Security Now! #859 - 02-22-22
A BGP Routing Attack

This week on Security Now!

This week we talk about another WordPress plug-in mess, this one so bad that WordPress themselves force-installed updates on more than 3 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. We unfortunately need to revisit the supercritical Magento / Adode 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.

SPARED NO EXPENSE
memegenerator.net
HIRES ONLY ONE COMPUTER PROGRAMMER TO RUN THE WHOLE PARK FULL OF MAN-EATING DINOSAURS
imgflip.com

Security News

The “UpdraftPlus” WordPress Plug-In

A three year old flaw has been found in the “UpdraftPlus” WordPress plug-in which is in active use at more than 3 million sites including those at organizations including Microsoft, Cisco and NASA. It has been assigned CVE-2022-0633 and given a high common vulnerability severity score of 8.5. The flaw can be readily weaponized to download a site's private data using an account on the vulnerable sites.

UpdraftPlus is a backup and restoration solution that's capable of performing full, manual, or scheduled backups of WordPress files, databases, plugins and themes, which can then be reinstated through the WordPress admin dashboard. As a consequence of a defect that was found by a security researcher at Automattic, WordPress' parent company, any logged-in user to a WordPress installation with UpdraftPlus installed would be able to obtain the privilege of downloading an existing backup, which should have been restricted to admin users only. 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 a heartbeat request containing a "data" parameter to obtain information about the most recent backup. Having this information, the attacker triggers the "send backup via email" function after manipulating the endpoint request. This function should be strictly restricted 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 is that any attacker would need to have an account of some kind on the site. So that minimizes the risk somewhat. However, the popularity of UpdraftPlus coupled with the simplicity of this attack makes for a potent combination.

The combination is so potent, infact, 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. 783,000 sites were updated by WordPress last Wednesday the 16th, and an additional 1.7 million were 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 3+ million sites using UpdraftPlus are recommended to update to version 1.22.3 (or 2.22.3 for the Premium version) to mitigate any potential exploitation. Another problem was fixed since then, related to printing auto-backup options under PHP 8. So the latest version available as of last Thursday, February 17 is 1.22.4.

“Xenomorph”

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: “Xenomorph.”

The problem 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 all is Android's fault. 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. Thus, nearly seven times the opportunity to trick someone into downloading something that hasn't yet been flagged as malicious and removed from the Google Playstore. Now, that said, the increased level of freedom that Android's apps and users demand and enjoy over Apple's does make the Android platform more comfortable for malware. But since we're here talking about Xenomorph, it's worth touching upon it just as a reminder to the Android-using contingent of our listeners.

For the past ten years, the Amsterdam, Netherlands, based security research company ThreatFabric has been focused upon banking malware. Yesterday they published the first report of their discovery of a new malware family they named “Xenomorph.” I'll share just the top of their report, then I'll summarize the rest. They start off by introducing us:

In February 2022, ThreatFabric came across a new Android banking Trojan, which 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.

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.

Just like the monster protagonist of the famous Ridley Scott’s franchise, this malware shares some aspects with its predecessor. However, despite its obvious ties to one of the most wide-spread malware of the last two years, Xenomorph is radically different from Alien in functionalities. This fact, in addition to the presence of not 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 Alien, or at least of someone familiar with its code base. However, this is only speculation at the moment.

Google has been doing what is, I'm sure, their best to keep their PlayStore 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 million. 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? You betcha.

A case in point, which put ThreatFabric onto the malware they came to name Xenomorph, was a Google PlayStore offering called “FastCleaner.” It's a utility aimed at speeding up a device by removing unused clutter and removing battery optimization blocks. The application itself seemed successful, with more than 50,000 installations reported on Google Play. But upon their analysis, the ThreatFabric folks recognized that this application belonged to the so-called “Gymdrop” 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 of the "FastCleaner" app, 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 they refer to as ExobotCompact.D. But the other was brand new, and thus obtained the name **Xenomorph**.

This is what they had to say about Xenomorph:

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 to steal credentials, combined with the use of SMS and Notification interception to log and use potential 2FA tokens.

The Accessibility engine powering this malware, together with the infrastructure and C2 protocol, are carefully designed to be scalable and updatable.

The information stored by the logging capability of this malware is very extensive, and if sent back to the C2 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 stage, based on the fact that many commands are present in the code of the malware, but are not implemented. In addition to this, the large amount of logging used also suggests that this might be a in-progress malware project.

Despite having an "in-progress" number of features, Xenomorph contains code to support much more. Its Accessibility Engine 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 functionalities.

Like many other Android Banking trojans, this trojan heavily relies on the overlay attack mechanism to fool its victims into revealing Personal Identifiable Information (PII), 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 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.

The post goes on to substantiate their observation that Xenomorph and Alien are closely related.

So what we have here is an instance of an automated profiling app. Apps like this one slip into the Google PlayStore everyday among the more than 3700 apps being added everyday. If it's granted Accessibility privileges on Android it will abuse those privileges to inventory the device, determine what intersections it might find with whomever downloaded the malware and the banking or cryptocurrency it's prepared to exploit, then it will lurk in the background waiting for its opportunity.

In that sense it represents the flipside of the highly targeted exploits of rare and valuable vulnerabilities in iOS, Chrome or Windows. Generic Android banking Trojans are opportunistic shotgun malware. They don't know or care who anyone is. The people behind this just keep pumping out freebie utilities which, clearly, tens to hundreds of thousands of people will download. And some will be unlucky.

Decrypting "The Hive"

We haven't spoken much about "The Hive" ransomware, mostly due to me working not to make this the ransomware podcast, just as I don't want it to be the Android malware podcast, either of which could easily be all we talk about every week. But this week we will talk about "The Hive" ransomware because the folks behind its design made a significant boo boo.

To set the stage, back in mid December, our original ransomware tracking site, BleepingComputer introduced Hive by writing:

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. [So these guys appeared in the middle of last year.] Security researchers, gleaning information straight from Hive's administrator 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.

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 23, 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% for affiliates, 20% for 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 RaaS they were representing, the researchers say that the technical details provided made it clear that the actor was referring to 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 the decryptability of ransomware-encrypted files was, quite naturally, the first thing to occur to anyone, especially those who are untrained in the ways of modern cryptography. As I've often said, this is a solved problem. Unlike in fiction, you cannot "run a bypass" or just "crack the crypto." 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, 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 Hive's crypto do? They made the one classic mistake: They rolled their own crypto.

It's such a dumb thing to do that the only possible reason I can see for doing so, is speed. Speed does matter for ransomware. Mass storage is massive and is only getting more so. So increasing the performance of bulk file encryption would be a potentially big deal. It could mean all the difference between getting everything encrypted before being discovered and being shut down. And we have seen previous ransomware boasting about its increased performance. In the RaaS world, where attracting affiliates away from other ransomware service providers could be beneficial, being known to have the fastest file encryption 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 (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 I recall that during one of my appearances on Leo's Call 4 Help show in Toronto I used that ubiquitous whiteboard to diagram how RC4 worked. It's not complicated but it has an undeserved bad reputation due to its misuse.

It's a bit tricky to use correctly because the first 3K or so bytes it generates should be discarded. RC4 is a keyed pseudo-random byte generator with a sort of built-in entropy pool. When used as a cipher it generates a pseudo-random bitstream which is XORed with plaintext to create the ciphertext. 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. And after mucking up their original implementation, the WiFi Alliance, who keeps demonstrating their lack of genius, chose to abandon RC4 and switch to AES. AES is wonderful. I'm a big fan of the Rijndael cipher, but it's 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 a 0 remains the same, and any bit you XOR with a 1 is inverted. And it's so counterintuitive that simply using a pseudo-random bitstream to choose which bits to flip in a plaintext could convert it to uncrackable ciphertext. But it can. And it does. We've also talked about how tricky it can be to use XOR properly. The similar famous case is the one-time pad. If it's 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 is ever possible to know a plaintext for the matching ciphertext, they can be XORed to recover the key stream. And a somewhat more convoluted version of that mistake is what the South Korean researchers discovered. Here's a short relevant passage from their 23-page paper where they provide an overview of when they found and did:

Recently, many ransomware attacks have been found to use a hybrid encryption scheme that encrypts user files with a symmetric cipher, and stores the encryption keys used with an asymmetric cipher. 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 to decrypt the encrypted files. However, certain ransomware may use a self-developed encryption algorithm when encrypting files. If attackers cryptographically misconfigure the ransomware, 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 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, this is the first successful attempt at decrypting the Hive ransomware.

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:

1. *We identified the way in which Hive ransomware generates and stores a master key for victim files. Hive ransomware generates 10 megabits of random data which it uses as a master key. For each file to be encrypted, 1 megabit and 1 kilobit 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 is possible to extract the keystream used for encryption.

2. *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. We found that this random keystream was sufficiently guessable.*

3. *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.*

From this description we can be pretty certain that encryption performance was the goal. The improperly conceived ransomware generated a single static reusable 10-megabit XOR keystream and then chose various 1-megabit chunks at a per-file offset into that keystream for its encryption. In taking this approach, temping as it was, they broke the cardinal rule of XOR-based encryption, which is to **never** reuse the same keystream.

The bad news is, software can be updated and I'm sure that The Hive's ransomware technology will be updated immediately. 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.

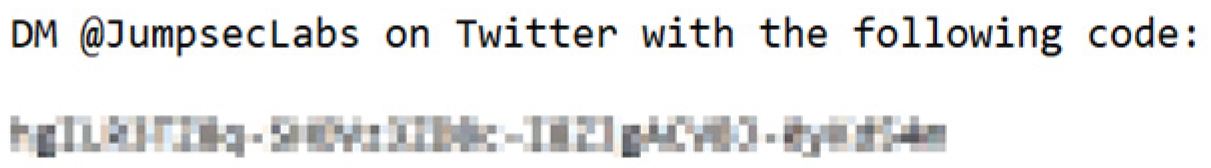
Still, this was very nice work and it demonstrates that we shouldn't simply assume that any random ransomware was properly designed to be truly uncrackable.

Un-Pixelating redacted text
Another fun story caught my eye this week. Dan Petro, the lead researcher at the security firm Bishop Fox titled his posting last week: *"Never, Ever, Ever Use Pixelation for Redacting Text"*
https://bishopfox.com/blog/unredacter-tool-never-pixelation

The issue here is the somewhat common practice of redacting text to make sections of it unreadable by "pixellating" 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.

Dan explains:

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, given a De Bruijn sequence of the correct font. 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 un-redact the following image:


DM @JumpsecLabs on Twitter with the following code:

[illegible]

And Dan then asks rhetorically “how could I refuse such a challenge”??

Work on un-redacting blurred or pixelated text has been done before, 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 the goal. He created a metric describing how closely the two matched, then guessed again. After he’d run through all possible combinations of what the first character or two might be, he chose the one which produced the most closely matching pixelation and then started working on the next character. And it worked.

Dan provides a demo of his algorithm searching and its animated GIF image is this week’s shortcut of the week. So it’s https://grc.sc/859 (or https://media.grc.com/misc/words.gif)

And the takeaway is... if you ever seriously want to redact text use a simple 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** way. 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.

No Internet For You!!

A small town in France was facing a mystery: From midnight until 3 AM 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 for managing the RF spectrum in France 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 cellphones 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, using a jamming device in France, as in most places, is illegal and carries a penalty of up to a 30,000 euros and 6 months in jail. And just in case anyone's wondering, they are also illegal in the US.

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. I think that they should just be required to hand over their phones, but apparently that's too confrontational.

If at first you don't succeed...
So it turns out that perhaps Adobe's Magento/Commerce Platform developers actually 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 Superbowl Sunday update... because they didn't fix the whole problem.

Very early (1:17am) 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 additional security protections necessary for CVE-2022-24086 and have released an update to address them (CVE-2022-24087). Adobe is not aware of any exploits in the wild for the issue addressed in this update (CVE-2022-24087).”

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 Abode's Commerce platform this is already old news to you. If not, this is one to fix immediately.

Ukrainian DDoS Attacks
Last Tuesday, in what had to be a plain and stark run up to Russia's designs on Ukraine, multiple institutions which are critical to Ukraine’s military and economy were hit with denial-of-service (DoS) attacks. The attacks triggered 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.

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 percent 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 3-way handshake whose purpose is to verify and establish a functioning two-way connection between the endpoints. Both sides exchange randomly chosen 32-bit SYNchronization packets which will be used to number the data that they subsequently exchange. And it's this 3-way handshake that prevents IP address spoofing. 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 are 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 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 multi-threaded nor particularly high performance. 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're built extremely complex and capable data-driven 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 web sites 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 everyday 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.

Sci-Fi

Yesterday, I pushed to finish the 3rd of the so-called "Bobiverse trilogy" so that I could report today. And I can now recommend it without hesitation.

All of our listeners who were telling me that I would love it, were right. As I had mentioned before, I was initially somewhat cooler and tentative about it. The author's writing style is more 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. By comparison, Dennis Taylor just jumps right in and starts telling the story. So while his style was initially a bit jarring to me, I wound up truly loving his story. It's also a lot of fun that "Bob" the primary character(s) in the books was himself an avid sci-fi fan, having read all of the same books and watched all of the same TV and movies as we, the readers of his story, have. 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.

There's now a 4th novel that's about twice the length of the largest of the previous three. And like the first three, it's available through both Kindle Unlimited and Audible, so that's next up for me while I await more from Ryk or Peter.

SpinRite News

InControl settled down quickly following last week's podcast announcement. 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. So that will be happening in a couple of 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 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 really is 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 — it just flashes by — but it allows SpinRite to quickly choose the fastest bulk data access method.

Then I realized that I could utilize that mini-benchmark to estimate the entire drive scan time and show that in the UI the same way that I had been showing it only after first deliberately performing a full benchmark, but without the full benchmark. I saw the result on Sunday, and it's somewhat astonishing what a huge difference just that little change makes: You start SpinRite, it discovers all of the drives attached to the machine, then takes you to the screen where you select which drive you want to run SpinRite on, and just as it has always shown the size of the drive, listed there beside each drive is the estimated time to scan that drive. It's just a small new feature, but nothing is more important.

A BGP Routing Attack

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 US dollars as a result of 407 fraudulent, intercepted and altered cryptocurrency transactions across a total of 325 KLAYswap customer wallets.

Such things have happened before, but the way this was engineered was diabolical and somewhat horrifying.

As we know, the success of most cryptocurrency heists can be traced back to some security failure on the part of the cryptocurrency exchange. Attackers might compromise the account of an employee who's using a weak password and no additional authentication factor, or bad guys might find a flaw in the platform's online server code, using that to steal funds from their customer's accounts. None of that happened here.

When any of us visit any sort of modern website, whether 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 with all browser-side scripting disabled are long since past. As we know, I was using NoScript as long as I possibly could, until I found myself needing to disable it more than not. So I finally gave up and switched to Gorhill's uBlock Origin. I sure hope that I never need to give that one up!

In any event, accepting a big 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 "developers.kakao.com", which was part of the KakaoTalk's developer infrastructure where KLAYswap hosted its official SDK which allowed 3rd-party apps to integrate with its services. This was all fine. Client browsers obtained their own version of JavaScript which allowed them to integrate with KLAYswap's back end services.

We've talked many time in the past about how Internet routing works. Or more specifically, about how routing tables function. In the IPv4 space, we start off with a 32-bit address. Part of the original genius of the designers of the Internet was to realize that if the Internet supported a 32-bit machine address space, and all of the machines associated with a single organization could have their addresses grouped together, 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, moving right to some degree, would be the same. The original designers called this common set of bits the "network" or a bit more formally, the "network prefix" and they realized that this could be used, globally, to route any packets whose leading (left end) bits were such-and-such to that entity's router for further distribution to the individual machines there.

Next, they realized that multiple organizations near each other, and being served by the same larger Internet Service Provider, might different 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 customer's networks. Then the ISP's routers could further sub-divide its larger, all inclusive network into its individual customers sub-networks.

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 Tier1 networks, 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 the each packets' address is compared to each entry in the router's routing table, looking for the longest network prefix in the table that matches the packet's destination IP. Once 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. This is what the Border Gateway Protocol provides. Routers that are connected to each other continually share their routing information. 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 *“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 makes this possible. And when you stop to think about it, the news of that new network needs to 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 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 has 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 before about Autonomous System, 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 packet traffic placed onto the Internet anywhere, whose network prefix matches any of the prefixes owned by that Autonomous System will have those packets routed to them.

KaKao.com Internet service provider has Autonomous System number 38099 and that provider owns 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, the remaining nine bits at the end on the right are IP addresses of 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 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, DREAMLINE CO. a major Internet carrier in South Korea with control over 1,793,723 IP addresses, managed to sneak a fraudulent BGP routing table update into the global network. 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-into inter-router BGP communications. 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 — snuck in a /24 route which specified one additional bit of 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 all this up were, of course, ready. They had a server ready to impersonate the original "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: https://developers.kakao.com/sdk/js/kakao.min.js

For a period of two hours, from 11:30 to 1:30, on February 3, 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, 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 OrbitBridge and FixedFloat, two cross-blockchain conversion services.

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 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 really ready for prime time.