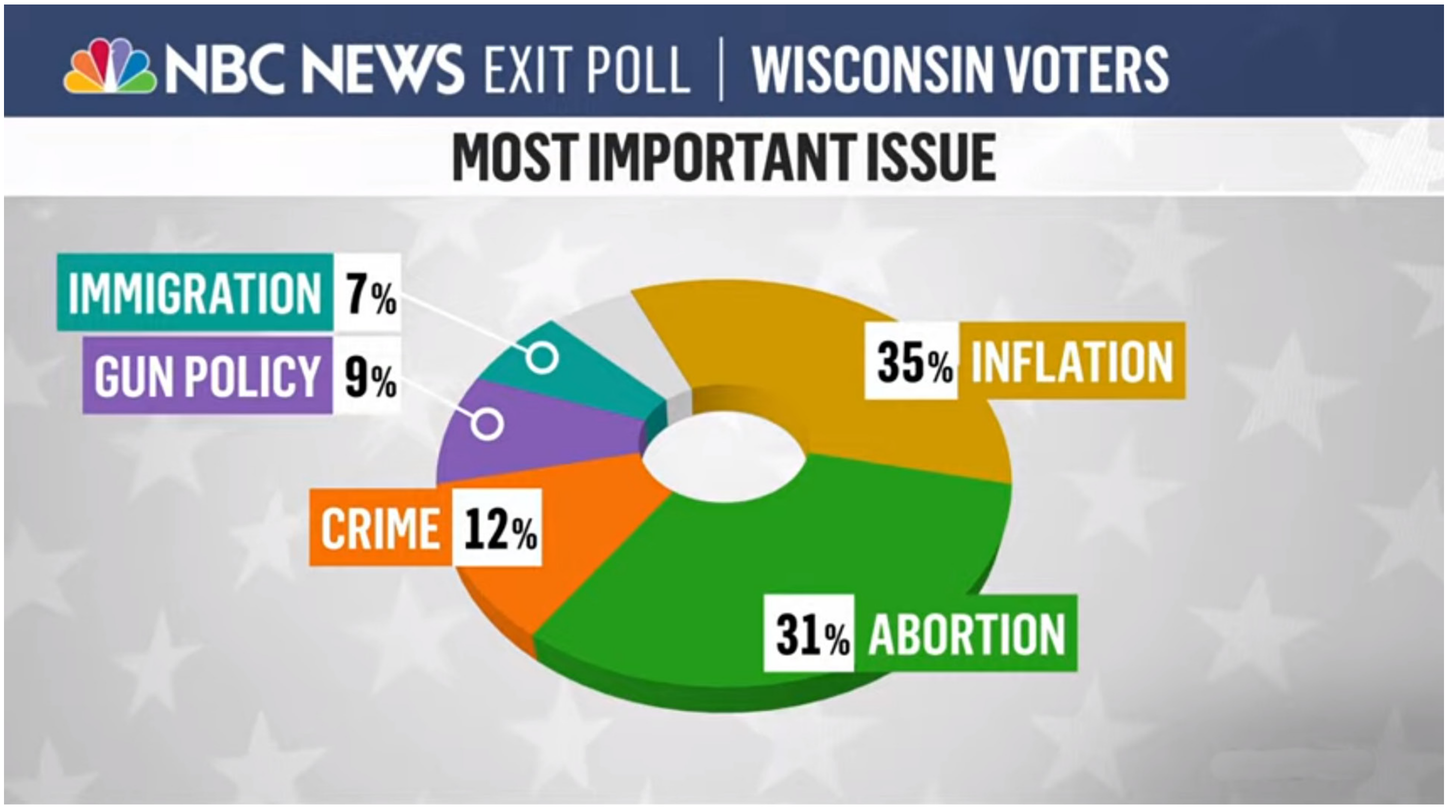
Security Now! #897 - 11-15-22
Memory-Safe Languages

This week on Security Now!

This week we have another event-filled Patch Tuesday retrospective. We look at a newly published horrifying automated host attack framework which script kiddies are sure to jump on. We have a welcome new feature for GitHib, crucial vulnerabilities in the LightSpeed web server, a spiritual successor to TrueCrypt and VeraCrypt for Linux, Australia's announcement of their intention to proactively attack the attackers, a controversial new feature in iOS 16.1.1, a couple more decentralized finance catastrophes, some miscellany and listener feedback. Then we'll finish by looking at a just published advisory from U.S.'s National Security Agency, our NSA, promoting the use of memory safe languages.

What's wrong with this chart?



Security News

Patch Tuesday review:
Last Tuesday was November's Patch Tuesday. We had security updates being released by Adobe, Microsoft, SAP, Android, VMware, Citrix, and others. In the case of Microsoft, 67 security flaws were fixed, including those two lingering “ProxyNotShell” 0-day vulnerabilities which have been plaguing on-site Exchange Server installations. There were also another four actively-exploited 0-days fixed by last week’s updates:

- an RCE in the JScript9 scripting language, detected in the wild by Google TAG team.
- a Mark-of-the-Web bypass known as “ZippyReads.”
- an Elevation of Privilege the Windows Print Spooler service, discovered by Microsoft.
- an EoP in the Windows CNG Key Isolation Service, also detected by Microsoft.

However, we didn't get past this month's patch Tuesday without the induction of some new headaches for enterprise admins. Since last Tuesday, Microsoft has been investigating the cause behind many reports (and many quick rollbacks of the month’s “improvements”) of enterprise domain controllers experiencing Kerberos sign-in failures and other authentication problems.

After Windows 2000, Kerberos replaced the creaky and never quite ready for prime time home-grown NT Lan Manager protocol (we’ve discussed NTLM’s many security problems through the years.) Kerberos was designed by MIT and first released the year after I first published SpinRite, yes, way back in 1988. So Kerberos has been around for quite a while. It’s a bullet-proof client server mutual authentication protocol that works. Well... at least it did for Windows until last week. I’m sure they’ll get it fixed. Microsoft has acknowledged the trouble saying:

After installing updates released on November 8, 2022 or later on Windows Servers with the Domain Controller role, you might have issues with Kerberos authentication. When this issue is encountered you might receive a Microsoft-Windows-Kerberos-Key-Distribution-Center Event ID 14 error event in the System section of Event Log on your Domain Controller with the below text. And, as a consequence...

- *Domain user sign-in might fail. This also might affect Active Directory Federation Services (AD FS) authentication.*
- *Group Managed Service Accounts (gMSA) used for services such as Internet Information Services (IIS Web Server) might fail to authenticate.*
- *Remote Desktop connections using domain users might fail to connect.*
- *You might be unable to access shared folders on workstations and file shares on servers.*
- *Printing that requires domain user authentication might fail.*

You know, so just your standard collapse following the application of a monthly round of patches and improvements. Thanks to a listener, Johan Mellberg (@infoball) who Tweeted about this, there’s a Reddit thread that explains what happened: Best practice is to disable the use of the very old RC4 cipher for Kerberos. But November’s trouble has hit those enterprises that did. This suggests that Microsoft did not test their November changes under current best practices.

https://www.reddit.com/r/sysadmin/comments/ypbpju/patch_tuesday_megathread_20221108/

I have a link in the show notes to the Reddit thread. It also documents the fixes to this which have worked for those who have implemented them. So that might be useful to some of our listeners. We tell everyone that patching as soon as possible can often be crucial. And we know that it can be. But it's also true, as we've just seen, that doing so might also cause major systemic failures throughout an organization. It's no wonder that those jobs are notoriously high stress.

Shennina Framework - Automating Host Exploitation with AI

This next bit of news falls under that category labelled "Not all things that can be done should be done." Unfortunately, a new, smart, host attack automation framework – yes, you heard that right, a host attack automation framework – was open sourced last Tuesday on GitHub. My first thought was "please don't make attacking hosts any easier for the script kiddies." But it's too late.

A cybersecurity engineer by the name of Mazin Ahmed explained that back in 2019 he and a fellow researcher, Khalid Farah participated in a security competition for developing offensive security tools. Not defensive, offensive. (And as we'll see a bit later on today, these might be of interest to Australia, but one thing at a time.) So, Mazin wrote:

I enjoy building security tools, and this competition was funded by HITB (Hack-in-the-Box) with a reward of $100,000 for the winners. I thought it would be an interesting challenge to work on as a side project.

I met my friend Khalid, he was also interested in winning this competition. We signed up, and once accepted, we started meeting regularly to build this project.

The Hack-In-The-Box CyberWeek AI Challenge had two categories: Host Exploitation or Malware Evasion. And unfortunately they chose the Host Exploitation option. He wrote:

Host exploitation sounds more relevant to our experience. The goal was to build a host exploitation framework using AI, and based on the concept of DeepExploit. The winning team should ideally prove the accuracy of the model, the improvement of the training and execution speed, and the technical features that have been added to the framework. We started experimenting with DeepExploit, and how it works, and we decided to start a new project based on the ideas we had on how to improve the tool.

This eventually ended up having us develop Shennina, a host exploitation framework. The project is 4 times faster than DeepExploit. We were excited about the results.

Shennina comes with a deception detection capability that detects if the machine being exploited is a Virtual Machine or Container, and then terminates post-exploitation once it's detected. This feature is powered by Metasploit modules. The Shenina Framework has qualified for the top 5 projects (out of 40 projects). We worked on developing the tool further to prepare for our final demo that will be live at HITB Abu Dhabi 2019.

Unfortunately, the rules of the competition and the judging criteria changed during the demo day. We enjoyed HITB CyberWeek 2019. It was an amazing journey, and I enjoyed building the Shennina Framework. I also presented my research on JWT hacking at that time [JWT are JSON Web Tokens] *it was a busy week :)*

We are planning to open-source the project and the experiment. There are no plans for further maintaining Shennina in the near future.

And that open sourcing of Shannina just came to pass. The powerful Shennina Host Exploitation framework is now up on GitHub. They do comment under the FAQ question *"Why are we solving this problem with AI?"* that *"The problem should be solved by a hash tree without using "AI", however, the HITB Cyber Week AI Challenge required the project to find ways to solve it through AI."*

Extracting some of the more interesting bits from the project's Abstract, they explain:

Shennina is an automated host exploitation framework. The mission of the project is to fully automate the scanning, vulnerability scanning/analysis, and exploitation using Artificial Intelligence. Shennina is integrated with Metasploit and Nmap for performing the attacks, as well as being integrated with an in-house Command-and-Control Server for exfiltrating data from compromised machines automatically.

Why oh why post this tool publicly?? They continue, writing...

Shennina scans a set of input targets for available network services, uses its AI engine to identify recommended exploits for the attacks, and then attempts to test and attack the targets. If the attack succeeds, Shennina proceeds with the post-exploitation phase.

The AI engine is initially trained against live targets to learn reliable exploits against remote services. Shennina also supports a heuristics mode for identifying recommended exploits. The documentation can be found in the Docs directory within the project.

Features:

- *Automated self-learning approach for finding exploits.*
- *High performance using managed concurrency design.*
- *Intelligent exploits clustering.*
- *Post exploitation capabilities.*
- *Deception detection.*
- *Ransomware simulation capabilities.*
- *Automated data exfiltration.*
- *Vulnerability scanning mode.*
- *Heuristic mode support for recommending exploits.*
- *Windows, Linux, and macOS support for agents.*
- *Scriptable attack method within the post-exploitation phase.*

- *Exploits suggestions for Kernel exploits.*
- *Out-of-Band technique testing for exploitation checks.*
- *Automated exfiltration of important data on compromised servers.*
- *Reporting capabilities.*
- *Coverage for 40+ TTPs (tactics, techniques, and procedures) within the MITRE ATT&CK Framework.*
- *Supports multi-input targets.*

That's just great. It's bad enough that the quite sophisticated and powerful NMAP scanner and Metasploit exploit frameworks, both which have been developing and evolving over many years, already allow sophisticated hackers to scan network regions and exploit them. But now this process has been automated with a somewhat smart front end that incorporates NMAP and Metasploit to create a turn-key all-in-one automated system that can be launched by junior hackers who would never have the ability to create such a tool themselves. Buckle up.

GitHub's welcome new feature

I do have a nice bit of news about GitHub. They've added a new "private vulnerability reporting" feature. It allows security researchers to report vulnerabilities in **public** repositories to their respective owners via a **private** channel. In other words, this new feature will allow sensitive security-related reports to be filed to repo owners without having to file a publicly-viewable entry in a repo's Issues Tracker. That seems like a really good idea.

https://github.blog/changelog/2022-11-09-privately-report-vulnerabilities-to-repository-maintainers/

Three LightSpeed vulnerabilities

There are presently 1.9 million unique instances of the LightSpeed web server providing services on the Internet. LightSpeed is ranked the 6th most popular web server which give it a 2% share of all Web Servers globally. The problem is, after some research by Palo Alto Networks' Unit 42, any editions of either the free open source edition, known as OpenLightSpeed, or the enterprise version which is just named LightSpeed, which have not been updated in the past month, need to be patched IMMEDIATELY.

The Unit 42 research team took a close look at the source code for OpenLightSpeed and discovered three different vulnerabilities in the web server. The three vulnerabilities have been confirmed to similarly affect the non-open source enterprise version. By chaining and exploiting the vulnerabilities, adversaries could compromise the web server to gain fully privileged remote code execution. The vulnerabilities discovered include:

- A Remote Code Execution (CVE-2022-0073) rated High severity (CVSS 8.8)
- A Privilege Escalation (CVE-2022-0074) rated High severity (CVSS 8.8)
- A Directory Traversal (CVE-2022-0072) rated Medium severity (CVSS 5.8)

A little over a month ago, Unit 42 responsibly disclosed their discovery of these vulnerabilities to the server’s publisher and maintainer LiteSpeed Technologies with suggested remediation. That occurred on Oct. 4th. LiteSpeed Technologies released a patch version (v1.7.16.1) two weeks later on Oct. 18th. So, individuals and organizations using OpenLiteSpeed versions 1.5.11 through 1.7.16 need to be updated immediately to 1.7.16.1. And LiteSpeed versions 5.4.6 through 6.0.11 need to move to 6.0.12 as soon as possible. Since there are 1.9 million of these servers in use, and since the fixes will be reflected in the open source version, bad guys will have no trouble seeing what was fixed and designing an exploit chain that Unit 42 warned of. Since the location of all LightSpeed web servers are known to Shodan and other Internet scanners, these servers can be found and attacked.

Shufflecake: Plausible deniability encrypted Linux volumes

Last Thursday, the guys at Kudelski Security Research open sourced an interesting utility that I imagine may be of interest to some of our listeners. Here's how Kudelski explains their creation:

Today we are excited to release Shufflecake, a tool aimed at helping people whose freedom of expression is threatened by repressive authorities or dangerous criminal organizations, in particular: whistleblowers, investigative journalists, and activists for human rights in oppressive regimes. Shufflecake is FLOSS (Free/Libre, Open Source Software). Source code in C is available and released under the GNU General Public License v3.0 or superior.

Shufflecake is a tool for Linux that allows creation of multiple hidden volumes on a storage device in such a way that it is very difficult, even under forensic inspection, to prove the existence of such volumes. Each volume is encrypted with a different secret key, scrambled across the empty space of an underlying existing storage medium, and indistinguishable from random noise when not decrypted. Even if the presence of the Shufflecake software itself cannot be hidden – and hence the presence of secret volumes is suspected – the number of volumes is also hidden. This allows a user to create a hierarchy of plausible deniability, where “most hidden” secret volumes are buried under “less hidden” decoy volumes, whose passwords can be surrendered under pressure. In other words, a user can plausibly “lie” to a coercive adversary about the existence of hidden data, by providing a password that unlocks “decoy” data. Every volume can be managed independently as a virtual block device, i.e. partitioned, formatted with any file system of choice, and mounted and dismounted like a normal disc. The whole system is very fast, with only a minor slowdown in I/O throughput compared to a bare LUKS-encrypted disk, and with negligible waste of memory and disk space.

You can consider Shufflecake a “spiritual successor” of tools such as Truecrypt and Veracrypt, but vastly improved. First of all, it works natively on Linux, it supports any file system of choice, and can manage up to 15 nested volumes per device, making deniability of the existence of these partitions highly plausible.

https://shufflecake.net/

https://research.kudelskisecurity.com/2022/11/10/introducing-shufflecake-plausible-deniability-for-multiple-hidden-filesystems-on-linux/

Australia has decided to get proactive!
Earlier I mentioned that Australia might have an interest in offensive as opposed to defensive cyber weaponry. As we have previously noted here, Australia has recently been having outsized problems with cyberattacks; they've had a slew of them and they're apparently getting a bit tried of them. So the straw that may have broken the camel's back, is the most recent ransomware cyberattack on an Australian private insurance provider named Medibank.

Medibank said that attackers broke into their network, stole internal files—including sensitive personal and healthcare details on 9.7 million Australians—then encrypted their files, and demanded a ransom. When Medibank refused to pay, the ransomware gang, going by the name of BlogXX, and believed to be a spin-off of REvil, began leaking some of Medibank's patient records to intimidate Medibank and create public pressure to pay the ransom. One of the group's first leaks was a file labeled "abortions." Not surprisingly, the BlogXX group's actions have galvanized the Australian government and Australia has now vowed to go on the offensive. The group's activities have been met with outrage and significant political attention from the highest levels of the Australian government.

Clare O'Neil, an Australian Minister for Home Affairs and Minister for Cyber Security posted a two and a half minute, very clear interview she gave to ABC Insider. Listen to what Clare has to say:

Mark Dreyfus and I announced
with the Deputy Prime Minister yesterday
49.5K views
0:03 / 2:24

https://twitter.com/ClareONeilMP/status/1591591081526775809

Clare's statement was released on the same day the Australian Federal Police issued a statement formally identifying the Medibank hackers as being located in Russia, which comes as a surprise to no one. Australian Federal Police commissioner, Reece Kershaw, said in his statement: *"We believe we know which individuals are responsible but will not be naming them. What I will say is that we will be holding talks with Russian law enforcement about these individuals."*

Despite an explosion in ransomware attacks, and as we noted last week, more than $1.2 billion dollars in payments during 2021 alone, the cases where law enforcement agencies fought back and hacked the hackers have been rare. There have been a few success stories, such as the DOJ managing to recover Colonial Pipeline's payment to the Darkside group, and both US CyberCom and the FBI hacking REvil's servers following the Kaseya-based attacks. In both cases, Darkside and REvil's operations were shut down following these proactive “offensive” operations. And those effective responses did appear to give attackers at least a bit of pause. Yet the problem obviously persists. So it will be very interesting to see what might come from Australia's new “we're not going to take this lying down anymore” stance.

Until now, the bad guys have had the advantage that they don’t care about breaking cyber intrusion laws, whereas global governments have been hampered by their need to play by the rules and act within the law. So......... what does Australia mean exactly when they say they’re going to be going after the bad guys? Are they going to break cyber intrusion laws to get that done? I guess we’ll see.

In addition to the announcement of the new offensive taskforce, O'Neil celebrated Medibank's decision to **not** capitulate to the demands of its attackers and suggested the government might look into a law which bans payments made to ransomware and data extortion attackers altogether, with the hope that this would strangle the financial incentive behind most attacks.

Apple's iOS 16.1.1 everyone file sharing time-limits to 10 minutes in China

In what observers are saying looks like Apple capitulating to demands from China, the latest iOS v16.1.1 and the next beta v16.2 have added a new timeout to the AirDrop sharing feature which limits the “Share with Everyone” option to 10 minutes. The reason Chinese influence is suspected is that, for the time being at least, this only affects iPhones purchased in China — the new restriction is tied to the hardware.

However, it's also worth noting that many people outside of China have celebrated the addition of this restriction as an extremely useful security improvement, since leaving “Everyone Sharing” inadvertently enabled could represent a significant security risk.

Interestingly, this is not the first time Apple has customized some aspects of their iPhone offerings for China. For example, believe it or not, the Taiwanese flag emoji is not available on iPhones sold in China. (Talk about being petty.) Apple also uses the hardware tied method to limit the volume level of its devices in EU, as required by law.

In the case of the 10 minute time limit, the motivation appears to be due to the fact that Chinese protesters had been using the feature to spread posters in opposition to Xi Jinping and the Chinese government.

And as for those who think that this is a useful feature, Bloomberg reported that Apple has plans to make this new AirDrop option available globally next year. (Though presumably optional as opposed to mandatory.)

A couple of Decentralized Finance notes because I can’t help myself:
The DeFi platform Pando said it was the target of a hack Saturday before last when a threat actor attempted to steal more than $70 million worth of cryptocurrency from the platform's wallets. Pando said that it managed to hold onto $50 million of the stolen funds, but the attacker successfully stole more than $21.8 million of its funds. They said the hacker used an Oracle attack against one of its protocols and is still hoping to negotiate with the attacker to return some of the stolen funds.

I wondered whether that had happened, since now that was about 10 days ago. But so far it doesn't appear to be the case. I found Pando's Twitter thread where they announced the hack of their system, then over the course of several days, they followed the stolen funds as they were anonymously moved from one currency or exchange to another. What a mess, watching nearly $22 million dollars moving around and being able to do nothing about it.

Also, the DeFi platform DFX Finance suffered a crypto-heist, reporting the loss of $4.185 million worth of cryptocurrency assets following an attack on its platform late last week. The company said the incident was identified as a reentrancy attack, which sounds like another of those increasingly common *“oh shoot, we failed to consider that possibility when we were designing our platform; there went more than $4 million.”*

I'm sure that everyone who follows this podcast by now knows that no one should rely upon what over and over again appear to be half-baked, poorly conceived, decentralized finance systems. And it clearly doesn't matter how well intended the creators and operators of these systems are. Willie Sutton explained that the reason he robbed banks was because that's where the money was. For many people, money is the great motivator. That’s the singular thing that’s driving the ransomware attacks. They couldn’t care less about anyone data. They want money. And these DeFi systems are apparently sloshing around in WAY more money than they are responsible enough to manage.

Miscellany

“The Helm” was unable to survive COVID-19.
“The Helm”, a past sponsor of TWiT and of this Security Now! podcast, recently announced that it would be unable to continue operating past the end of the year. So they provided their users with 60 days notice and had already stopped accepting new subscriptions.

They explained that in 2019 they had relocated from Mexico to China in order to improve their scalability and to begin working on a v2 system. But then COVID his and their scheduling times more than doubled.

I’m bringing this up here for two reasons: First, anyone who had obtained their own domain that was hosted at The Helm’s registrar, you may wish to move it to someone else, like Hover (my own registrar before they also became a TWiT sponsor). Current users will have only until the end of the year to get their domain moved.

The second reason is that on their FAQ page, in response to the question “What can I do with the Helm server after you shutdown?” they wrote:

We are working on a firmware update for Helm servers to be converted to Linux servers running Armbian. We will provide documentation for this conversion, along with pointers to some guides for running mail and file sharing services if you would like to continue using your Helm. We expect this firmware update to be available in early to mid December.

https://support.thehelm.com/hc/en-us/articles/10831596925203-Helm-Shutdown-FAQdown

So, just a heads-up to anyone listening that some action will need to be taken between now and the end of the year. If you have a domain there you'll want to get it moved. And then be on the lookout for new firmware that will allow you to continue using your Helm server independently.

Elon meets Twitter:
There's no way for a weekly podcast to possibly follow the insanity that has befallen Twitter not that Elon has taken the reins. But one of our listeners, Ed Ross, put me onto a Twitter thread from yesterday that gave me a chuckle and which I thought our listeners would appreciate. So, Elon Announce-Tweets:

Elon Musk / @elonmusk

Part of today will be turning off the "microservices" bloatware. Less than 20% are actually needed for Twitter to work!

And after the results of pulling those plugs results in a new catastrophe-du-jur, then someone named Zach Silverberg Tweets:

zach silberberg / @zachsilberberg

Apparently they didnt turn off 2-factor authentication but they DID turn off the service that sends you the 2-factor authentication code. So, if you log out and try to log in with an authentication code you simply wont receive one.

Closing The Loop

Daniel Smith / @danielbsmith

This could be a new SN saying, and he quoted someone on Twitter named Lewis Locklock who replied to someone else: ***"There's nothing more permanent than a temporary solution."***

Dennis Keefe, Financial Coach @prosperuscoach

Hi Steve, wanted to let you know the sponsor link for Drata seems to send you to a 404. I'd hate for y'all to lose money. Love the show! https://drata.com/twit

Neil Baldridge / @neilbaldridge

Hi Steve. Just a quick "Thank You" for passing along your recommendation of the Silver Ships series. I'm just into book 6 and find the stories to be a great read. I'm trying to pace myself but without much success. It's so nice to find an extensive series that is so enjoyable.

I agree totally. The series author, Scott Jucha (you-ha) suggests that after finishing book 13, the reader should switch over to the 4-book "Pyreans" series. So that's what I did. I'm nearly finished with those four books, so I can attest that they are every bit as delicious as the Silver Ships series. Those four books trace the history of an entirely different Earth descended colony and its characters are every bit as interesting and alive as those in the Silver Ships. And what's intriguing is that by this point the reader has grown to intimately know two very different groups of human colonists, and Scott has explained that the two threads are going to be merging once the reader resumes with the Silver Ships series in book 14. I've come to know this diverse set of characters so well that I cannot **wait** to see what Scott has in store for when they meet! It promises to be so cool!

Tiemo Kieft / @TiemoKieft

Hey Steve, just wanted to respond to the whole OpenSSL debacle. You mentioned not many web servers accept client certs, this is not the case though. Web servers serving big sites accept client certs because CDNs and DDOS protection services like CloudFlare use a client cert to assert their identity to web servers they proxy. This feature is used to ensure that the server only responds to requests proxied through CloudFlare. CloudFlare calls this feature 'Authenticated Origin Pulls '.

That certainly makes sense as a reason for web servers to solicit and accept client certs. I guess I still doubt that the number is huge, but I agree that it would definitely be another reason for some increased vulnerability to the recent OpenSSL v3.x flaw.

Nariman Aga-Tagiyev / @aganariman

Dear Steve, after listening to episode 892 I decided to replace my RaspberryPi home VPN server with a ZimaBoard which I ordered online right away. Following getting started instructions from the CasaOS I created a new user and a complex password using its WebUI on port 80. But I was quite concerned when I noticed that the credentials set are only for the CasaOS Web application. The board came with more than 10 default users on its DebianOS which still had their default passwords after the initial configuration steps. I could login over SSH using casaos casaos username and password. The root's default password is also casaos. There are no warnings or instructions on the ZimaBoard and CasaOS websites about how to remove default users and passwords from the board. I'm very concerned that if those boards are installed as a router according to this article:
https://www.zimaboard.com/application/hardware_router
... anyone will be able to SSH to it if owners don't proactively delete all debian users.
Thank you for your podcasts, your active listener for a few years already.

SpinRite

As for SpinRite, I'm getting very very close to the first pre-release alpha of 6.1. I ended up rewriting a bunch of the command-line parser to make drive selection extremely flexible and powerful. Then, as planned, I implemented the final verification for SpinRite by arranging to take a hash of an entire drive span over which SpinRite would run in order to absolutely verify that not a single bit had been changed across all of SpinRite's machinations with the drive. And I discovered to my surprise that my AHCI driver would, in a very low but still non-zero percentage of 16-megabyte transfers, occasionally not properly read the drive's data.

To make sure that it wasn't the brand new hashing system that was at fault for this, I tried the same thing with an IDE drive using SpinRite's new Bus Mastering DMA driver, and that worked perfectly. So I know that my confidence testing code is working and that the new Bus Mastering driver is also working perfectly. But not so the new AHCI driver. This happened Sunday evening and yesterday morning, so I haven't had a second yet to dig into this. My guess is that something is going on with my end-of-transfer status testing where I'm shutting down the controller before it may have finished transferring everything into RAM. Since it's transferring data asynchronously in the background it's definitely necessary to make sure it's finished. The good news is, thanks to now being able to take a hash of everything SpinRite does, I'm able to quickly test any changes I make.

So, I'm anxious to get back in there tonight to see what's going on. And I expect to be able to say next week that I'm finished with this round of SpinRite development and that it's ready for its final round of testing.

Memory-Safe Languages

Executive summary
Modern society relies heavily on software-based automation, implicitly trusting developers to write software that operates in the expected way and cannot be compromised for malicious purposes. While developers often perform rigorous testing to prepare the logic in software for surprising conditions, exploitable software vulnerabilities are still frequently based on memory issues. Examples include overflowing a memory buffer and leveraging issues with how software allocates and deallocates memory. Microsoft revealed at a conference in 2019 that from 2006 to 2018, 70 percent of their vulnerabilities were due to memory safety issues. Google also found a similar percentage of memory safety vulnerabilities over several years in Chrome. Malicious cyber actors can exploit these vulnerabilities for remote code execution or other adverse effects, which can often compromise a device and be the first step in large-scale network intrusions.

Commonly used languages, such as C and C++, provide a lot of freedom and flexibility in memory management while relying heavily on the programmer to perform the needed checks on memory references. Simple mistakes can lead to exploitable memory-based vulnerabilities. Software analysis tools can detect many instances of memory management issues and operating environment options can also provide some protection, but inherent protections offered by memory safe software languages can prevent or mitigate most memory management issues.

NSA recommends using a memory safe language when possible. While the use of added protections to non-memory safe languages and the use of memory safe languages do not provide absolute protection against exploitable memory issues, they do provide considerable protection. Therefore, the overarching software community across the private sector, academia, and the U.S. Government have begun initiatives to drive the culture of software development towards utilizing memory safe languages.

The memory safety problem
How a software program manages memory is core to preventing many vulnerabilities and ensuring a program is robust. Exploiting poor or careless memory management can allow a malicious cyber actor to perform nefarious acts, such as crashing the program at will or changing the instructions of the executing program to do whatever the actor desires. Even un-exploitable issues with memory management can result in incorrect program results, degradation of the program's performance over time, or seemingly random program crashes.

Memory safety is a broad category of issues related to how a program manages memory. One common issue is called a "buffer overflow" where data is accessed outside the bounds of an array. Other common issues relate to memory allocation. Languages can allocate new memory locations as a program is executing and then deallocate the memory, also called releasing or freeing the memory, later when the memory is no longer needed. But if this is not done carefully by the developer, new memory may be allocated again and again as the program executes. Consequently, memory is not always freed when it is no longer needed, resulting in a memory leak that could cause the program to eventually run out of available memory. Due to logic errors, programs can also attempt to use memory that has been freed, or even free memory that has already been freed. Another issue can arise when languages allow the use of a variable that has not been initialized, resulting in the variable using the value that was previously set at that location in memory. Finally, another challenging issue is called a race condition. This issue can occur when a program's results depend on the order of operation of two parts of the program accessing the same data. All of these memory issues are much too common occurrences.

By exploiting these types of memory issues, malicious actors—who are not bound by normal expectations of software use—may find that they can enter unusual inputs into the program, causing memory to be accessed, written, allocated, or deallocated in unexpected ways. In some cases, a malicious actor can exploit these memory management mistakes to access sensitive information, execute unauthorized code, or cause other negative impacts. Since it may take a lot of experimenting with unusual inputs to find one that causes an unexpected response, actors may use a technique called "fuzzing" to either randomly or intelligently craft multitudes of input values to the program until one is found that causes the program to crash. Advances in fuzzing tools and techniques have made finding problematic inputs easier for malicious actors in recent years. Once an actor discovers they can crash the program with a particular input, they examine the code and work to determine what a specially crafted input could do. In the worst case, such an input could allow the actor to take control of the system on which the program is running.

Memory safe languages
Using a memory safe language can help prevent programmers from introducing certain types of memory-related issues. Memory is managed automatically as part of the computer language; it does not rely on the programmer adding code to implement memory protections. The language institutes automatic protections using a combination of compile time and runtime checks. These inherent language features protect the programmer from introducing memory management mistakes unintentionally. Examples of memory safe language include C#, Go, Java, Ruby, Rust, and Swift. Even with a memory safe language, memory management is not entirely memory safe. Most memory safe languages recognize that software sometimes needs to perform an unsafe memory management function to accomplish certain tasks. As a result, classes or functions are available that are recognized as non-memory safe and allow the programmer to perform a potentially unsafe memory management task. Some languages require anything memory unsafe to be explicitly annotated as such to make the programmer and any reviewers of the program aware that it is unsafe. Memory safe languages can also use libraries written in non-memory safe languages and thus can contain unsafe memory functionality. Although these ways of including memory unsafe mechanisms subvert the inherent memory safety, they help to localize where memory problems could exist, allowing for extra scrutiny on those sections of code. Languages vary in their degree of memory safety instituted through inherent protections and mitigations. Some languages provide only relatively minimal memory safety whereas some languages are very strict and provide considerable protections by controlling how memory is allocated, accessed, and managed. For languages with an extreme level of inherent protection, considerable work may be needed to simply get the program to compile due to the checks and protections.

Memory safety can be costly in performance and flexibility. Most memory safe languages require some sort of garbage collection to reclaim memory that has been allocated, but is no longer needed by the program. There is also considerable performance overhead associated with checking the bounds on every array access that could potentially be outside of the array.

Alternatively, a similar performance hit can exist in a non-memory safe language due to the checks a programmer adds to the program to perform bounds checking and other memory management protections. Additional costs of using non-memory safe languages include hard-to-diagnose memory corruption and occasional program crashes along with the potential for exploitation of memory access vulnerabilities. It is not trivial to shift a mature software development infrastructure from one computer language to another. Skilled programmers need to be trained in a new language and there is an efficiency hit when using a new language. Programmers must endure a learning curve and work their way through any "newbie" mistakes. While another approach is to hire programmers skilled in a memory safe language, they too will have their own learning curve for understanding the existing code base and the domain in which the software will function.

Application security testing
Several mechanisms can be used to harden non-memory safe languages to make them more memory safe. Analyzing the software using static and dynamic application security testing (SAST and DAST) can identify memory use issues in software. Static analysis examines the source code to find potential security issues. Using SAST allows all of the code to be examined, but it can generate a considerable number of false positives through identifying potential issues incorrectly. However, SAST can be used throughout the development of the software allowing issues to be identified and fixed early in the software development process. Rigorous tests have shown that even the best-performing SAST tools only identify a portion of memory issues in even the simplest software programs and usually generate many false positives.

In contrast to SAST, dynamic analysis examines the code while it is executing. DAST requires a running application. This means most issues will not be identified until late in the development cycle, making the identified problem more expensive to fix and regressively test. DAST can only identify issues with code that is on the execution path when the tool is run, so code coverage is also an issue. However, DAST has a much lower percentage of false positives than SAST. Issues such as a memory leak can be identified by DAST, but the underlying cause of the memory issue may be very difficult to identify in the software.

Neither SAST nor DAST can make non-memory safe code totally memory safe. Since all tools have their strengths and weaknesses, it is recommended that multiple SAST and DAST tools be run to increase the chances that memory or other issues are identified. Working through the issues identified by the tools can take considerable work, but will result in more robust and secure code. Vulnerability correlation tools can intake the results from multiple tools and integrate them into a single report to simplify and help prioritize analysis.

Anti-exploitation features
The compilation and execution environment can be used to make it more difficult for cyber actors to exploit memory management issues. Most of these added features focus on limiting where code can be executed in memory and making memory layout unpredictable. As a result, this reduces a malicious actor's opportunities to use the exploitation tradecraft of executing data as code and overwriting a return address to direct program flow to a nefarious location.

Leveraging options, such as Control Flow Guard (CFG), will place restrictions on where code can be executed. Similarly, Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP) add unpredictability to where items are located in memory and prevent data from being executed as code. Bypassing ASLR and DEP is not insurmountable to a malicious actor, but it makes developing an exploit much more difficult and lowers the odds of an exploit succeeding. Anti-exploitation features can help mitigate vulnerabilities in both memory safe and non-memory safe languages.

The path forward
Memory issues in software comprise a large portion of the exploitable vulnerabilities in existence. NSA advises organizations to consider making a strategic shift from programming languages that provide little or no inherent memory protection, such as C/C++, to a memory safe language when possible. Once again, some examples of memory safe languages include C#, Go, Java, Ruby, Rust and Swift. Memory safe languages provide differing degrees of memory usage protections, so available code hardening defenses, such as compiler options, tool analysis, and operating system configurations, should be used for their protections as well. By using memory safe languages and available code hardening defenses, many memory vulnerabilities can be prevented, mitigated, or made very difficult for cyber actors to exploit.?

I wanted everyone to hear and consider that, and for those of our listeners who may be in positions where a choice of implementation language can be made for applications where the program's security and integrity would be important, to consider perhaps breaking with the status quo and choosing to bite the bullet and make a switch to a more secure programming language and environment.

Take the relatively new Rust language for example, since its first stable release in January 2014, Rust has been adopted by companies including Amazon, Discord, Dropbox, Facebook, Mozilla, Google, and Microsoft.

And if changing to a memory-safe language cannot be done for whatever reason, at least consider the idea of adding the SAST and DAST – static and dynamic application security testing – to existing project development lifecycle.

All of that could really pay off.