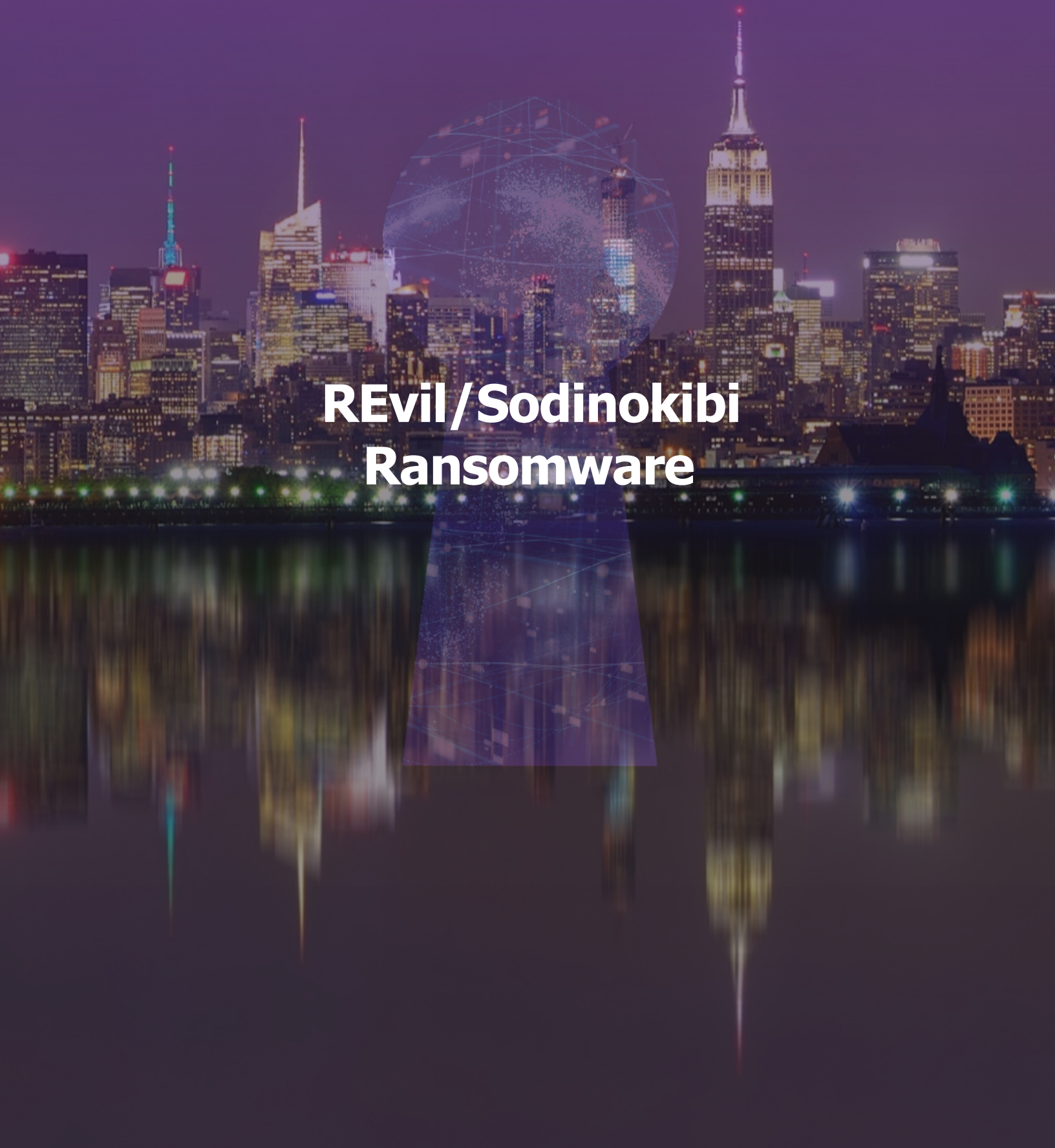
LIFARS
your digital world, **secured**



REvil/Sodinokibi Ransomware

TABLE OF CONTENTS

OVERVIEW

During a recent client engagement, the LIFARS DFIR team encountered the REvil/Sodinokibi Ransomware group. The typical attack vector chosen by this group is either the exploitation of vulnerable network devices or brute-force attacks on Remote Desktop Protocol servers. In this case study we will present selected artifacts that will provide important insight into the threat actors' behavior.

The patient zero machine was a development server with a public IP address. In this article we will focus on artifacts identified on this host, even though different artifacts, and the ransomware sample itself were identified on other affected systems. Patient zero had ports 139(NetBIOS) and 3389(Remote Desktop Protocol) open, which was most probably the initial vector of compromise. As previously stated,

REvil/Sodinokibi threat actor looks to brute force the RDP service. We were able to find evidence that is consistent with this claim (high volume of failed logon attempts), but other artifacts lead the team to believe the attacker first used anonymous logons via NetBIOS to gain important account information, for example usernames. Having port 139 open within your Local Area Network (LAN) is necessary: It enables applications and network hosts to communicate with network hardware and transmit data across the network. Contrary to needing port 139 open within your LAN, having this port on your Wide Area Network (WAN) or over the internet is an enormous security risk. The presented case is an excellent example how can attackers leverage such configuration. These two open ports coupled together made the initial attack vector quite simple, even for script kiddies themselves.

USER ASSIST ARTIFACTS

Execution of the following executables was attributed to the initially compromised user account:

- CVE-2017-0213_x64.exe
- Sharphound.exe
- Kiwi Parser.exe

The UserAssist registry key allows examiners to see what programs were recently executed on the system by a specific user, using GUI. For this part of the examination process we chose to use the tool Registry Explorer, which helped the team identify a trove of executables that were run by the threat actors. Although most of these files were deleted, they were able to provide useful insights into some of the steps taken by the attacker. These executables were run by the user which we initially suspected was the victim of the brute force attack. We suspected this user because of the multiple failed logon attempts followed by a type 10 successful logon, which equates to a Remote Desktop connection.

For example, on the desktop of the initially exploited user we found CVE-20170213_x64.exe, a Windows COM privilege escalation vulnerability. If this file contained exploit of this vulnerability, attackers could run it to elevate privileges within a network. We could not verify this hypothesis, as the file was no longer present on the system and it could not be recovered from the disk. Neither have we found additional artifacts that would throw any light on what happened after executing this file.

UserAssist key also provided the team with evidence that SharpHound.exe was executed. SharpHound is an ingestor - official data collector - for the popular penetration testing tool BloodHound. BloodHound is a powerful GUI application that will map an entire active directory environment, while also identifying attack paths and detecting the shortest path to privileged accounts or domain controllers. Figure 1 shows the

BloodHound results used in a simulated environment; Figure 2 shows another feature of the tool that provides information and references to abusing a certain permission.

SharpHound execution is a sign of the threat actors' attempts to map the network and gain intelligence allowing for privilege escalation.

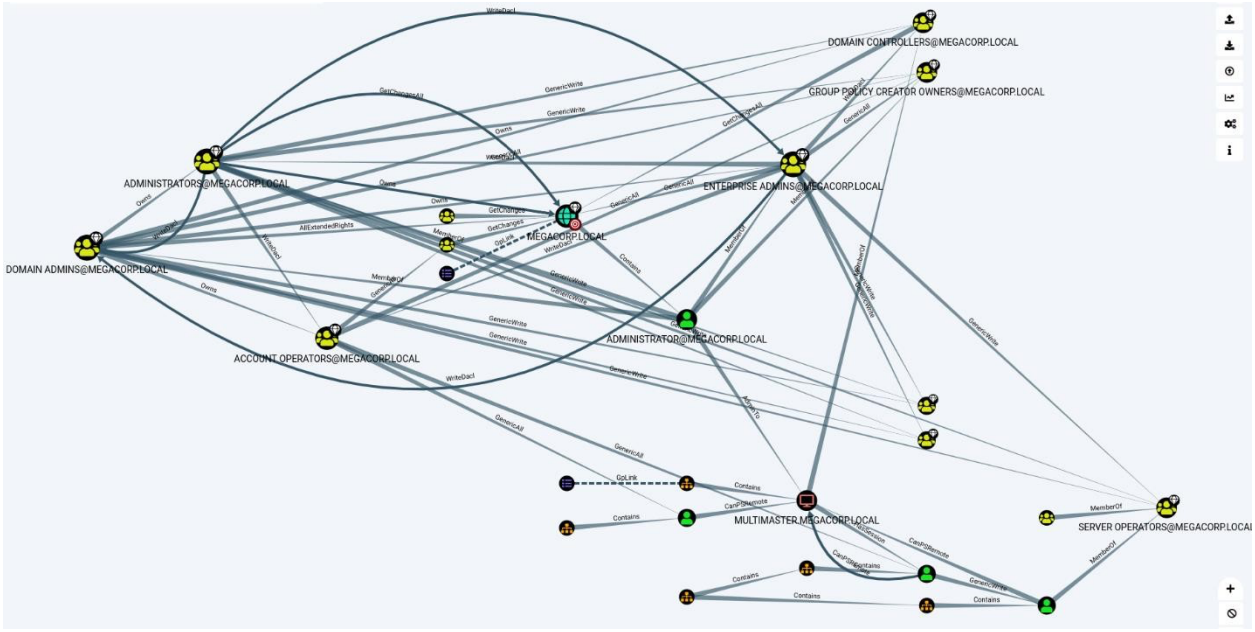


*Figure 1: BloodHound GUI mapping AD environment - *test network**

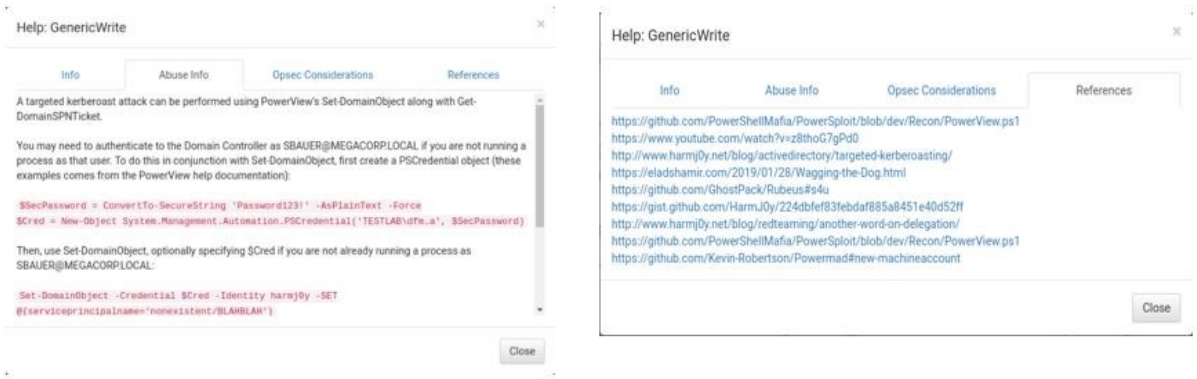


Figure 2: Abuse information and references

Along with the two previously mentioned executables, the DFIR team was able to find evidence of execution of a file called Kiwi Parser.exe, which is associated with a known Windows OS nemesis, Mimikatz. At this stage of the attack the threat actors most likely had administrative privileges, probably gained from successful exploitation of the CVE-2017-0213, as the file of corresponding name had been executed on the system. With administrative privileges, Mimikatz makes it easy for an attacker to gather credentials and to further use them for lateral movement along the network in a variety of different fashions (Pass-the-Hash, Pass-the-Ticket, Kerberos Golden Ticket, etc.). With the knowledge from Bloodhound, the threat actors were able to see exactly where the Mimikatz needed to be used. These three executables used together paint a clear picture of the process taken – or at least attempted - by the threat actors during the post-

exploitation phase. The threat actors were able to elevate their privileges, probably using a known vulnerability, map the active directory environment to find the path of least resistance to the domain controller, and dump user credentials.

Due to the manner UserAssist registry key works, we were able to find when listed executables run on Patient zero, and which user account was responsible for their execution. LIFARS DFIR team made use of these facts while investigating other systems involved in the attack: the knowledge of the compromised username allowed for a more targeted investigative approach.

POWERSHELL ACTIVITY

As a part of the standard investigative process, our team investigated PowerShell event logs. While reviewing PowerShell logs, we identified a Base64 encoded PowerShell command. Such commands always catch investigators attention! Upon closer review and after decoding the Base64 payload, true action of the command revealed:

Decoded command: Get-WmiObject Win32_ShadowCopy | ForEach-Object

{$_.Delete():}

The command deleted Volume Shadow copies. This is a routine technique for ransomware actors to deploy (see Figure 3). Shadow copies are a Microsoft Windows feature that creates backup copies or snapshots of computer files or volumes. This technique would make sense to a ransomware attacker so the victim would not be able to revert their system to a prior state before the attack happened.

Figure 3 – PowerShell logs indicating shadow copies deletion

But there was more to find in PowerShell logs. The team found more suspicious PowerShell activity that pointed to a Cobalt Strike stager. Cobalt Strike is a publicly available framework that assists in loading shellcode onto victims' computers. Cobalt does have legitimate uses for penetration tests, but most recently there has been an increase in its use by threat actors. In Figure 4.1 we see the beginning of the code with the emphasis on the conversion from Base64 (again). By reviewing Figure 4.2 it is evident that the Base64 string is being compressed by using gzip. Attackers often use such operations in their stagers to reduce the size of large files. Encoding and compression can help prevent antimalware detection.

LIFARS your digital world, secured
244 Fifth Avenue, Suite 2035, New York, NY 10001
LIFARS.com (212) 222-7061 info@lifars.com

Figure 4.1 – Base64 conversion

Figure 4.2 – gzip compression

Once the contents in the Base64 string are decoded and decompressed, another remote command can be observed (see Figure 5.1 and 5.2). The resulting script contains another Base64 string and loads the shellcode into memory. After observing and breaking down the lines of code, it looks most probably as a type of sophisticated fileless malware.

Figure 5.1 – Fileless malware (beginning)

```
for ($x = 0; $x -lt $var_code.Count; $x++) {
	$var_code[$x] = $var_code[$x] -bxor 35
}

$var_va = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll VirtualAlloc), (func_get_delegate_type @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr])))
$var_buffer = $var_va.Invoke([IntPtr]::Zero, $var_code.Length, 0x3000, 0x40)
[System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length)

$var_runme = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($var_buffer, (func_get_delegate_type @([IntPtr]) ([Void])))
$var_runme.Invoke([IntPtr]::Zero)
"@

If ([IntPtr]::size -eq 8) {
	start-job { param($a) IEX $a } -RunAs32 -Argument $DoIt | wait-job | Receive-Job
}
else {
	IEX $DoIt
}
```

Figure 5.2 – Fileless malware (ending)

A WORD ON FILELESS MALWARE

With the development of sophisticated security solutions, fileless malware grown in popularity over the years. The advantage of fileless malware is its ability to evade security techniques while also frustrating forensic efforts because it does not leave a footprint. The reason behind its ability to evade security tactics is that, like the name implies, there are no files or folders that are left on the system, in fact, the infections go straight to memory and do not touch the hard drive. This type of malware often uses native tools such as Powershell and Windows Management Instrument (WMI) to run the scripts locally. Since this malware works in memory, when the system is rebooted the evidence is lost, adding another layer of difficulty to an already challenging investigation.

CLOSER LOOK AT THE PAYLOAD

In the beginning of the code in Figure 5.1, there are two distinct functions visible – func_get_proc_address and func_get_delegate_type. To start, func_get_proc_address returns a memory address for a specific procedure, in this case

Microsoft.Win32.UnsafeNativeMethods, which is located in System.dll as declared in the script.

The next function, func_get_delegate_type, looks to be creating a new delegate named 'ReflectedDelegate'. This delegate can be used to manage the necessary types of inputs and outputs within the PowerShell code, so the script itself knows what to expect when retrieving different types of functions inside different .dll libraries. For example, later in the script (Figure 5.2) it can be observed that the func_get_proc_address makes a call to kernel32.dll.

Moving on, we can see that the decryption loop uses a XOR decryption type with a key of 35. By using the VirtualAlloc function that is seen to be stored in the $var_va variable, this script is able to allocate the necessary amount of space needed for the payload and then subsequently store the payload into memory. Finally, after the function is stored in memory, the $var_runme variable is used to execute the payload.

MALWARE EXECUTABLE

While running malware scans on the mounted image, one of AV solutions detected a malicious executable:

File: 1

Generic.Malware/Suspicious,

R:\USERS******\APPDATA\ROAMING\X86_MICROSOFT-

WINDOWSMP43DECD_31BF3856AD364E35_6.1.7600.16385_NONE_B40981B052

84B367\DPNADDR.EXE, No Action By User, 0, 392686, 1.0.25486, , shuriken,

This file was determined to be malware completely written in AutoIt script. The analyzed malware employed a well-known method of achieving persistence: Scheduled Tasks. Using this Windows mechanism, the malware sample was executed in 1-minute intervals, thus the threat actor ensured that even after rebooting the machine, their access to the machine will be preserved. Malware was based on Qulab Stealer and Clipper, a tool for stealing credentials, browser history, cookies, but is also able to replace the contents of clipboard. An experienced LIFARS malware analyst reverse engineered this rare malware sample and wrote an in-depth study on it. To read more on this sample please visit the LIFARS website by clicking the link provided: https://lifars.com/knowledge-center/clipper-autoit-v2-quilclipper-autoit-malware/

CONCLUSION

The REvil/Sodinokibi ransomware group use tactics that take advantage of network errors. It is extremely important to make sure publicly accessible servers do not have open ports that threat actors can take advantage of. In this case, ports 139 and 3389 acted as the initial entrance for treat actors. After crossing this wide-open door, threat actors eventually gained administrative rights, laterally moved to other systems and executed ransomware on multiple servers, leaving behind encrypted files and ransomware notes, pushing the victim into paying for data decryption. The threat actors have used multiple popular techniques and tactics along the way: PowerShell stager with encoded payloads, payload execution in memory, usage of SharpHound and Mimikatz, achieving persistence through Scheduled Tasks. On the other hand, malware written in AutoIt script is rather rare.

Ransomware can completely cripple companies of all sizes by not only taking money, but also time and resources that would be normally used for business aspects. Reputation loss and possible legal follow-up in the case that sensitive data are encrypted are other threats imposed by ransomware groups. Securing your network on every possible level is only one of preventive measures we advise our clients to implement. Detection of adversary actions taken during the attack is crucial, but just as important is the capability to contain the incident and to take appropriate remediation steps to recover. Check out LIFARS portfolio to find a suitable solution, which can improve your company’s cyber-security posture today!

REFERENCES

https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapivirtualalloc

https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapigetprocaddress

https://lifars.com/knowledge-center/clipper-autoit-v2-quilclipper-autoit-malware/

244 Fifth Avenue, Suite 2035, New York, NY 10001
LIFARS.com (212) 222-7061 info@lifars.com