www.welivesecurity.com /2022/09/06/worok-big-picture/

Worok: The big picture

⋮ 9/6/2022

Focused mostly on Asia, this new cyberespionage group uses undocumented tools, including steganographically extracting PowerShell payloads from PNG files

Thibaut Passilly
6 Sep 2022 - 11:30AM

Focused mostly on Asia, this new cyberespionage group uses undocumented tools, including steganographically extracting PowerShell payloads from PNG files

ESET researchers recently found targeted attacks that used undocumented tools against various high-profile companies and local governments mostly in Asia. These attacks were conducted by a previously unknown espionage group that we have named Worok and that has been active since at least 2020. Worok's toolset includes a C++ loader CLRLoad, a PowerShell backdoor PowHeartBeat, and a C# loader PNGLoad that uses steganography to extract hidden malicious payloads from PNG files.

Who is Worok?

During the ProxyShell (CVE-2021-34523) vulnerability disclosure in early 2021, we observed activity from various APT groups. One exhibited characteristics common with TA428:

- Activity times
- Targeted verticals
- Usage of ShadowPad

The rest of the toolset is very different: for example, TA428 took part in the Able Desktop compromise in 2020. We consider that the links are not strong enough to consider Worok to be the same group as TA428, but the two groups might share tools and have common interests. We decided to create a cluster and named it Worok. The name was chosen after a mutex in a loader used by the group. Further activity with variants of the same tools was then linked to this group. According to ESET's telemetry, Worok has been active since late 2020 and continues to be active as of this writing.

Back in late 2020, Worok was targeting governments and companies in multiple countries, specifically:

- A telecommunications company in East Asia
- A bank in Central Asia
- A maritime industry company in Southeast Asia
- A government entity in The Middle East
- A private company in southern Africa

There was a significant break in observed operations from 2021-05 to 2022-01, but Worok activity returned in 2022-02, targeting:

- An energy company in Central Asia
- A public sector entity in Southeast Asia

Figure 1 presents a visual heatmap of the targeted regions and verticals.

Figure 1. Map of the targeted regions and verticals

Considering the targets' profiles and the tools we've seen deployed against these victims, we think Worok's main objective is to steal information.

Technical analysis

While the majority of initial accesses are unknown, in some cases through 2021 and 2022 we have seen exploits used against the ProxyShell vulnerabilities. In such cases, typically webshells have been uploaded after exploiting these vulnerabilities, in order to provide persistence in the victim's network. Then the operators used various implants to gain further capabilities.

Once access had been acquired, the operators deployed multiple, publicly available tools for reconnaissance, including Mimikatz, EarthWorm, ReGeorg, and NBTscan, and then deployed their custom implants: a first-stage loader, followed by a second stage .NET loader (PNGLoad). Unfortunately, we have not been able to retrieve any of the final payloads. In 2021, the first-stage loader was a CLR assembly (CLRLoad), while in 2022 it has been replaced, in most cases, by a full-featured PowerShell backdoor (PowHeartBeat) – both execution chains are depicted in Figure 2. These three tools are described in detail in the following subsections.

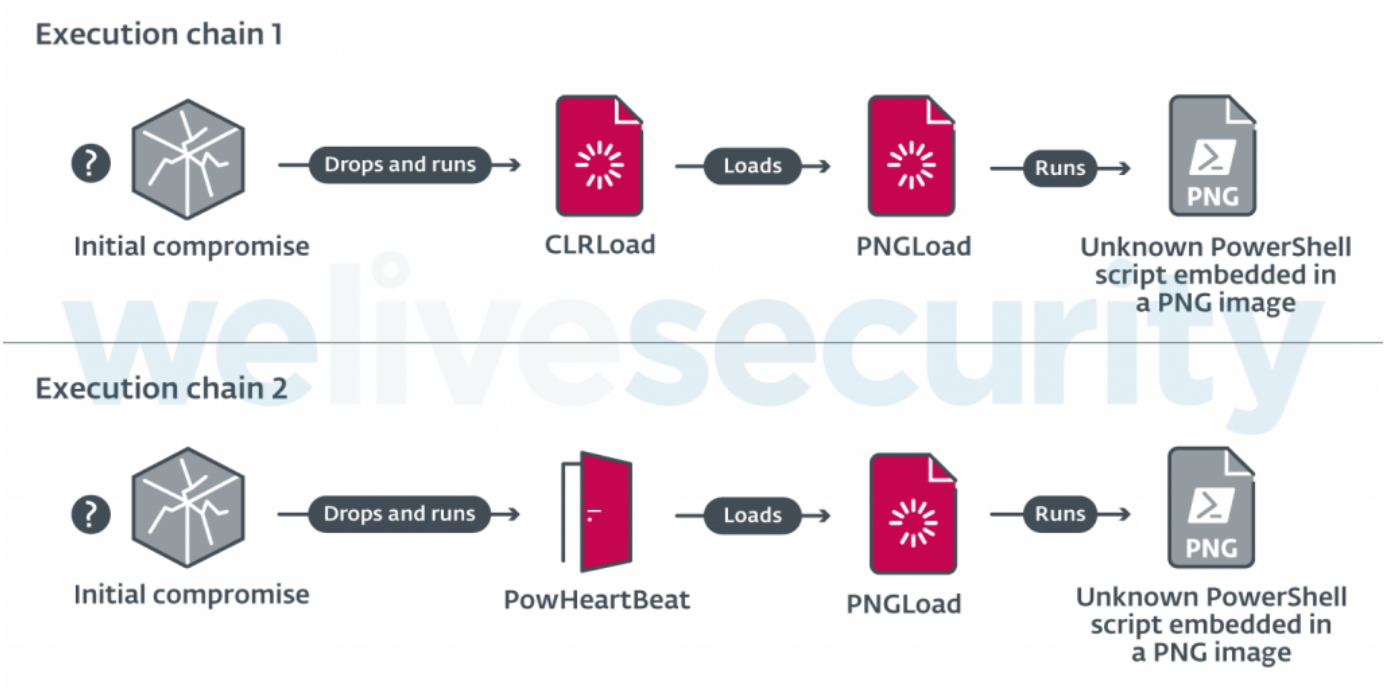

Figure 2. Worok compromise chains

CLRLoad: CLR assembly loader

CLRLoad is a generic Windows PE that we have seen in both 32-and 64-bit versions. It is a loader written in C++ that loads the next stage (PNGLoad), which must be a Common Language Runtime (CLR) assembly DLL file. That code is loaded from a file located on disk in a legitimate directory, presumably to mislead victims or incident responders into thinking it is legitimate software.

Some CLRLoad samples start by decoding the full path of the file whose content they will load as the next stage. These file paths are encoded with a single-byte XOR, with a different key in every sample. Decoded or cleartext, these file paths are absolute, with the following being those we have encountered:

- C:\Program Files\VMware\VMware Tools\VMware VGAuth\xsec_1_5.dll
- C:\Program Files\UltraViewer\msvbvm80.dll
- C:\Program Files\Internet Explorer\Jsprofile.dll
- C:\Program Files\WinRar\RarExtMgt.dll
- C:\Program Files (x86)\Foxit Software\Foxit Reader\lucenelib.dll

Next, a mutex is created and we've seen a different name in every sample. The loader checks for this mutex; if found, it exits, because the loader is already running. In one of the samples, the mutex Wo0r0KGWhYGO was encountered, which gave the group its name of Worok.

CLRLoad then loads a CLR assembly from the possibly decoded file path. As unmanaged code, CLRLoad achieves this via CorBindToRuntimeEx Windows API calls in 32-bit variants, or CLRCreateInstance calls in 64-bit variants.

PowHeartBeat: PowerShell backdoor

PowHeartBeat is a full-featured backdoor written in PowerShell, obfuscated using various techniques such as compression, encoding, and encryption. Based on ESET telemetry, we believe PowHeartBeat replaced CLRLoad in more recent Worok campaigns as the tool used to launch PNGLoad.

The first layer of the backdoor code consists of multiple chunks of base64-encoded PowerShell code. Once the payload is reconstructed, it is executed via IEX. Once decoded, another layer of obfuscated code is executed, which we can see in Figure 3.

```
function install-scapy243([String] $pyDatalog0171, [String] $entrypoints03, [String] $websocketclient0560, [Byte[]] $dpkt192)
){
$torch120cpu = [Convert]::FromBase64String($websocketclient0560);
$encoding = New-Object System.Text.ASCIIEncoding;
$derivedPass = New-Object System.Security.Cryptography.PasswordDeriveBytes($pyDatalog0171, $encoding.GetBytes($entrypoints03), "
[Byte[]] $scikitlearn0213 = $derivedPass.GetBytes(16);
$Pillow610 = New-Object System.Security.Cryptography.TripleDESCryptoServiceProvider;
$Pillow610.Mode = [System.Security.Cryptography.CipherMode]::CBC;
[Byte[]] $pyparsing242 = New-Object Byte[]($torch120cpu.Length);
$g = $Pillow610.CreateDecryptor($scikitlearn0213, $dpkt192);
$i = New-Object System.IO.MemoryStream($torch120cpu, $True);
$j = New-Object System.Security.Cryptography.CryptoStream($i, $g, [System.Security.Cryptography.CryptoStreamMode]::Read);
$r = $j.Read($pyparsing242, 0, $pyparsing242.Length);
$i.Close();
$j.Close();
$Pillow610.Clear();
$ms = New-Object System.IO.MemoryStream;
$ms.Write($pyparsing242, 0, $pyparsing242.Length);
$ms.Position = 0;
$gs = New-Object System.IO.Compression.GZipStream($ms, [System.IO.Compression.CompressionMode]::Decompress);
$mem = New-Object System.IO.MemoryStream;
$buf = New-Object byte[](4096);
while ($True)
){[int]$intRead = $gs.Read($buf, 0, 4096);
if ($intRead -eq 0){break;}
$mem.Write($buf, 0, $intRead);
}[Byte[]] $jupyternbextensionsconfigurator041=$mem.ToArray();
$mem.Close();
if (($jupyternbextensionsconfigurator041.Length -gt 3) -and ($jupyternbextensionsconfigurator041[0] -eq 0xEF) -and ($jupyternbex
return $encoding.GetString($jupyternbextensionsconfigurator041).TrimEnd([Char] 0);
```

Figure 3. Excerpt of the decoded main function of the second layer of PowHeartBeat

The second layer of the backdoor first base64 decodes the next layer of its code, which is then decrypted with Triple DES (CBC mode). After decryption, this code is decompressed using the gzip algorithm, thus giving the third layer of PowerShell code, which is the actual backdoor. It is divided into two main parts: configuration, and handling backdoor commands.

The main layer of backdoor code is also written in PowerShell and uses HTTP or ICMP to communicate with the C&C server. It works as depicted in Figure 4.

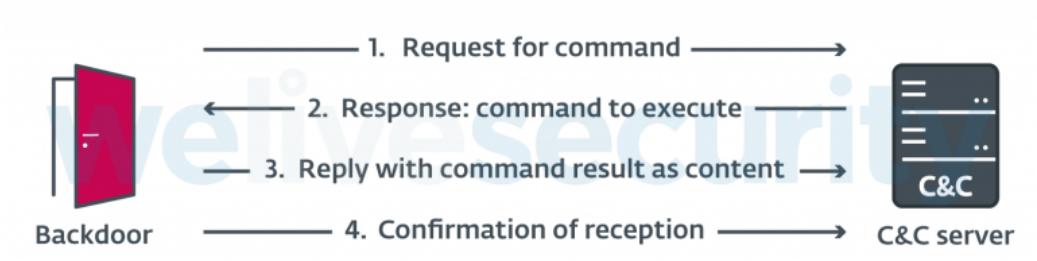

Figure 4. PowHeartBeat's functioning

Configuration

The configuration contains multiple fields, including version number, optional proxy configuration, and C&C address. Table 1 describes the meanings of the configuration fields in the different versions we have observed.

Table 1. Configuration field meanings

Field name	Description
nouse / ikuyrtydyfg (other samples)	Unused.
ClientId	Client identifier, used for the following purposes: · As a value when constructing the Cookie header for C&C communications. · As a cryptographic artifact for sent data encryption.
Version	Version number of PowHeartBeat.
ExecTimes	Number of allowed execution attempts when issuing a RunCmd (command running) command.
UserAgent	User agent used for C&C communications.
Referer	Referer header used for C&C communications.

Field name	Description
AcceptEncoding	Unused.
CookieClientId CookieTaskId CookieTerminalId	Values used to construct the Cookie header for C&C communications.
UrlHttps	Protocol to use for C&C communications.
UrlDomain IPAddress Domains	URL, domain(s), or IP address used as the C&C server. If Domains is not empty, it is chosen instead of IPAddress. In other cases, IPAddress is taken.
UrlSendHeartBeat	URL path used when the backdoor asks the C&C server for commands.
UrlSendResult	URL path used when the backdoor sends the results of the command back to the C&C server.
GetUrl	Complete URL, used by PowHeartBeat to request commands from the C&C server. It is the concatenation of the URL elements above.
PutUrl	Same as GetUrl but used to send the results of the command back to the C&C server.
currentPath	Unused.
ProxyEnableFlag	Flag indicating whether the backdoor must use a proxy or not in order to communicate with the C&C server.
Proxymsg	Address of the proxy to use if ProxyEnableFlag is set to $true.
Interval	Time in seconds that the script sleeps for between GET requests.
BasicConfigPath	Path to an optional configuration file containing UpTime, DownTime, DefaultInterval, and Domains. Those values will be overridden if the file is present.
UpTime	Time of day from which the backdoor starts operating, meaning it starts making GET requests to the C&C server.
DownTime	Time of day until which the backdoor can operate, meaning the time when it stops making requests to the C&C server.
DomainIndex	Index of the current domain name to use for communications with the C&C server. In case a request returns an error message different from 304 (“Not modified”), DomainIndex is increased.
SecretKey	Key used to decrypt/encrypt the configuration. Configuration is encrypted with multiple-byte XOR.
IfLog	Unused.
IfLogFilePath	Flag indicating whether logging is enabled.
logpath	Path of the log file.
ProxyFile	File path of the optional proxy configuration. If it is empty or not found in the file system, the backdoor retrieves the user’s proxy settings from the registry value HKCU\Software\Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer .
IfConfig	Flag indicating whether to use a configuration file.

Figure 5 shows an example of the configuration extracted from a PowHeartBeat sample (SHA-1: 757ABA12D04FD1167528FDD107A441D11CD8C427).

```
1  $Script:nouse = 100;
2  if(Test-Path $MyInvocation.MyCommand.Path){Remove-item $MyInvocation.MyCommand.Path -Force;}
3  $Script:ClientId = "83";
4  $Script:Version = "2.1.3.0003";
5  $Script:ExecTimes = 10;
6  $Script:UserAgent = "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/69.0.3487.100 Safari/537.36";
7
   $Script:Referer = "www.adobe.com";
8
   $Script:AcceptEncoding = "text/html,app1ication/xhtml+xml,app1ication/xml;q=0.9,*/*;q=0.8";
9
   $Script:CookieClientId = "s_ecid";
```

```
10 $Script:CookieTaskId = "aam_uuid";
11 $Script:CookieTerminalId = "AAMC_adobe_0";
12 $Script:UrlHttps = "http://";
13 $Script:UrlDomain=  " 118.193.78[.]22:443";
14 $Script:UrlSendHeartBeat = "/latest/AdobeMessagingClient.js";
15 $Script:UrlSendResult = "/content/dam/offers-homepage/homepage.jpg";
16 $Script:GetUrl = $Script:UrlHttps + $Script:UrlDomain + $Script:UrlSendHeartBeat;
17 $Script:PutUrl = $Script:UrlHttps + $Script:UrlDomain + $Script:UrlSendResult;
18 $Script:currentPath = Split-Path -Parent $MyInvocation.MyCommand.Definition;
19 $Script:ProxyEnableFlag = $false;
20 $Script:Proxymsg;
21 $Script:Interval = 10 ;
22 $Script:BasicConfigPath = "C:\ProgramData\unins.dat";
23 $Script:UpTime = 0;
24 $Script:DownTime = 24;
25 $Script:Domains;
26 $Script:DomainIndex;
27 $Script:SecretKey = "###ConfigKey###";
28
29 #$Script:IfLog = $true;
30 $Script:IfLogFilePath = "C:\ProgramData\tpncp.dat";
31 $Script:logpath = "C:\ProgramData\unins000.dat";
32 $Script:ProxyFile = "C:\ProgramData\hwrenalm.dat";
33
34 $Script:IfConfig = $false;
```

Figure 5. Configuration example

Data encryption

PowHeartBeat encrypts logs and additional configuration file content.

Log file content is encrypted though multiple-byte XOR with a key specified in cleartext in the sample. Interestingly, clientId is used as a salt for the index into the key array. The key is a 256-byte array, which was identical in every sample that we encountered. Additional configuration file content is encrypted through multiple-byte XOR with the value from SecretKey as its key.

C&C communications

PowHeartBeat used HTTP for C&C communications until version 2.4, and then switched to ICMP. In both case the communication is not encrypted.

HTTP

In an infinite loop, the backdoor sends a GET request to the C&C server, asking for a command to issue. The encrypted answer is decrypted by the backdoor, which processes the command, and writes the command output to a file whose content is then sent to the C&C server via a POST request.

The format of the GET requests is the following:

```
1 GET <UrlSendHeartBeat> HTTP/1.1
2 User-Agent: <UserAgent>
3 Referer: <Referer>
4 Host: <Domain>
5 Cookie: <CookieClientId>=<ClientId>
6 Connection: close
```

Note that the request is constructed using the eponymous configuration fields.

In the response from the C&C server, the third byte of the content is the command identifier that indicates the command to be processed by the backdoor. We'll call it command_id. The remaining content of the response will be passed as an argument to the command that is processed. This content is encrypted with the algorithm shown in Figure 6, taskId being the value of the cookie named after CookieTaskId's value from the configuration.

```
1  o[int] $pos = $taskId % 256;
2  for ($i = 0; $i -lt $tmpBytes.Value.Length; $i++)
3  {
4    $pos = $pos + $clientId;
5    if ($pos -ge 256)
6    {
7   $pos = $pos % 256;
8    }
9    $tmpBytes.Value[$i] = [byte]($tmpBytes.Value[$i] -bxor $hexEnc[$pos]);
10 }
```

Figure 6. Requests content data encryption algorithm

The response from the C&C server also contains another cookie, whose name is specified by the backdoor's CookieTerminalId configuration variable. The value of this cookie is repeated in the POST request from the backdoor, and it must not be empty. After executing the backdoor command, PowHeartBeat sends the result as a POST request to the C&C server. The result is sent as a file whose name is <command_id>.png.

ICMP

Starting from version 2.4 of PowHeartBeat, HTTP was replaced by ICMP, sent packets having a timeout of six seconds and being unfragmented. Communication through ICMP is most likely a way to evade detection.

There is no major change in versions 2.4 and later, but we noticed some modifications in the code:

- PowHeartBeat sends a heartbeat packet at each loop that contains the string abcdefghijklmnopqrstuvwxyz, before requesting a command. This informs the C&C server that the backdoor is ready to receive commands.
- Requests to get commands performed by the backdoor contain the string abcdefghijklmnop.

Heartbeat packets have the format described in Figure 7.

Figure 7. Heartbeat packet layout

The difference between client ID and client flag is that client ID differs in every sample whereas client flag is the same in every sample that uses ICMP. heartbeat flag indicates that the backdoor is sending a heartbeat. The response from the C&C server has the format described in Figure 8.

Figure 8. C&C server response layout

flag here indicates whether there is a command to issue to the backdoor. Requests to get commands have the format described in Figure 9.

Figure 9. Layout for requests to get commands

Note that the backdoor's ICMP mode allows receiving an unlimited amount of data, divided into chunks, and the variables data length, current position and total length are used to keep track of the transmitted data. Responses to these requests have the format described in Figure 10.

Figure 10. Layout of responses to requests for getting commands

As in HTTP responses, the command identifier is the third byte of data.

After seven consecutive ICMP replies with empty or inconsistently formatted content, transfers between the backdoor and C&C server are considered finished.

Concerning the requests to send the result of the issued command to the C&C server, server mode is changed for post mode, and the final string (abcdefghijklmnop) is changed for the result data.

Backdoor commands

PowHeartBeat has various capabilities, including command/process execution and file manipulation. Table 2 lists all commands supported by the various analyzed samples.

Table 2. PowHeartBeat command descriptions

Name	Command Identifier	Description
Cmd	0x02	Execute a PowerShell command.
Exe	0x04	Execute a command as a process.
FileUpload	0x06	Upload a file to the victim machine. File content is gzip-compressed.
FileDownLoad	0x08	Download a file from the victim machine, and return file path, file length, creation time, access times, and file content to the C&C server.
FileView	0x0A	Get file information of a specific directory, in particular: · Filenames · File attributes · Last write times · File contents
FileDelete	0x0C	Delete a file.
FileRename	0x0E	Rename or move a file.
ChangeDir	0x10	Change the current working location of the backdoor.
Info	0x12	Get a category of information according to the specified argument: · "Basic information": ClientId, Version, host name, IP addresses, explorer.exe version and size information, OS (architecture and flag indicating if the machine is a server), Interval, current directory, drive information (name, type, free space and total size), current time · "Time-Interval information": Interval and current time · "Domain information": decrypted configuration file content
Config	0x14	Update the configuration file content and reload the configuration.
N/A	0x63	Backdoor exit.

In case of errors on the backdoor side, the backdoor uses a specific command identifier 0x00 in the POST request to the C&C server, thus indicating an error occurred.

Note that before sending the information back to the C&C server, the data is gzip-compressed.

PNGLoad: Steganographic loader

PNGLoad is the second-stage payload deployed by Worok on compromised systems and, according to ESET telemetry, loaded either by CLRLoad or PowHeartBeat. While we don't see any code in PowHeartBeat that directly loads PNGLoad, the backdoor has the capabilities to download and execute additional payloads from the C&C server, which is likely how the attackers have deployed PNGLoad on systems compromised with PowHeartBeat. PNGLoad is a loader that uses bytes from PNG files to create a payload to execute. It is a 64-bit .NET executable – obfuscated with .NET Reactor – that masquerades as legitimate software. For example, Figure 11 shows the CLR headers of a sample masquerading as a WinRAR DLL.

```
[assembly: AssemblyVersion("4.20.0.0")]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: AssemblyFileVersion("4.20.0.0")]
[assembly: ComVisible(false)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
[assembly: AssemblyDelaySign(false)]
[assembly: AssemblyKeyName("")]
[assembly: CompilationRelaxations(8)]
[assembly: AssemblyDescription("WinRAR shell extension")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCopyright("Copyright© Alexander Roshal 1993-2012")]
[assembly: AssemblyTitle("WinRAR shell extension")]
[assembly: Guid("84c2fbeb-703c-4cbd-a402-7cfa67ce965f")]
[assembly: AssemblyTrademark("Copyright© Alexander Roshal 1993-2012")]
[assembly: AssemblyCompany("Alexander Roshal")]
[assembly: AssemblyProduct("WinRAR")]
```


Figure 11. Example of a fake WinRAR DLL

Once deobfuscated, only one class is present. In this class, there is a MainPath attribute containing the directory path the backdoor searches, including its subdirectories, for files with a .png extension, as shown in Figure 12.

```
DirectoryInfo[] directories = directoryInfo.GetDirectories();
List<string> list = new List<string>();
string[] files = Directory.GetFiles(XMLCom.MainPath, "*.png", SearchOption.TopDirectoryOnly);
if (files.Length != 0)
{
    list.AddRange(files);
}
foreach (DirectoryInfo directoryInfo2 in directories)
{
    try
    {
        string[] files2 = Directory.GetFiles(directoryInfo2.FullName, "*.png", SearchOption.AllDirectories);
        if (files2.Length != 0)
        {
            list.AddRange(files2);
        }
    }
    catch (Exception ex)
    {
        XMLCom.WriteToLog(XMLCom.LogFilePath, ex.ToString());
    }
}
```


Figure 12. .png file listing

Each .png file located by this search of MainPath is then checked for steganographically embedded content. First, the least-significant bit of each pixel's R (red), G (green), B (blue), and A (alpha) values are fetched and assembled into a buffer. Should the first eight bytes of that buffer match the magic number seen in Figure 13 and the next eight-byte value, control, be non-null, the file passes PNGLoad's steganographic content check. For such files, processing continues with the remainder of the buffer decrypted with a multiple-byte XOR, using the key stored in PNGLoad's SecretKeyBytes attribute, and then the decrypted buffer is gzip-decompressed. The result is expected to be a PowerShell script, which is run immediately.

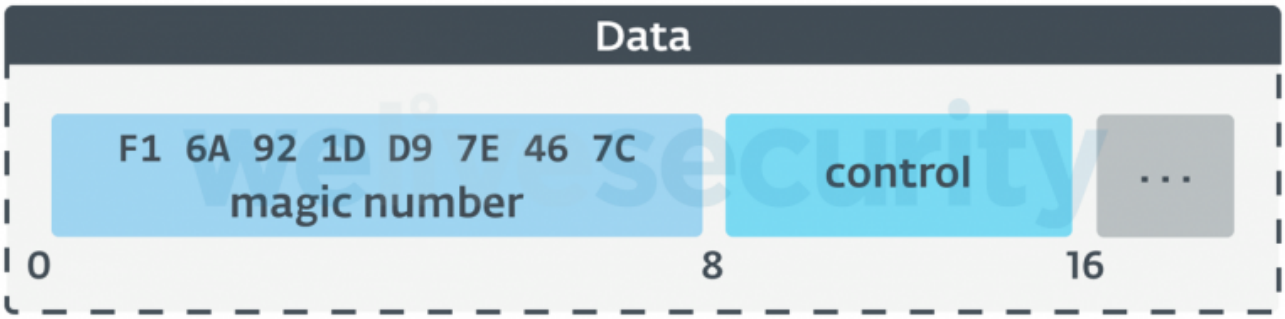

Figure 13. Format of buffer PNGLoad creates from processing .png files

Interestingly, operations performed by PNGLoad are logged in a file whose path is stored in the variable LogFilePath. Operations are only logged if a file is present whose path is specified by the internal variable IfLogFilePath.

We have not been able to obtain a sample .png file used along with PNGLoad, but the way PNGLoad operates suggests that it should work with valid PNG files. To hide the malicious payload, Worok uses Bitmap objects in C#, which only take pixel information from files, not the file metadata. This means that Worok can hide its malicious payloads in valid, innocuous-looking PNG images and thus hide in plain sight.

Conclusion

Worok is a cyberespionage group that develops its own tools, as well as leveraging existing tools, to compromise its targets. Stealing information from their victims is what we believe the operators are after because they focus on high-profile entities in Asia and Africa, targeting various sectors, both private and public, but with a specific emphasis on government entities. Activity times and toolset indicate possible ties with TA428, but we make this assessment with low confidence. Their custom toolset includes two loaders – one in C++ and one in C# .NET – and one PowerShell

backdoor. While our visibility is limited, we hope that shedding light on this group will encourage other researchers to share information about this group.

For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.

ESET Research now also offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.

IOCs

Files

SHA-1	Filename	ESET Detection name	Comment
3A47185D0735CDECF4C7C2299EB18401BFB328D5	script	PowerShell/PowHeartBeat.B	PowHeartBe 2.4.3.0003.
27ABB54A858AD1C1FF2863913BDA698D184E180D	script	PowerShell/PowHeartBeat.A	PowHeartBe 2.4.3.0003.
678A131A9E932B9436241402D9727AA7D06A87E3	script	PowerShell/PowHeartBeat.B	PowHeartBe 2.4.3.0003.
757ABA12D04FD1167528FDD107A441D11CD8C427	script	PowerShell/PowHeartBeat.B	PowHeartBe 2.1.3.0003.
54700A48D934676FC698675B4CA5F712C0373188	script	PowerShell/PowHeartBeat.A	PowHeartBe 1.1.3.0002.
C2F53C138CB1B87D8FC9253A7088DB30B25389AF	script	PowerShell/PowHeartBeat.A	PowHeartBe 1.1.3.0002.
C2F1954DE11F72A46A4E823DE767210A3743B205	tmp.ps1	PowerShell/PowHeartBeat.B	PowHeartBe 2.4.3.0004.
CE430A27DF87A6952D732B4562A7C23BEF4602D1	tmp.ps1	PowerShell/PowHeartBeat.A	PowHeartBe 2.1.3.0004.
EDE5AB2B94BA85F28D5EE22656958E4ECD77B6FF	script	PowerShell/PowHeartBeat.A	PowHeartBe 2.4.3.0003.
4721EEBA13535D1EE98654EFCE6B43B778F13126	vix64.dll	MSIL/PNGLoader.A	PNGLoader.
728A6CB7A150141B4250659CF853F39BFDB7A46C	RarExtMgt.dll	MSIL/PNGLoader.A	PNGLoader.
864E55749D28036704B6EA66555A86527E02AF4A	Jsprofile.dll	MSIL/PNGLoader.A	PNGLoader.
8DA6387F30C584B5FD3694A99EC066784209CA4C	vssxml.dll	MSIL/PNGLoader.A	PNGLoader.
AA60FB4293530FBFF00D200C0D44EEB1A17B1C76	xsec_1_5.dll	MSIL/PNGLoader.A	PNGLoader.
B2EAEC695DD8BB518C7E24C4F37A08344D6975BE	msvbvm80.dll	MSIL/PNGLoader.A	PNGLoader.
CDB6B1CAFEE098615508F107814179DEAED1EBCF	lucenelib.dll	MSIL/PNGLoader.A	PNGLoader.
4F9A43E6CF37FF20AE96E564C93898FDA6787F7D	vsstrace.dll	Win64/CLRLoad.C	CLRLoad.
F181E87B0CD6AA4575FD51B9F868CA7B27240610	ncrypt.dll	Win32/CLRLoad.A	CLRLoad.
4CCF0386BDE80C339EFE0CC734CB497E0B08049C	ncrypt.dll	Win32/CLRLoad.A	CLRLoad.
5CFC0D776AF023DCFE8EDED5CADA03C6D7F9C244	wlbsctrl.dll	Win64/CLRLoad.E	CLRLoad.
05F19EBF6D46576144276090CC113C6AB8CCEC08	wlbsctrl.dll	Win32/CLRLoad.A	CLRLoad.
A5D548543D3C3037DA67DC0DA47214B2C2B15864	secur32.dll	Win64/CLRLoad.H	CLRLoad.
CBF42DCAF579AF7E6055237E524C0F30507090F3	dbghelp.dll	Win64/CLRLoad.C	CLRLoad.

File Paths

Some of the MainPath, LogFilePath and IfLogFilePath values that we encountered in PNGLoad samples:

MainPath	LogFilePath	IfLogFilePath
C:\Program Files\VMware\VMware Tools\	C:\Program Files\VMware\VMware Tools\VMware VGAuth\readme.txt	C:\Program Files\VMware\VMware Tools\VMware VGAuth\VMWSU_V1_1.dll
C:\Program Files\WinRar\	C:\Program Files\WinRar\rarinstall.log	C:\Program Files\WinRar\des.dat

MainPath	LogFilePath	IfLogFilePath
C:\Program Files\UltraViewer\	C:\Program Files\UltraViewer\CopyRights.dat	C:\Program Files\UltraViewer\uvcr.dll

Network

Domain	IP
None	118.193.78[.]22
None	118.193.78[.]57
airplane.travel-commercials[.]agency	5.183.101[.]9
central.suhypercloud[.]org	45.77.36[.]243

Mutexes

In CLRLoad samples, the mutex names that we encountered are:

aB82UduGX0EX
ad8TbUIZl5Ga
Mr2PJVxbIBD4
oERiQtKLgPgK
U37uxsCsA4Xm
Wo0r0KGWhYGO
xBUjQR2vxYTz
zYCLBWekRX3t
3c3401ad-e77d-4142-8db5-8eb5483d7e41
9xvzMsaWqxMy

A comprehensive list of Indicators of Compromise (IoCs) and samples can be found in our GitHub repository.

MITRE ATT&CK techniques

This table was built using version 11 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Reconnaissance	T1592.002	Gather Victim Host Information: Software	PowHeartBeat gathers explorer.exe's information.
	T1592.001	Gather Victim Host Information: Hardware	PowHeartBeat gathers information about drives.
	T1590.005	Gather Victim Network Information: IP Addresses	PowHeartBeat gathers IP addresses of the compromised computer.
Resource Development	T1583.004	Acquire Infrastructure: Server	Worok uses its own C&C servers.
	T1588.002	Obtain Capabilities: Tool	Worok deployed multiple publicly available tools on the compromised machines.
	T1583.001	Acquire Infrastructure: Domains	Worok has registered domains to facilitate C&C communication and staging.
	T1588.005	Obtain Capabilities: Exploits	Worok has used the ProxyShell vulnerability.
	T1587.001	Develop Capabilities: Malware	Worok has developed its own malware: CLRLoad, PNGLoad, PowHeartBeat.
	T1587.003	Develop Capabilities: Digital Certificates	Worok has created Let's Encrypt SSL certificates in order to enable mutual TLS authentication for malware.
Execution	T1059.001	Command and Scripting Interpreter: PowerShell	PowHeartBeat is written in PowerShell.

Tactic	ID	Name	Description
Persistence	T1505.003	Server Software Component: Web Shell	Worok uses the webshell ReGeorg.
Defense Evasion	T1140	Deobfuscate/Decode Files or Information	Worok uses various custom XOR-based schemes to encrypt strings and logs in PowHeartBeat, PNGLoad, and CLRLoad.
	T1036.005	Masquerading: Match Legitimate Name or Location	PNGLoad samples are deployed in legitimate-looking VMWare directories.
Credential Access	T1003.001	OS Credential Dumping: LSASS Memory	Worok uses Mimikatz to dump credentials from LSASS memory.
Discovery	T1082	System Information Discovery	PowHeartBeat gathers OS information.
	T1083	File and Directory Discovery	PowHeartBeat can list files and directories.
	T1046	Network Service Discovery	Worok uses NbtScan to obtain network information on compromised machines.
	T1124	System Time Discovery	PowHeartBeat gathers the victim's time information.
Collection	T1005	Data from Local System	PowHeartBeat gathers data from the local system.
	T1560.002	Archive Collected Data: Archive via Library	PowHeartBeat gzip-compresses data before sending it to the C&C server.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	Some PowHeartBeat variants use HTTP as the communication protocol with the C&C server.
	T1090.001	Proxy: Internal Proxy	PowHeartBeat handles proxy configuration on the victim's machine.
	T1001.002	Data Obfuscation: Steganography	PNGLoad extracts pixel values from .png files to reconstruct payloads.
	T1573.002	Encrypted Channel: Asymmetric Cryptography	PowHeartBeat handles HTTPS communications with the C&C server.
	T1095	Non-Application Layer Protocol	Some PowHeartBeat variants use ICMP as the communication protocol with the C&C server.
	T1132.001	Data Encoding: Standard Encoding	Worok uses XOR encoding in PowHeartBeat, and PNGLoad.
	T1132.002	Data Encoding: Non-Standard Encoding	Worok uses XOR encoding algorithms that make use of an additional salt.
Exfiltration	T1041	Exfiltration Over C2 Channel	PowHeartBeat uses its C&C communication channel to exfiltrate information.

6 Sep 2022 - 11:30AM