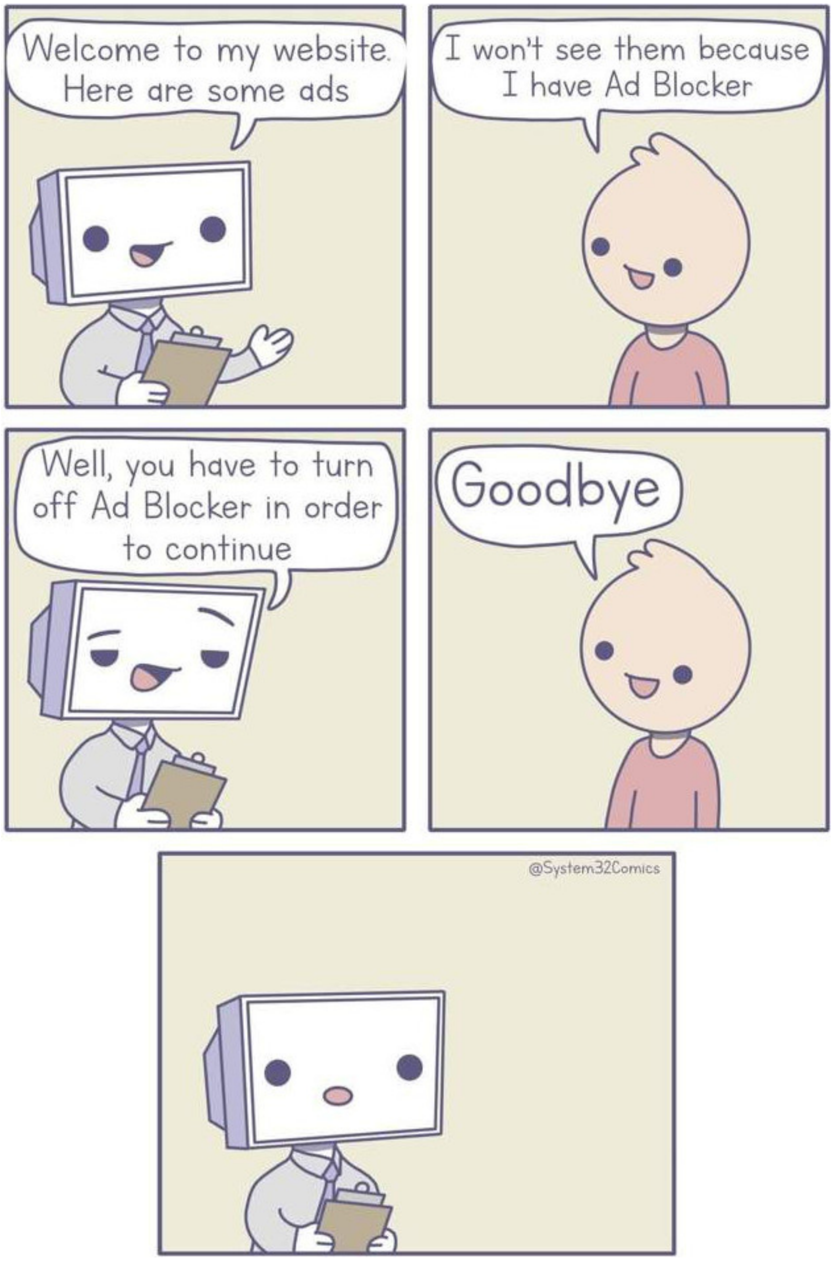
Security Now! #827 - 07-13-21
REvil's Clever Crypto

This week on Security Now!

The past week has been dominated by the unimaginable mess that Microsoft has created with what have become multiple failed attempts to patch the two "PrintNightmare" flaws... and the continuing "Cleanup on Isle 5" following what is widely regarded as the single most significant ransomware supply-chain attack event ever. So, today we first catch up on the still sadly relevant "PrintNightmare" from which the industry has been unable to awaken. We'll cover a few more bits of security news. Then, as planned, we'll take a deep dive into the detailed operation of the REvil / Sodinokibi malware's cryptographic design.


Welcome to my website. Here are some ads
I won't see them because I have Ad Blocker
Well, you have to turn off Ad Blocker in order to continue
Goodbye
@System32Comics

Security News

The "PrintNightmare" Continues...

First of all, I don't think that last week's podcast was over before I began receiving helpful Tweets from listeners letting me know that it was no longer true that the Windows Print Spooler service could be stopped and that applications would then be able to print directly to the system's printers. Tweeters were informing me that that wasn't true, and they were correct.

I dug into this a bit and saw that Windows printers still show the option to print directly to the printer rather than to go through the spooler. So I thought, okay, so printing directly needs to be chosen. But no, that doesn't work anymore either. For whatever reason, the Windows Print Spooler service DOES need to be running for anything to print. And this has apparently been true for some time, since I tried this on my Windows 7 machine and no printers appear, under any circumstances, when the Print Spooler service is not running.

So, for any machines that may actually need to be able to print something, disabling the vulnerable local printer spooler service will not be a useful workaround.

As we know, and described last week, two different problems have been discovered in the print spooler. There's a remote code execution (RCE) problem, and a Local Privilege Escalation (LPE) problem. What Microsoft fixed a month ago was only one of the two. They fixed the local privilege escalation problem. That's good, because escalation of local privilege is an extremely valuable capability for malware to obtain. Obtaining execution of attacker-provided code on a modern OS is one thing. But any serious exploitation of the machine often requires more system privileges than the typical user has. And this, of course, is all by design. So, last month's Patch Tuesday resolved the local privilege escalation. Unfortunately, it failed to resolve the other trouble, which was a remote code execution vulnerability.

As we also explained last week, it turns out that over the years, while Windows was quietly requiring the print spooler service to always be running, it was also adding some fancy new on-the-fly printer driver installation options.

In a big networked office environment, what happens if your local Windows client, the machine you're perched in front of, doesn't currently contain a driver for some printer in another on-campus building to which you want to print? Wouldn't it be slick if Windows could see that it's missing a needed driver for that printer, go find it somewhere, install it into your local machine for you, and then print to that remote printer without all that usual muss and fuss?

In this case, in answer to our often posed rhetorical question "WHAT could possibly go wrong?" we learn that bad guys have figured out how to trick Windows into downloading their malware by disguising it as a printer driver and saying to Windows the equivalent of "Oh no! We need this printer driver now!" Microsoft describes this capability, which was added way back in Windows 2000, as Point and Print:

> Point and Print is a term that refers to the capability of allowing a user on a Windows 2000 and later client to create a connection to a remote printer without providing disks or other installation media. All necessary files and configuration information are automatically

downloaded from the print server to the client.

Point and Print technology provides two methods by which you can specify files that should be sent from the print server to the client machine:

- Files can be associated with a printer driver. These files are associated with every print queue that uses the driver.
- Files can be associated with individual print queues.

For more information, see Supporting Point and Print During Printer Installations.

When a user application calls the AddPrinterConnection function (described in the Microsoft Windows SDK documentation) to make a printer connection, the following operations are performed:

- Driver-associated and queue-associated files are downloaded from the print server to the client.
- Current values of the printer's configuration parameters, which are stored in the server's registry under the printer's key, are downloaded to the client.

It’s not the Point and Print service that’s directly being exploited with malicious intent, but rather the operating system supported underpinnings — specific API calls — that are used by the Point and Print service to achieve this background magical printer driver installation.

The CERT Coordination Center’s vulnerability note for VU#383432 is titled: “Microsoft Windows Print Spooler allows for RCE via AddPrinterDriverEx()” and it explains:

The Microsoft Windows Print Spooler service fails to restrict access to functionality that allows users to add printers and related drivers, which can allow a remote authenticated attacker to execute arbitrary code with SYSTEM privileges on a vulnerable system.

The **RpcAddPrinterDriverEx()** function is used to install a printer driver on a system. One of the parameters to this function is the DRIVER_CONTAINER object, which contains information about which driver is to be used by the added printer. The other argument, dwFileCopyFlags, specifies how replacement printer driver files are to be copied. An attacker can take advantage of the fact that any authenticated user can call **RpcAddPrinterDriverEx()** and specify a driver file that lives on a remote server. This results in the Print Spooler service spoolsv.exe executing code in an arbitrary DLL file with SYSTEM privileges.

Note that while original exploit code relied on the **RpcAddPrinterDriverEx** to achieve code execution, an updated version of the exploit uses **RpcAsyncAddPrinterDriver** to achieve the same goal. Both of these functions achieve their functionality using **AddPrinterDriverEx**.

While Microsoft has released an update for CVE-2021-1675, it is important to realize that this update does NOT protect against public exploits that may refer to PrintNightmare or CVE-2021-1675.

On July 1, Microsoft released [a bulletin for] CVE-2021-34527 [and its patch Tuesday, July 6th] This bulletin states that CVE-2021-34527 is similar but distinct from the vulnerability that is assigned CVE-2021-1675, which addresses a different vulnerability in **RpcAddPrinterDriverEx()**. The attack vector is different as well. CVE-2021-1675 was addressed by the June 2021 security update.

So this is where we were last Tuesday when, near the middle of last week's podcast, the patch for this second distinct vulnerability in Windows printing became available.

But the next day, last Wednesday, the tech press was bubbling over with the news that, incredible as it might seem, even after applying this patch Windows could still be exploited in some scenarios. For example, "The Hacker News" titled their coverage of this surprising result: "Microsoft's Emergency Patch Fails to Fully Fix PrintNightmare RCE Vulnerability." In ArsTechnica, Dan Goodin's coverage was titled: "Microsoft's emergency patch fails to fix critical "PrintNightmare" vulnerability" and Dan added: "Game-over code-execution attacks are still possible even after fix is installed." And Lawrence Abrams wrote in BleepingComputer: "Microsoft's incomplete PrintNightmare patch fails to fix vulnerability."

So what happened?

Microsoft's position is that the main concern is that non-administrator users had the ability to load their own printer drivers. This is what they have changed. Microsoft wrote: "After applying the updates, users who are not administrators can only install signed print drivers to a print server. Administrator credentials will be required to install unsigned printer drivers on a printer server going forward."

The CERT/CC's vulnerability analyst, Will Dormann, cautioned that the patch "only appears to address the Remote Code Execution (RCE via SMB and RPC) variants of the PrintNightmare, and not the Local Privilege Escalation (LPE) variant," thereby allowing attackers to abuse the latter to gain SYSTEM privileges on vulnerable systems. Further testing, which the security community has jumped on, has revealed that exploits targeting the flaw **could** bypass the remediations entirely to gain **both** local privilege escalation **and** remote code execution.

To which Microsoft is now saying: Yeah, uh huh, that's right. That's on purpose. That's the way it's supposed to work." Microsoft is clearly stating that this behavior is deliberate. They go so far as to write — in their bulletin — the following is taken directly from their bulletin:

Having **NoWarningNoElevationOnInstall** set to **1** makes your system vulnerable by design.

And you know... it's been staring us right in the face this entire time. We've needed a new slogan to go with the new Windows. And now we have it: "Windows 11 — Vulnerable by Design!"

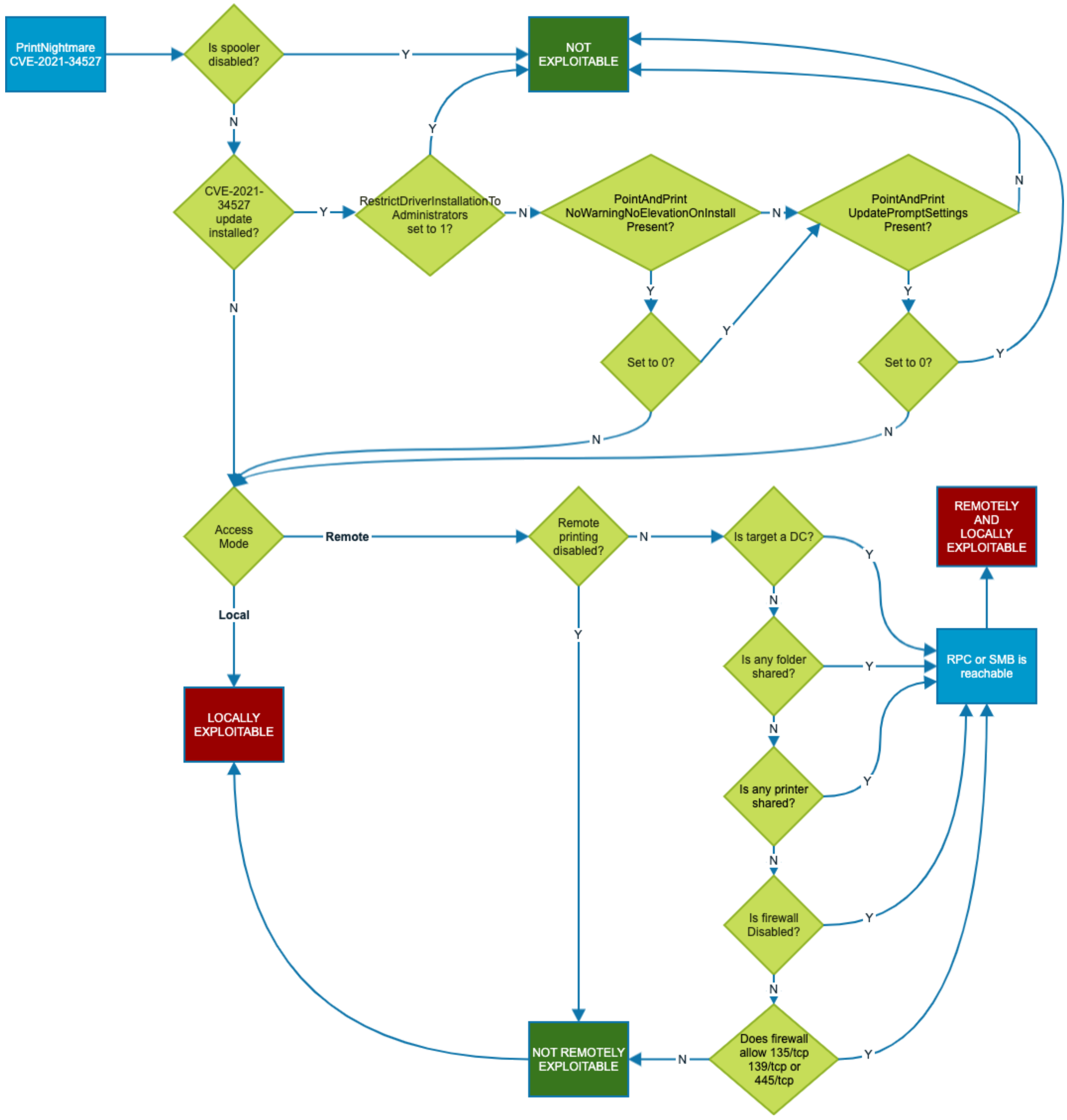
Microsoft's (updated) bulletin explains it precisely:

In addition to installing the updates, in order to secure your system, you must confirm that the following registry settings are set to **0** (zero) or are **not defined** (Note: These registry keys do not exist by default, and therefore are already at the secure setting.), also that your Group Policy setting are correct (see FAQ):

```
HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows
NT\Printers\PointAndPrint
  NoWarningNoElevationOnInstall = 0 (DWORD) or not defined (default setting)
  UpdatePromptSettings = 0 (DWORD) or not defined (default setting)
```

Having NoWarningNoElevationOnInstall set to 1 makes your system vulnerable by design.

https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-34527



PrintNightmare CVE-2021-34527
Is spooler disabled?
NOT EXPLOITABLE
CVE-2021-34527 update installed?
RestrictDriverInstallationToAdministrators set to 1?
PointAndPrint NoWarningNoElevationOnInstall Present?
PointAndPrint UpdatePromptSettings Present?
Set to 0?
Set to 0?
Access Mode
Remote
Local
Remote printing disabled?
Is target a DC?
REMOTELY AND LOCALLY EXPLOITABLE
RPC or SMB is reachable
Is any folder shared?
Is any printer shared?
Is firewall Disabled?
Does firewall allow 135/tcp 139/tcp or 445/tcp
LOCALLY EXPLOITABLE
NOT REMOTELY EXPLOITABLE
Y
N

It appears that Windows 7, 8, 8.1, Server 2008 and 2012 will always be “vulnerable by design” even after being patched. But Windows Server 2016, 2019 and Windows 10 & 11 require Point And Print to be configured. After applying the patches, these later versions of Windows will no longer be vulnerable as long as those two registry keys either do not exist or are set to zero.

In other words, apply the out-of-band patch and be sure that the two keys noted above under
`HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows NT\Printers\PointAndPrint`
do not exist. If you don’t know you need them, remove them for safety and security.

It must be that, at least in the short term, Microsoft was unable to robustly fix this trouble with a simple patch without also disabling some functionality that some of their users rely upon. So, if you need this Point And Print stuff enabled, the trade-off — we’re told by Microsoft themselves — by their deliberate design and clear admission, will be a persistent local privilege escalation vulnerability in all of those Windows systems.

Which brings us back around to our favorite rhetorical question: What could possibly go wrong?

Kaseya — Not nearly as bad as it could have been!
Following the shock that as many as 1500 end-user clients of 60 MSPs were simultaneously attacked by REvil/Sodinookibi ransomware, we have become quite used to the individual nightmare stories that follow. But something quite significant was absent from the Kaseya VSA server-based attacks: Although the REvil affiliate behind these intrusions used vulnerabilities in Kaseya’s VSA servers to transfer and run the REvil ransomware in their client systems, that’s as far as they went. And it has turned out that what they **didn’t** do was significant:

When ransomware gangs conduct the typical attack, they breach a network to actually get into a victim’s network. They set up shop, take a good long look around, establish some mechanism for persistence and survey the territory. Then, as a means of increasing their extortion leverage, they’ll exfiltrate bracing amounts of internal private and probably confidential data. One they’ve done that, they will carefully wipe and completely eradicate any and all backups that have been identified during their stealthy penetration. And, finally, as their last act, at a time when it’s least likely to be noticed, the attackers will trigger the spontaneous encryption of every available computer within the victim’s network.

But that’s not what happened during the Kaseya attacks. As I noted above, the REvil affiliate simply used their Kaseya VSA access to download and run the encrypting ransomware. NO network penetration was performed. As I mentioned last week, indications were that no internal data was exfiltrated. Now we know that for sure. And no backups were located and wiped. Although it’s not nothing to have encrypted an organization’s computers, it turns out that it actually makes a huge difference when the ransom price of a per-client decryptor is $5 million and that client discovers that all of their backups are still in place and are usable.

The result of this half-baked ransomware blitz, affecting as many individual organizations as it has, is that ransoms are very often not being paid. Instead, encrypted systems are being restored from backups. Without the data exfiltration and no way to restore except by paying ransom and obtaining a decryption key, the REvil affiliate lost a tremendous amount of leverage over their victim organizations.

Multiple victim organizations and an MSP have told BleepingComputer that none of their backups were affected and that they chose to restore from their backups rather than paying ransom. Bill Siegel, the CEO of Coveware, a leading ransomware negotiation firm, told BleepingComputer that this is what they are also seeing and that not a single one of their clients has had to pay a ransom.

Bill Siegel said: “Impacted MSPs are going to be stretched for a while as they restore their clients, but so far none of the clients we have triaged have needed to pay a ransom. I'm sure there are some victims out there that will need to, but this could have been a lot worse.”

So it’s great news that a week or two out, we’re not talking about 1500 individual and extremely expensive disasters as a result of the Kaseya VSA server mess. Questions have been raised about how the attackers knew of the vulnerability. Did the news that it was soon to be patched somehow leak? Did they know that they only had a brief window of exploitability? If they were the typical REvil affiliate they surely had to know that not entering individual networks to survey, exfiltrate and wipe backups would dramatically reduce their leverage. But perhaps they hoped to make up for that in numbers? Perhaps they assumed that they’d score enough ransoms to make up for those who would choose to restore from backups? Perhaps they hoped that many smaller organizations were still not backing up? We’ll probably never know. But let’s hope that this remains the “biggest ever” ransomware attack for some time.

https://ransomwhe.re/

If you’re interested in Ransomware, you’re definitely going to want to checkout Jack Cable’s new Ransomwhere site... spelled ransomwhe(dot)re. When I was pulling the show notes for this podcast together, Jack’s site had tracked $60,274,058.06 in all-time ransom payments.

From the site we learn that Netwalker holds the all time record, followed in decreasing order by REvil/Sodinokibi, Ryuk, DarkSide, RagnarLocker, Egregor, Conti, Bitpaymer/DoppelPaymer, SynAck and Qlocker.

And he shows a cool ransom payment chart showing the ransomware, the Bitcoin address, the date, BTC amount, and the transaction hash. He also has a section showing the latest attack reports and a form for sending him new reports.

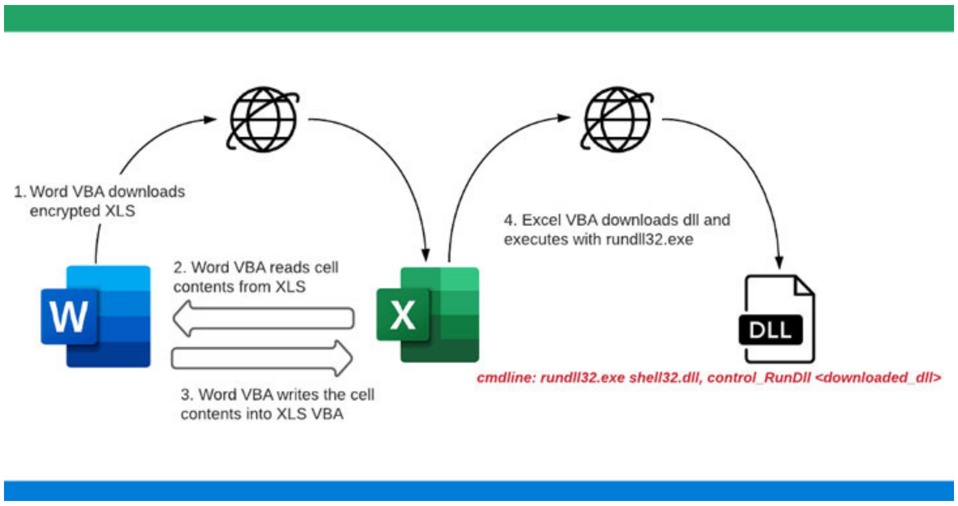
The top three items on his site’s FAQ are:

Why track ransomware payments?
Transparency is crucially needed in assessing the spread of ransomware and the efficacy of mitigations. Fortunately, due to the transparent nature of Bitcoin, it's easy to track payments with knowledge of receipt addresses. By crowdsourcing ransomware payment addresses, we hope to provide an open resource for the security community and the public.

How complete is the data?
As Ransomwhere is new, we are still working on building out our dataset. Reports have placed total ransomware revenue in 2020 at up to $350 million.

Can't someone fake a report?
While it's impossible to verify with complete certainty that a report is accurate, we aim to utilize the wisdom of the crowds to prevent abuse. All reports are required to include a screenshot of the ransomware payment demand, and will be reviewed before being displayed. Addresses with more than one report from different sources will be given priority, and all elements of all reports will be publicly available. We will remove reports if we believe they are untruthful.

Microsoft Office Users: There's a new malware-protection bypass


1. Word VBA downloads encrypted XLS
2. Word VBA reads cell contents from XLS
3. Word VBA writes the cell contents into XLS VBA
4. Excel VBA downloads dll and executes with rundll32.exe
DLL
cmdline: rundll32.exe shell32.dll, control_RunDll <downloaded_dll>

I want to walk everyone through the design and operation of a newly discovered Office exploit so that we can all obtain a deeper appreciation for what we're up against in the ongoing struggle to simply use our own machines safely.

"Zloader" as it's known, is a banking Trojan which steals credentials and other private information from users of targeted financial institutions. This new attack uses Office's Word and Excel, working cleverly in concert, to disable Office's macro warnings and to then download and install this Zloader banking malware without any security tools recognizing and flagging it as such. So this will be a look behind those famous links that are received in phishing eMails.

The initial attack vector is a standard inbox-based phishing message containing a Word document attachment that, itself, contains no malicious code. Thus, it wouldn't typically trigger an email gateway scanner or any client-side antivirus software to block the attack.

Since Office automatically disables macros, the attackers eMail contents attempts to trick the naïve eMail recipient to enable them with a message appearing inside the Word document. The word document explains: "This document was created by a previous version of Microsoft Office Word. To view or edit this document, please click the 'Enable editing' button on the top bar, and then click 'Enable content'."

Since people transacting in Word document attachments will have likely encountered similar legitimate Word version conversion messages in the past, the document's legitimate appearing instructions have some likelihood of being followed, especially when all of the typical phishing accoutrements of the eMail being expected, coming from a known and expected source, etc. are piled on top.

Now the fun begins...

Visual Basic and Applications (VBA) code embedded in the Word document uses DDE (Dynamic Data Exchange) to read a specially crafted .XLS Excel spreadsheet from a remotely located foreign domain into an Excel cell to create and then run an Excel macro. That macro, in turn, populates an additional cell in the same XLS document with another VBA macro, which disables Office's defenses. The Word document is able to indirectly set the policy in the registry: 'Disable Excel Macro Warning,' after which it invokes the malicious macro function from the Excel file. The Excel file downloads the Zloader payload and executes it on-the-fly using rundll32.exe.

So what we have is a back and forth interaction between two products, Word and Excel, which have both grown into incredibly complex application hosting containers. And with a bit of benign-appearing social engineering, an unwitting user, doing nothing more than they have probably done many times before, allows foreign code to be downloaded into their machine and run.

So we find ourselves back again, wondering whose fault this is? There's no doubt that the power of Word and Excel have enabled marvelous capabilities and the construction of valuable applications of their own. And it's also true that users of these programs which have been around for many years, and which have gone through many changes and versions will often receive actual requests for actions and conversions based upon version mismatches.

So it seems to me that the problem is that eMail is **still**, in 2021, not being regarded by Microsoft as a sufficiently dangerous vector. And any fair reading of the evidence would lead to the conclusion that this appears to be a huge blind spot for Microsoft. Malicious macros hosted by Outlook eMail predate this podcast's first episode nearly 16 years ago, and at this rate it doesn't appear that this podcast is going to outlive them. A huge amount of effort has been made to successfully sandbox our web browsers because they represent an obvious source of externally-provided potentially malicious code. Why is eMail any different? By definition it's externally-provided, and if it's allowed to have executable Office document content attached, which includes Visual Basic for Applications code, the profile of its attack surface is no different from a web browser's.

This is one of those maddening things where we keep hearing about the same problem over and over, year after year, yet nothing gets done about it. So what do we inevitably wind up doing? We blame the user for being duped when they behave in a perfectly natural way. It's not right.

Ransomware negotiators are now in high demand

What do you do if all of your victims speak English but you only speak Russian and yet you need to interact with them to negotiate a ransom settlement?

We've talked about the evolution of the ransomware ecosystem. How there are now penetration providers who specialize in breaking in, but that's as far as they go. Those break-in penetrations

are purchased by Ransomware as a Service (RaaS) affiliates who then move into the victim networks to inventory and exfiltrate, wipe backups and eventually encrypt the network.
And on the victim's side we have Ransomware attack insurers and the emergence of third-party ransom escrow providers. To this milieu we now add "negotiators" who are interposed between the attackers and their victims and their victims' agents.

A recently published study of RaaS trends revealed that the traditional go-it-alone ransomware operations have virtually disappeared in favor of multi-tiered organizations. And most recently this has led to a high demand for individuals to take over the negotiation part of an attack chain.

As KELA, the publisher of the study explained, a typical ransomware attack comprises four stages: malware/code acquisition, spread and the infection of targets, the extraction of data and/or maintaining persistence on impacted systems, and monetization. There are actors in each 'area,' and recently, demand has increased for extraction and monetization specialists in the ransomware supply chain.

The emergence of so-called negotiators in the monetization arena, in particular, is now a trend in the RaaS space. KELA researchers say that specifically, more threat actors are appearing that manage the negotiation aspect, as well as piling on the pressure -- such as though calls, DDoS attacks, and making threats including the leak of information stolen during a ransomware attack unless a victim pays.

This newly clarified role has emerged due to the need for individuals able to manage conversational English in order to effectively conduct the many aspects of post-attack negotiations. A spokesman for KELA said: "This part of the attack also seems to be an outsourced activity -- at least for some affiliates and/or developers. The ransomware ecosystem, therefore, more and more resembles a corporation with diversified roles inside the company and multiple outsourcing activities."

Initial access brokers, as they're now called, are also seeing increasing demand. After observing dark web and forum activity for over a year, the researchers say that privileged access to compromised networks has surged in price. Some listings are now 25% - 115% more than previously recorded, especially when domain admin-level access has been achieved. And these "intrusion specialists" are being paid between 10% and 30% of a ransom payment.

Miscellany

You WILL use the new Start menu... and you WILL like it!
Last Thursday, Microsoft released Windows 11 build 22000.65 to Insiders on the 'Dev' channel. And, believe it or not, support for the manual Registry value that could be added to cause earlier pre-release builds of Windows 11 to revert to a Win10 style left-aligned Start menu had been removed. At this time it's no longer possible to ask Windows 11 to continue with the Win10 style Start menu that many have grown used to and wished to continue using. Hopefully, Microsoft will choose to surface an official option for the "Classic Win10 Start menu" in the Win11 Settings.

SpinRite

I don't have anything earthshaking to report since last week on the SpinRite front. I'm at work rewriting SpinRite's core, converting it from its historical track-based design to a linear sector stream architecture. Once that's finished, I'll first put everything through its paces, then I'll turn the GRC newsgroup gang loose with the code that will become the pre-release of SpinRite v6.1. People have been writing, asking for the SpinRite v6.1 beta. So I wanted to clarify that nothing yet exists for anyone to use. As soon as it does exist, it will be made available, in beta form, for existing v6.0 users to download. So stay tuned. :)

REvil's Clever Crypto

I saw somewhere that Kaspersky Labs wrote: "REvil uses the Salsa20 symmetric stream algorithm for encrypting the content of files and the keys for it with an elliptic curve asymmetric algorithm. Decryption of files affected by this malware is impossible without the cybercriminals' keys due to the secure cryptographic scheme and implementation used in the malware."

And then, as I noted last week, we also learned that it was possible to obtain decryption keys for files of an given file extension, or for an entire single enterprise, or for all enterprises that were victims of the attack. So I wanted to explore and share the design of REvil's Clever Crypto...

But first off, I've noted that most of the tech press is unaware or inaccurate about their use of the term REvil vs Sodinokibi. I try to use them correctly, but I'm often needing to quote their misuse. So, just for the record, REvil is the gang and Sodinokibi is their encrypting malware. Though I titled this episode REvil's Clever Crypto since it rolls off the tongue more easily, it's the clever crypto employed by the REvil gang in the design of their Sodinokibi malware.

Before I describe the awesome cryptographic design employed by Sodinokibi, I want to quote something from Acronis' broader description of their reverse engineering of Sodinokibi. To any non-Windows API coder this going to sound mostly like a bunch of mumbo-jumbo connected by a few recognizable bit of English. But I think it's worthwhile to set the stage for understanding what we're facing with the design of Sodinokibi. Here's what Acronis found:

To encrypt user files, Sodinokibi uses I/O Completion Ports and creates multiple encryption threads to a maximum of twice the number of processor cores present on the machine and associates these threads to the created I/O completion port. These threads use the ***GetQueuedCompletionStatus*** *Win API function to wait for a completion packet to be queued to the I/O completion port before they proceed to the file encryption.*

Once the threads are created and waiting for I/O packets to arrive, Sodinokibi starts enumerating user files on all the local drives and network shares [for anyone who may have been wondering whether Sodinokibi followed network shares] except CDROM and RAMDISK drives, and begins associating files which are not in the exempted "folder, file or file extension" list to this I/O completion port by calling their user function ***AddFileToIoCompletionPort*** *and calls the* ***PostQueuedCompletionStatus*** *Win API to post an I/O packet on the I/O completion*

port which will trigger the thread waiting on this I/O completion port to resume and proceed to encrypt files.
*The user function **AddFileToIoCompletionPort** also generates a unique Salsa20 key for each file that is to be encrypted and passes this Salsa20 key to the encrypting thread as part of the structure containing other metadata that has to be written to file as well after encryption using lpOverlapped parameter of **PostQueuedCompletionStatus** Win API. During enumeration, it also creates a ransom note file in all folders that are not in the exempted folder list. Once there are no more files to enumerate the main thread waits in a loop until the total number of files encrypted and renamed equals to the total number of files added to the I/O completion port for encryption.*

Finally, it sets a flag which indicates there are no more files to enumerate and posts multiple I/O completion packets, by doing this it makes sure that extra threads waiting for files should resume and break the execution flow to finish immediately.

I've been programming Windows for several decades and what I just read not only makes perfect sense to me, but it is precisely the optimal maximum performance multi-threaded design for working with any large collection of objects under the Windows API. In fact, this is precisely the design I use to manage all of GRC's web server side services where a lot of stuff needs to be going on at the same time.

What I read above amounts to using low-level native Windows APIs to create a queue of file encryption jobs which will be serviced by a pool of processor core threads. The reason this is so efficient is that jobs are removed from the queue explicitly avoiding expensive thread context switches. Typical queues would wind up assigning jobs in a round-robin fashion where the longest waiting thread would get assigned the next task. But Windows "CompletionPort" design deliberately allows the same thread that's posting its completion status to obtain the next job on the queue and to begin working on it immediately, without losing its scheduling time slice.

To put this into context, it's a small matter. For Sodinokibi, where the individual jobs will be comparatively large, since file chunks are being concatenated then encrypted and renamed, the overhead of thread context switching would be insignificant. But what this showed me was that whoever this piece of code really knew their way around the Windows API and really knew what they were doing.

Sodinokibi's Crypto Design
When interviewed, the REvil gang has expressed extreme pride in their creation of Sodinokibi. As one researcher commented, this pride was warranted because Sodinokibi is one of the few ransomware families that actually thought about its cryptographic scheme and how to account for various situations.

There are four applications of public / private keypairs used in Sodinokibi, as follows:

There's an **operator** keypair whose public half is hardcoded and built into every Sodinokibi code sample that has been obtained. This value is set by the REvil operators and cannot be set or changed the REvil affiliate.

There's a **campaign** keypair whose public half is stored as the "**PK**" value in Sodinokibi's per

victim or per campaign configuration, depending upon the application. This value can be modified by the Affiliate.
Then there's a **system-specific** keypair that's generated as an individual system is being encrypted. The private half of the keypair is encrypted using **both** the **operator** and the **campaign** public keys and is stored in each encrypted file and ransom note.

And finally, there's a unique **per-file** keypair individually generated for each encrypted file. The public half is stored in plaintext within the Sodinokibi file header that's appended to the front of the encrypted file's contents.

Bulk file encryption is performed using Dan Bernstein's excellent and quite fast Salsa20 symmetric cipher. The symmetric key which will be used for both encryption and decryption is obtained by combining the private half of the per-file keypair with the public system key using Elliptic Curve Diffie Helman under Curve25519.

What's so clever about this is that when needing to decrypt the file, Sodinokibi obtains the public half of the per-file key from the file's appended header which it combines with the private half of the **system key**, again using ECDH to recreate the same shared symmetric key. And, yes, anyone who is deeply familiar with SQRL will recognize that I designed exactly the same approach into SQRL's identity lock and unlock capability. These guys know their crypto.

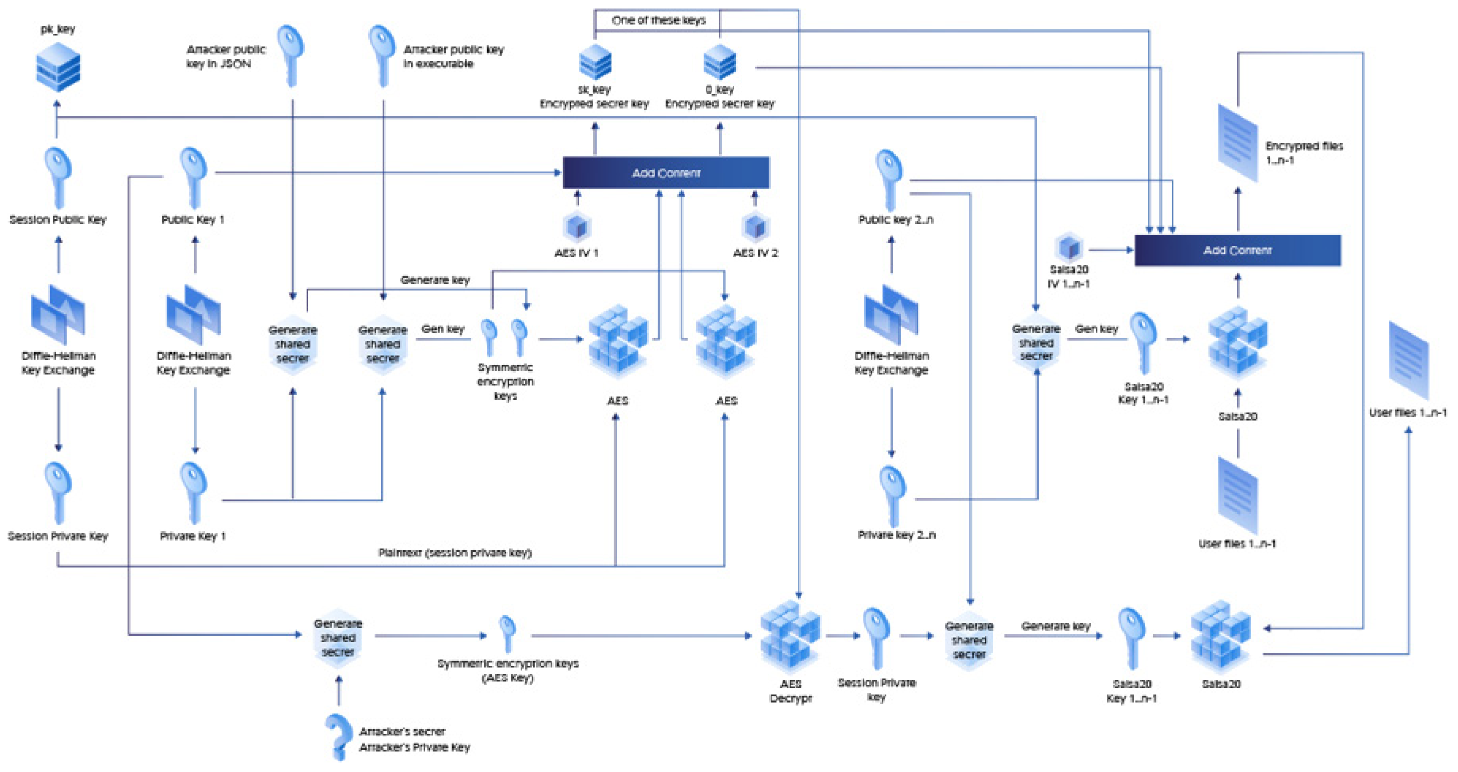
The key takeaway here (ignore the pun) is that ONLY the private half of the **system key** is required to decrypt every file for that system. Each file to be decrypted provides its public key to the ECDH function to obtain the symmetric Salsa20 key for its own decryption. The other property this has is that every single file that's encrypted is encrypted under a different and unique symmetric key.

So, how do we recover that all-important private half of the **system key**? Recall that the private half of the system key was encrypted separately using the public halves of both the **operator** and the **campaign** key. This means that the private half of either the **operator** or the **campaign** key can be used to decrypt the **system key**.

Since the public half of the **operator** key is embedded in the Sodinokibi code which is individualized for each affiliate, and cannot be changed by the affiliate, this provides the REvil operators with a per-affiliate backdoor enabling them to unilaterally decrypt anything that any of their software is used to encrypt. This provides the REvil operators with protection against any rogue affiliate who might violate their rules.

Both of these approaches would decrypt an entire campaign. But we saw with the Kaseya attack that the attacking affiliate was offering to decrypt individual systems for the bargain price of only $45k. The way this is done is by distributing a decryptor that only contains a specific system's previously-decrypted **private key**. Given the public half of any **system key**, the operator's **private key** can be used to obtain the private half of that single system's key. And it can then be combined with each file's **per-file** key to obtain its Salsa20 symmetric decryption key.

Without an explanation like what I've just provided, a flowchart of Sodinokibi's cryptographic operations is quite daunting. And, in fact, even WITH a careful description, Sodinokibi's design can be a bit overwhelming, as shown by the diagram below:



The bottom line is that whoever designed this beast several years ago really thought the problem all the way through. Unlike many of the less professionally designed half baked ransomware we've seen, these guys did not make any mistakes that would allow for short-circuiting the need to pay for decryption. And in the process they have created a powerful, flexible and efficient file encryption system.

The Acronis reverse-engineering coverage included documentation of the folders, file and file extensions that Sodinokibi skips over and leaves as-is. Since it makes for some interesting reading, if you're into that sort of detail, I've ended this podcast's show notes with those lists:

Exempted Folders: $windows.~bt, intel, program files (x86), program files, msocache, $recycle.bin, $windows.~ws, tor browser, boot, system volume information, perflogs, google, application data, windows, programdata, windows.old, appdata, mozilla

Exempted Files: bootfont.bin, boot.ini, ntuser.dat, desktop.ini, iconcache.db, ntldr, ntuser.dat.log, thumbs.db, bootsect.bak, ntuser.ini, autorun.inf

Exempted File extensions: themepack, ldf, scr, icl, 386, cmd, ani, adv, theme, msi, rtp, diagcfg, msstyles, bin, hlp, shs, drv, wpx, deskthemepack, bat, rom, msc, lnk, cab, spl, ps1, msu, ics, key, msp, com, sys, diagpkg, nls, diagcab, ico, lock, ocx, mpa, cur, cpl, mod, hta, exe, icns, prf, dll, nomedia, idx