitter. His DMs are open at @SGgrc. @SGgrc is his Twitter handle.

I think we have done everything we need to do. A last chance to take our survey. It ends on the 28th, so this is the last chance you have to do it. If you go to TWiT.tv/survey22, take a few minutes. Helps us know you better for both purposes of ad sales, but also to decide what kind of programming you want. And yes, I know, it doesn't ask you if you use Linux. We're just going to take it as assumed that if you use a computer, you're using Linux. So, you know, you don't have to even write that in. We just know. We just know you use Linux. TWiT.tv/survey22. Of course it's voluntary. But it sure does help us an awful lot. Steve, have a great week. I'm off to the Bobiverse.

Steve: Cool.

Leo: And I'll see you next time on Security Now!.

Steve: You're going to have fun. See you then. Bye.