

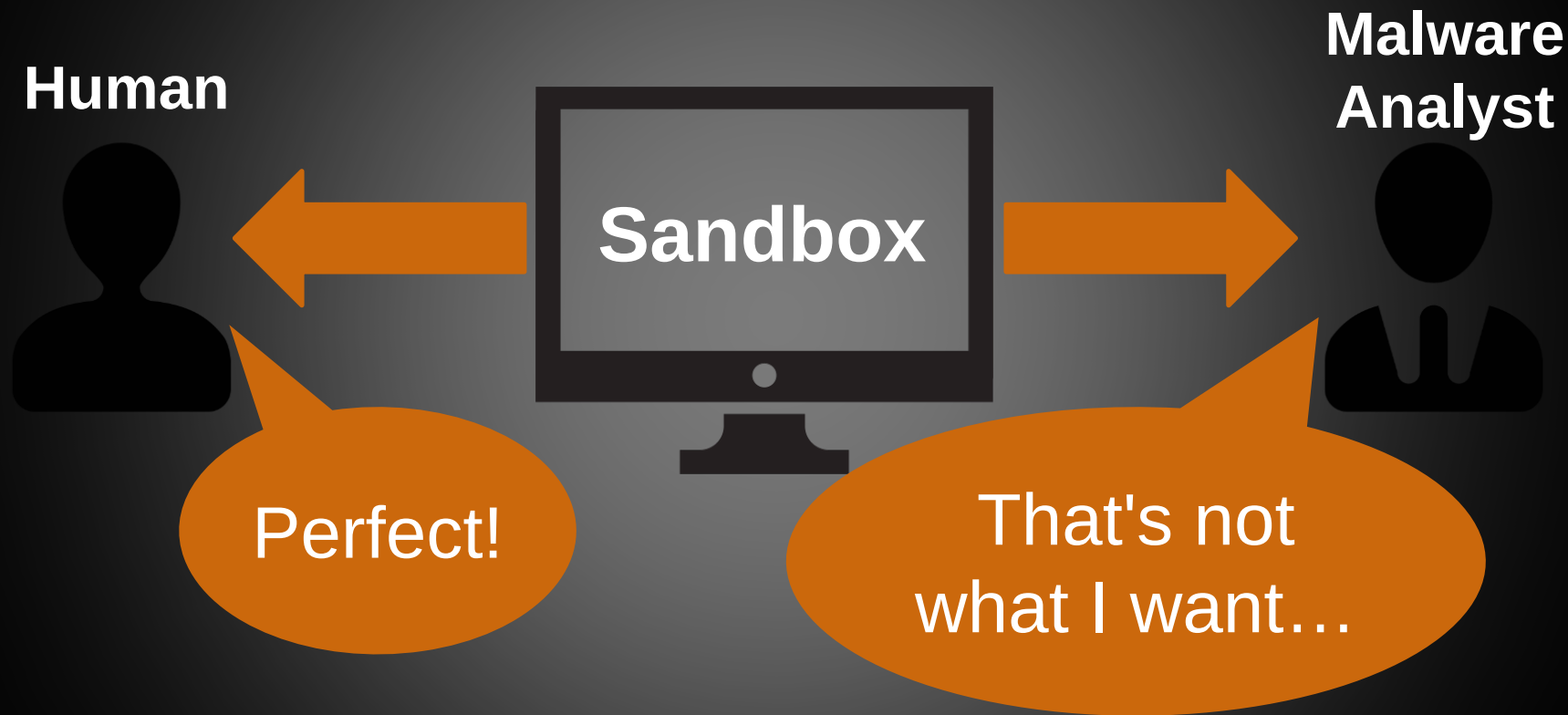
DEF CON DEMO LAB



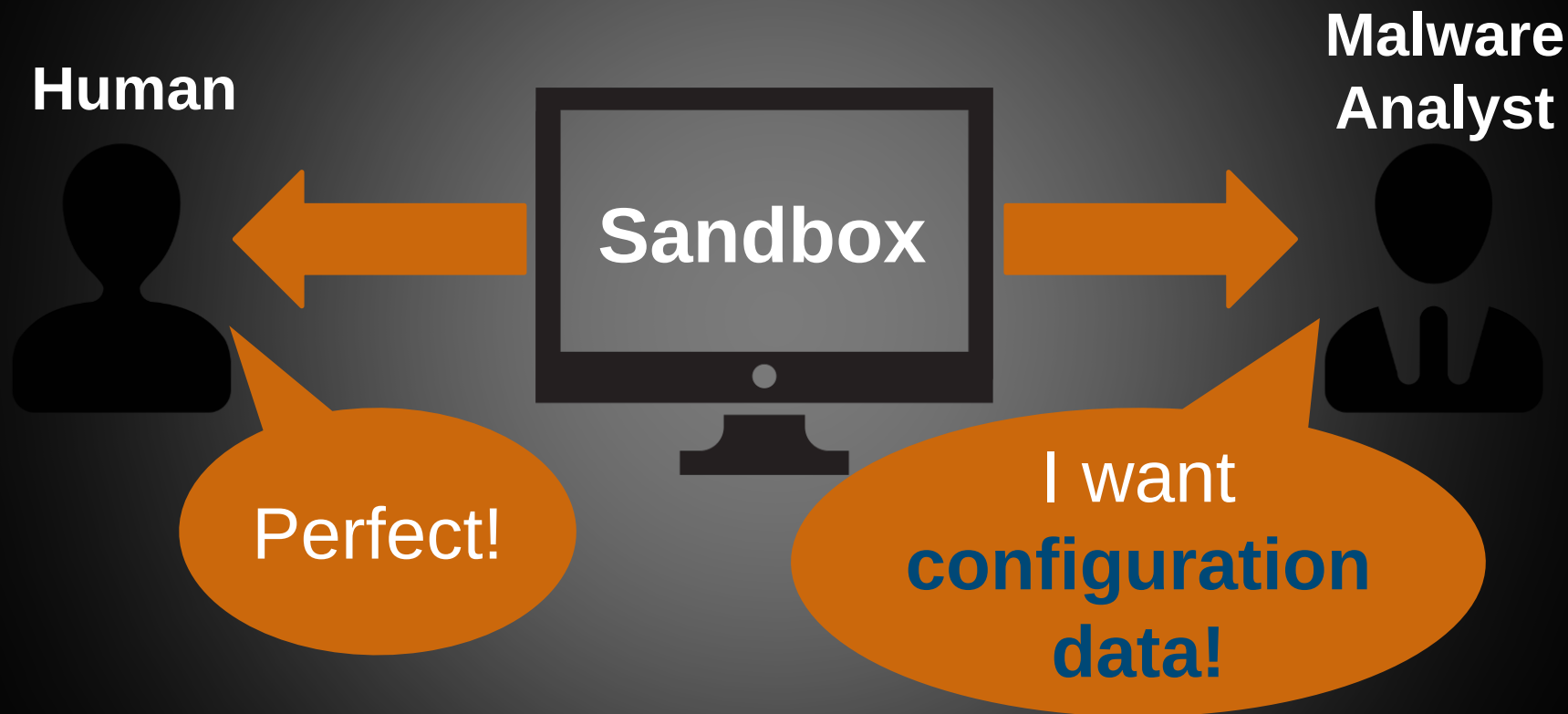
with MalConfScan



Motivation



Motivation



Why do we need malware configuration data?

Many variants of malware code are almost unchanged, and only configuration data is different.

- If the configuration data is known, there is no need for static analysis.

Configuration data contains important information that cannot be obtained by Sandbox analysis.

- Including campaign id, encryption key etc.

It's very simple.

Step 1

Malware Analysis

- Understand encryption techniques
- Understand configuration structures

How to Extract Malware Configuration Data Manually

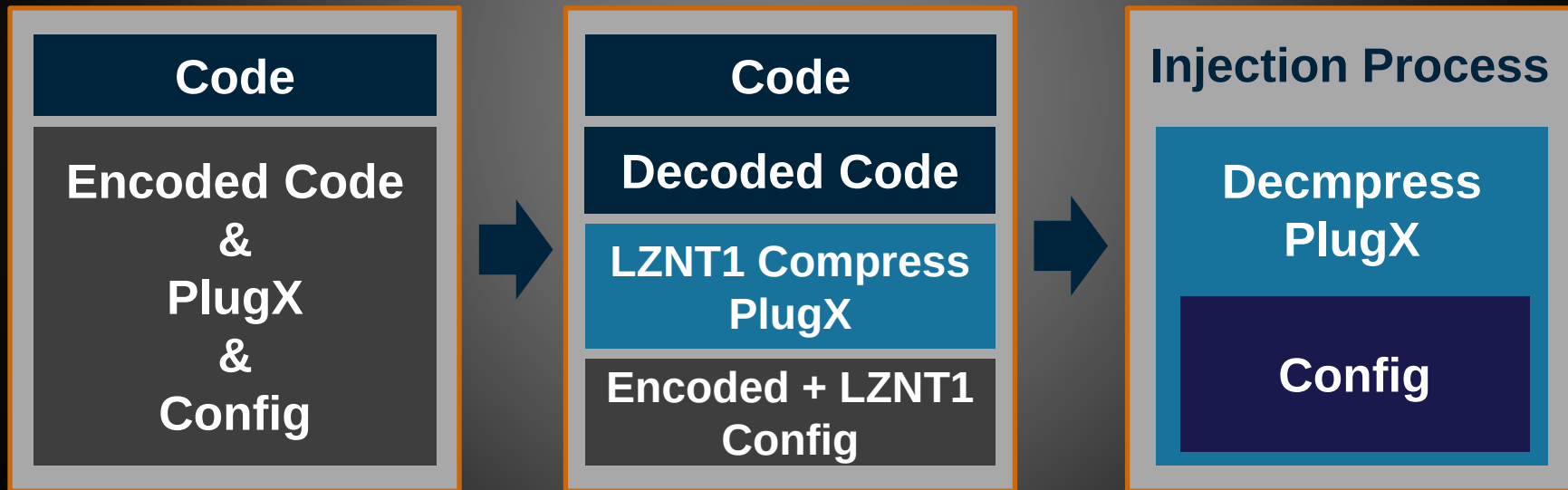
Step 2

Create tool

That's all.

How to Extract PlugX Configuration

In PlugX data,
PlugX main module and configuration are encoded.



PlugX Encoding Method

PlugX uses a custom encoding method.

Config size 0x2540

```
for i in range(strings_len-4):  
    set1 = (((decode_key << 7) - (decode_key >> 3) + i + 1899663297) >> 16) ^ (((decode_key << 7) - (decode_key >> 3) + i + 1899663297) >> 24) & 0xffffffff  
    decode_key = ((decode_key << 7) - (decode_key >> 3) + i + 1899663297) & 0xffffffff  
    result_data = (decode_key & 0xff) ^ ((decode_key >> 8) & 0xff) ^ encode_data[i] ^ set1 & 0xff
```



Config size 0x36A4

```
key1 = encode_key ^ 20140918  
key2 = encode_key ^ 353  
  
for i in range(strings_len):  
    key1 += 3373 & 0xffffffff  
    key2 -= 39779 & 0xffffffff  
    result_data = ord(encode_data[i]) ^ (((key2 >> 16) & 0xff ^ ((key2 & 0xff ^ (((key1 >> 16) & 0xff ^ (key1 - (key1 >> 8) & 0xff))  
- (key1 >> 24) & 0xff)) - (key2 >> 8) & 0xff)) - (key2 >> 24) & 0xff)
```

PlugX Configuration Structure

```
1 struct PlugXConfig
2 {
3     int hide_flag;
4     int field_4;
5     int field_8;
6     int field_C;
7     int field_10;
8     int DeleteFlag;
9     int KeylogFlag;
10    int field_1C;
11    int UnknownFlag;
12    int field_24;
13    char timer1_1;
14    char timer1_2;
15    char timer1_3;
16    char timer1_4;
17    char timer2_1;
18    char timer2_2;
19    char timer2_3;
20    char timer2_4;
21    char host_access_timetable[672];
22    char DNS[16];
23    HostInfo host1;
24    HostInfo host2;
25    HostInfo host3;
26    HostInfo host4;
27    HostInfo host5;
28    HostInfo host6;
29    HostInfo host7;
30    HostInfo host8;
31    HostInfo host9;
32    HostInfo host10;
33    HostInfo host11;
34    HostInfo host12;
35    HostInfo host13;
36    HostInfo host14;
37    HostInfo host15;
38    HostInfo host16;
39    char HTTP[2048];
40    char Proxy1[196];
41    char Proxy2[196];
42    char Proxy3[196];
43    char Proxy4[196];
44    int Install_mode;
45    char InstallDir[512];
46    char ServiceName[512];
47    char ServiceDisplayName[512];
48    char ServiceComment[512];
49    int InstallRegKey;
50    char RunKeyName[512];
51    char RunKeyValueName[512];
52    int InjectionFlag;
53    char Process1[512];
54    char Process2[512];
55    char Process3[512];
56    char Process4[512];
57    int UACFlag;
58    char UACProcess1[512];
59    char UACProcess2[512];
60    char UACProcess3[512];
61    char UACProcess4[512];
62    char ID1[512];
63    char ID2[512];
64    char Mutex[512];
65
66    int delete_file_time;
67    char ScreenCaptureFile[512];
68    int val1;
69    __int16 val1_port1;
70    __int16 val1_2;
71    int val2;
72    __int16 val2_port1;
73    __int16 val2_2;
74    int val3;
75    __int16 val3_port1;
76    __int16 val3_2;
77    int val4;
78    __int16 val4_port1;
79    __int16 val4_2;
80    int IPRangeSearchFlag;
81    int ipdstart1;
82    int ipstart2;
83    int ipstart3;
84    int ipstart4;
85    int ipend1;
86    int ipend2;
87    int ipend3;
88    int ipend4;
89    char val14;
90    char val15;
91    char val16;
92    char val17;
93    char val18;
94    char val19;
95    char field_36A2;
96    char field_36A3;
97 };
```

How to Extract TSCookie Configuration

TSCookie uses only RC4 for encryption.



TSCookie Configuration Structure

```
1 struct config
2 {
3     int field_0;
4     int port1_1;
5     int port1_2;
6     int server_flag1;
7     char server1[240];
8     int field_100;
9     int port2_1;
10    int port2_2;
11    int server_flag2;
12    char server2[240];
13    int field_200;
14    int port3_1;
15    int port3_2;
16    int server_flag3;
17    char server3[240];
18    int field_300;
19    int port4_1;
20    int port4_2;
21    int server_flag4;
22    char server4[240];
23    char proxy_server[128];
24    int proxy_port;
25    char field_484[124];
26    char id[256];
27    int field_600;
28    int rc4_key;
29    char field_608[660];
30    int sleep_time;
31 };
```

What is MalConfScan?

- **MalConfScan** is a Volatility plugin that extracts configuration data of known malware.
- Volatility is an open-source memory forensics framework for incident response and malware analysis.
- MalConfScan searches for malware in memory images and dumps configuration data.



Example (RedLeaves configuration data)

```
mal@works:/opt/vol2.4$ python vol.py malconfscan -f mem.image --profile=Win7SP1x86 -p 3872
```

```
Volatility Foundation Volatility Framework 2.4
```

```
[+] Searching memory by Yara rules.
```

```
[+] Detect malware by Yara rules.
```

```
[+] Process Name      : iexplore.exe
```

```
[+] Process ID       : 3872
```

```
[+] Malware name      : Lavender
```

```
[+] Base Address(VAD) : 0x530000
```

```
[+] Size              : 0xCC000
```

```
-----  
Process: iexplore.exe (3872)
```

```
[Config Info]
```

```
Server1      : firefoxcomt.arkouowi.com
```

```
Server2      : update.arkouowi.com
```

```
Server3      : firefoxcomt.arkouowi.com
```

```
Server4      : update.arkouowi.com
```

```
Port         : 443
```

```
Mode         : TCP and HTTP
```

```
ID           : 2018-1-8-NewBattle
```

```
Mutex        : jH10689DS
```

```
Key          : babybear
```

```
UserAgent    : Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2
```

```
Proxy server  :
```

```
Proxy username :
```

```
Proxy password :
```

Supported Malware Families

Supported Malware Families		
Ursnif	TSCookie	AZORult
Emotet	TSC_Loader	NanoCore RAT
Smoke Loader	xxmm	AgentTesla
PoisonIvy	Datper	FormBook
CobaltStrike	Ramnit	NodeRAT
NetWire	HawkEye	njRAT
PlugX	Lokibot	TrickBot
RedLeaves	Bebloh	Remcos
QuasarRAT	AsyncRAT 	WellMess 
ELF_PLEAD 		

Supported Malware Families

	 ANY.RUN @anyrun_app	
Ursnif	TOP10 last week's threats report:	AZORult
Emotet		NanoCore RAT
Smoke Loader	↓ #AgentTesla 147 (164)	AgentTesla
PoisonIvy	↓ #NanoCore 146 (154)	FormBook
CobaltStrike	↓ #Lokibot 142 (150)	NodeRAT
NetWire	↑ #Servhelper #TA505 81 (10)	njRAT
PlugX	↑ #Formbook 75 (54)	TrickBot
RedLeaves	↑ #Azorult 66 (39)	Remcos
QuasarRAT	↑ #Trickbot 57 (11)	WellMess 
ELF_PLEAD 	↓ #Ursnif 53 (60)	
	↓ #njRAT 52 (69)	
	↑ #HawkEye  48 (47)	

Why use Volatility?

Advantages of Dumping Configuration Data from Memory

No Need to Unpack

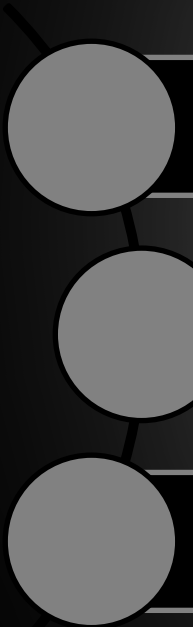
- Unpacking malware is not necessary when extracting configuration data.

No Need to Decode

- Configuration data may be already decoded.
- No need to know how to decrypt configuration data.

In Addition

- This tool also dumps more than configuration data if needed.



- Configuration Data

- Decoded Strings

- DGA Domains

Example (Bebloh configuration data and DGAs)

```
mal@works:/opt/vol2.4$ python vol.py malconfscan -f mem.image --profile=Win7SP1x86 -p 2724
Volatility Foundation Volatility Framework 2.4
[+] Searching memory by Yara rules.
[+] Detect malware by Yara rules.
[+] Process Name      : explorer.exe
[+] Process ID       : 2724
[+] Malware name      : Bebloh
[+] Base Address(VAD) : 0x1B0000
[+] Size              : 0x10000
-----
Process: explorer.exe (2724)

[Config Info]
RSA key          : 0602000000a4000052534131000400000100010071406719fdbaf787464a0b6b1c4bb589e897b8a4e1
9cd2aee225c643e9afd3cb2c67ee4c4980a954c1ba7464d3e669be70177c76d3498383fdb7e6aad684d6911e722f238ee900659c0.
afe2b2db12a882edaa5b83eba0843bcb12392f931d850681d4e7900b99f95d02d3cb9c
Sleep count      : 0
Seed URL         : benistora.com/auth/
Sleep time       : 180
Botid            : 6CEEC4FA6817A661A9
Registry subkey   : ecywhvdabais
Network Chack    : Enable
Botnet           : 0000000000000001
Registry key     :
DGA 0            : benistora.com
DGA 1            : tndhxdvll.net
DGA 2            : faqpohoit.com
DGA 3            : mixjavydp.net
DGA 4            : g5oqoa3f5.com
DGA 5            : g5mvbekzi.net
DGA 6            : 5qdgoweg.com
DGA 7            : zqynvek4u.net
DGA 8            : rhv1zax3c.com
DGA 9            : ypygujsi2e.net
DGA 10           : prs5c4jsi39h.com
DGA 11           : zg9p2ojkl.net
DGA 12           : ckpj3ylqzfr9.com
DGA 13           : 9j1ctowlvueinc.net
```

Example (FormBook decoded strings)

```
mal@works:/opt/vol2.4$ python vol.py malconfscan -f mem.image --profile=Win7SP1x86 -p 1948
Volatility Foundation Volatility Framework 2.4
[+] Searching memory by Yara rules.
[+] Detect malware by Yara rules.
[+] Process Name      : explorer.exe
[+] Process ID       : 1948
[+] Malware name      : Formbook
[+] Base Address(VAD) : 0x7BD0000
[+] Size             : 0x142000
-----
Process: explorer.exe (1948)

[Config Info]
C&C URI 1      : www.trogmack.com/br/
Encoded string 0 : USERNAME
Encoded string 1 : LOCALAPPDATA
Encoded string 2 : USERPROFILE
Encoded string 3 : APPDATA
Encoded string 4 : TEMP
Encoded string 5 : ProgramFiles
Encoded string 6 : CommonProgramFiles
Encoded string 7 : ALLUSERSPROFILE
Encoded string 8 : /c copy "
Encoded string 9 : /c del "
Encoded string 10 : \Run
Encoded string 11 : \Policies
Encoded string 12 : \Explorer
Encoded string 13 : \Registry\User
Encoded string 14 : \Registry\Machine
Encoded string 15 : \SOFTWARE\Microsoft\Windows\CurrentVersion
Encoded string 16 : Office\15.0\Outlook\Profiles\Outlook\
Encoded string 17 : NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook\
Encoded string 18 : \SOFTWARE\Mozilla\Mozilla
Encoded string 19 : \Mozilla
Encoded string 20 : Username:
Encoded string 21 : Password:
```

Additional Feature

- **malstrscan** function can list strings to which the hollowed process refers.

Configuration data is usually encoded by malware.

Most of malwares writes decoded configuration data on memory.

This feature list decoded configuration data when possible.

Example

```
mal@works:/opt/vol2.4$ python vol.py malstrscan -f mem.image --profile=Win7SP1x86 -p 1260
Volatility Foundation Volatility Framework 2.4
[+] Searching for malicious memory space.
[+] Detect Process Hollowing space.
[+] Process Name      : svchost.exe
[+] Process ID       : 1260
[+] Base Address(VAD) : 0x15190000
[+] Size             : 0x3D000
[+] Vad Protection    : PAGE_EXECUTE_READWRITE
-----
Process: svchost.exe (1260)

0x151BE5FC: SeDebugPrivilege
0x151BE610: ntdll.dll
0x151BE61C: RtlCreateUserThread
0x151BE630: Advapi32.dll
0x151BE640: RegDeleteKeyExA
0x151BE650: open
0x151BE658: kernel32
0x151BE664: GetSystemWow64DirectoryA
0x151BE680: kernel32.dll
0x151BE690: GetNativeSystemInfo
0x151BE6A4: IsWow64Process
0x151BE6B4: com.%s.sdb
0x151BE6C0: %s\cmd.%s.bat
0x151BE6D0: username
0x151BE6DC: \.\.\.\LocalLow\
0x151BE6F0: start " " "%s"
0x151BE700: "%windir%\%s\iscsicli.exe"
0x151BE720: system32
0x151BE72C: syswow64
0x151BE738: /q "%s"
```

More Additional Feature



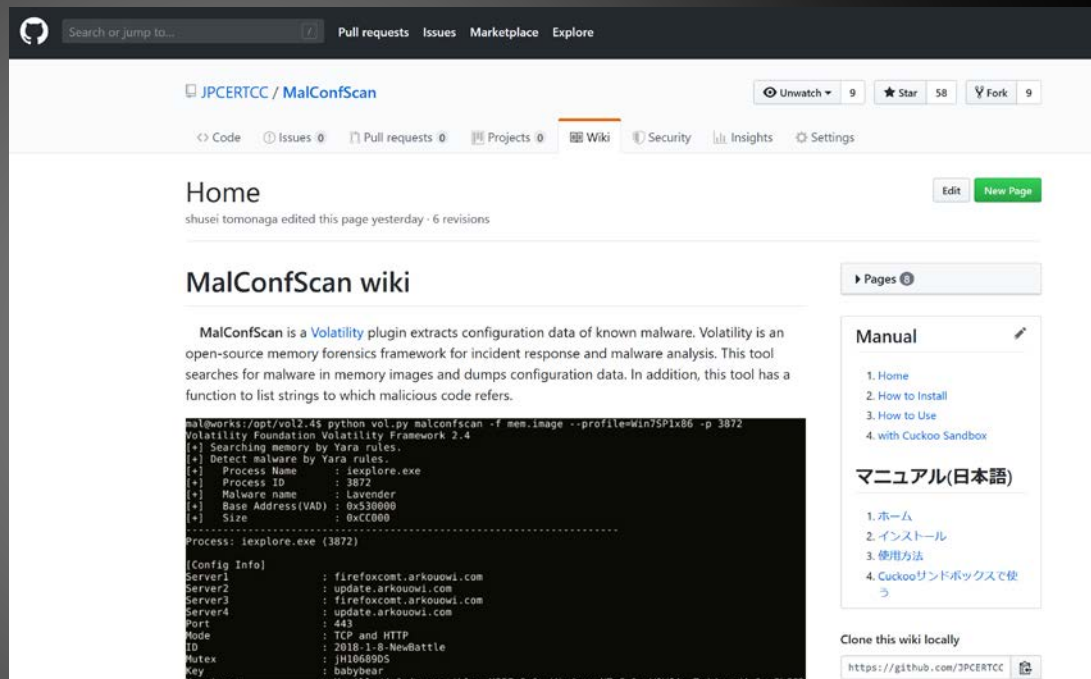
- **linux_malconfscan** searches for malware in Linux OS memory images and dumps configuration data.
- Few malware supported.
 - **WellMess**
 - **ELF_PLEAD**

DEMONSTRATION



How to Use

MalConfScan Wiki



The screenshot shows the GitHub repository page for JPCERTCC/MalConfScan. The 'Wiki' tab is selected, displaying the 'Home' page. The page content includes a description of MalConfScan as a Volatility plugin and a terminal output of the tool's execution. On the right side, there is a 'Manual' section with a list of links: 1. Home, 2. How to Install, 3. How to Use, and 4. With Cuckoo Sandbox. Below this, there is a section for 'マニュアル(日本語)' (Manual in Japanese) with links: 1. ホーム, 2. インストール, 3. 使用方法, and 4. Cuckooサンドボックスで使う. At the bottom right, there is a 'Clone this wiki locally' section with a link to the GitHub repository.

Home

shusei tomonaga edited this page yesterday · 6 revisions

MalConfScan wiki

MalConfScan is a [Volatility](#) plugin extracts configuration data of known malware. Volatility is an open-source memory forensics framework for incident response and malware analysis. This tool searches for malware in memory images and dumps configuration data. In addition, this tool has a function to list strings to which malicious code refers.

```
mal@works:/opt/vol2.4$ python vol.py malconfscan -f mem.image --profile=Win7SP1x86 -p 3872
Volatility Foundation Volatility Framework 2.4
[+] Searching memory by Yara rules.
[+] Detect malware by Yara rules.
[+] Process Name      : iexplore.exe
[+] Process ID       : 3872
[+] Malware name      : Lavender
[+] Base Address(VAD) : 0x530000
[+] Size              : 0xCC000
-----
Process: iexplore.exe (3872)
[Config Info]
Server1      : firefoxcont.arkouow1.com
Server2      : update.arkouow1.com
Server3      : firefoxcont.arkouow1.com
Server4      : update.arkouow1.com
Port         : 443
Mode         : TCP and HTTP
ID           : 2018-1-8-NewBattle
Mutex        : JH1069D5
Key          : babybear
HostAgent    : Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.3; WOW64; Trident/5.0; SLCC/
```

Manual

1. Home
2. How to Install
3. How to Use
4. With Cuckoo Sandbox

マニュアル(日本語)

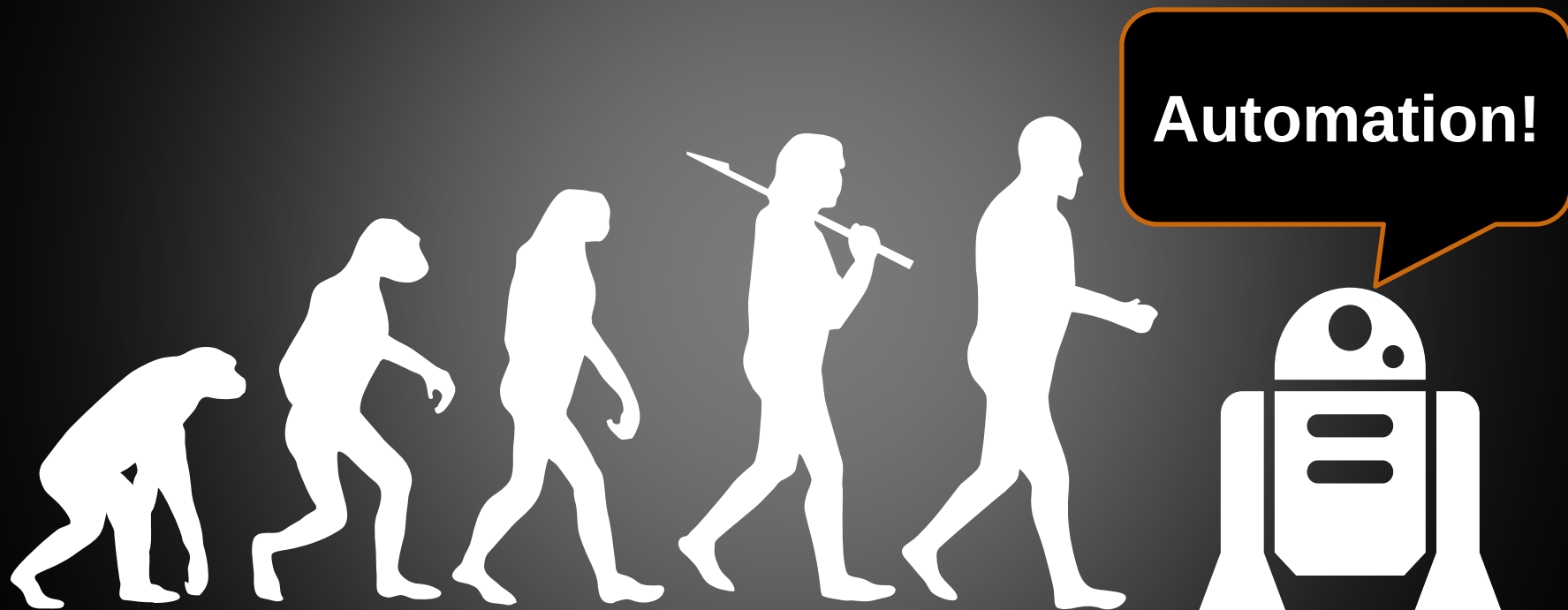
1. ホーム
2. インストール
3. 使用方法
4. Cuckooサンドボックスで使う

Clone this wiki locally

<https://github.com/JPCERTCC>

<https://github.com/JPCERTCC/MalConfScan/wiki>

Next Stage



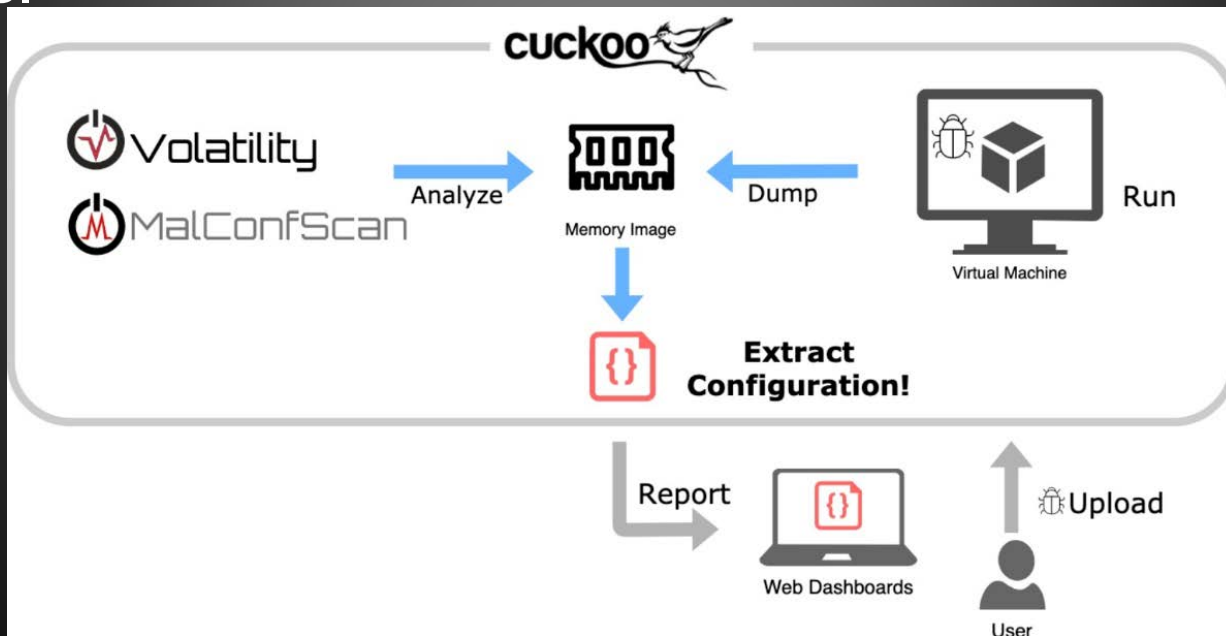
What is MalConfScan-with-Cuckoo?

- **MalConfScan-with-Cuckoo** is Cuckoo Sandbox plugin for MalConfScan.
- The plugin adds the function to extract known malware's configuration data from memory dump and add the MalConfScan report to Cuckoo Sandbox.



How it Works

- This tool uses Cuckoo's memory dump function to extract configuration data of executed malware from memory dumps.



Overview




Dashboard
Recent
Pending
Search

Submit
Import

VM Memory Dump

[Download Full Memory Dump](#)

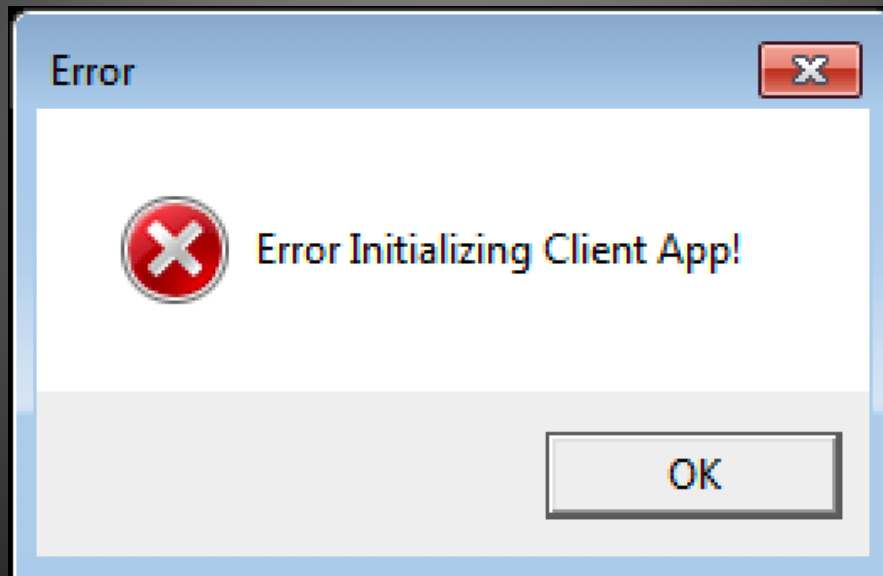
Process List	Services	Kernel Modules	Device Tree	Code Injection	Timers	Psxview
Messagehooks	API Hooks	Callbacks	Yarascan	SSDT	IDT	GDT
						Sockets
MalConfScan						

Process Name	PID	Malware Name	Base Address(VAD)	Size
iexplore.exe	2248	Himawari	0x04521984	0x00815104

Config Field	Config Value
Server1	diamond.ninth.biz
Server2	diamond.ninth.biz
Server3	diamond.ninth.biz
Server4	diamond.ninth.biz
Port	443
Mode	TCP and HTTP
ID	2017-11-28-MACRO
Mutex	Q34894iq
Key	usotsuki

Anti-Analysis

- Anti-Analysis functions disturbs the analysis in sandbox
- Some of the malware have these functions
 - Ursnif variants (targeting Japan) etc.



Anti-analysis techniques

■ Generic

- Language settings
- Execution after reboot
- Total physical memory
- Count of processors etc.

■ Virtualization

- CPUID (CPU brand, virtualization setting, etc.)
- Device info (Device name, MAC address, etc.)
- Registry keys etc.

■ Processes

- Process name (wireshark, OllyDbg, Process Monitor, etc.)

Configure your VM.

How to configure you VM

Ursnif have some anti-analysis functions.

```
.text:00401C02 27C 8D 44 24 10    lea     eax, [esp+278h+enc_warning_msg]
.text:00401C06 27C E8 32 13 00 00    call    mal_decode_0
.text:00401C08 27C 8B D8            mov     ebx, eax
.text:00401C0D 27C E8 C4 F7 FF FF    call    mal_chk_cpuid
.text:00401C12 27C 85 C0            test    eax, eax
.text:00401C14 27C 74 12            jz      short loc_401C28
```

```
.text:00401C28
.text:00401C28
.text:00401C28 27C E8 26 F8 FF FF    call    mal_chk_vm_strings
.text:00401C2D 27C 85 C0            test    eax, eax
.text:00401C2F 27C 75 E5            jnz     short loc_401C16
```

```
.text:00401C31 27C FF 15 58 50 40 00    call    ds:GetTickCount
.text:00401C37 27C 3D A8 61 00 00    cmp     eax, 25000
.text:00401C3C 27C 76 09            jbe     short loc_401C47
```

```
.text:00401C3E 27C E8 1D F6 FF FF    call    chk_cpuid_execution_speed
.text:00401C43 27C 85 C0            test    eax, eax
.text:00401C45 27C 75 CF            jnz     short loc_401C16
```

CPU Brand Detection

Device Name Detection

Boot-time Detection

Debugger Detection

Anti-Analysis : CPU Brand Name Detection

```
push    80000002h
lea     eax, [ebp+var_3C]
push    eax
call    get_cpu_id_brand
push    80000003h
lea     eax, [ebp+var_2C]
push    eax
call    get_cpu_id_brand
push    80000004h
lea     eax, [ebp+var_1C]
push    eax
call    get_cpu_id_brand
push    esi
lea     eax, [ebp+var_3C]
push    eax
mov     [ebp+var_C], 0
call    cmp_cpu_brand
test    eax, eax
jz      short loc_401447
```

Call CUID opcode to dump the CPU brand name.

```
mov     eax, 8000000[2-4]h
__cuid
```

Check the CPU brand name if it includes "XEON".

Anti-Anti-Analysis: Fake the CPU Brand Name (VMware)

Fake the return value of CPUID with VM configuration

Insert following settings to your .vmx file

```
cpuid.80000002.0.eax = "0110:0101:0111:0100:0110:1110:0100:1001"  
cpuid.80000002.0.ebx = "0010:1001:0101:0010:0010:1000:0110:1100"  
cpuid.80000002.0.ecx = "0111:0010:0110:1111:0100:0011:0010:0000"  
cpuid.80000002.0.edx = "0100:1101:0101:0100:0010:1000:0110:0101"  
cpuid.80000003.0.eax = "0011:0101:0110:1001:0010:0000:0010:1001"  
cpuid.80000003.0.ebx = "0011:0101:0101:1001:0011:0111:0010:1101"  
cpuid.80000003.0.ecx = "0101:0000:0100:0011:0010:0000:0011:0100"  
cpuid.80000003.0.edx = "0010:0000:0100:0000:0010:0000:0101:0101"  
cpuid.80000004.0.eax = "0011:0000:0011:0010:0010:1110:0011:0001"  
cpuid.80000004.0.ebx = "0000:0000:0111:1010:0100:1000:0100:0111"  
cpuid.80000004.0.ecx = "0000:0000:0000:0000:0000:0000:0000:0000"  
cpuid.80000004.0.edx = "0000:0000:0000:0000:0000:0000:0000:0000"
```

Before After

システム

評価: システムの評価を利用できません。

プロセッサ: Intel(R) Xeon(R) CPU E5-2665 0 @ 2.40GHz 2.39 GHz



システム

評価: システムの評価を利用できません。

プロセッサ: Intel(R) Core(TM) i5-7Y54 CPU @ 1.20GHz 2.59 GHz (2 プロセッサ)

Anti-Analysis : Device Name Detection

```
if ( SetupDiGetDeviceRegistryPropertyA(  
    v0,  
    &DeviceInfoData,  
    0xCu,  
    &PropertyRegDataType,  
    (PBYTE)lpMem_Heap_0,  
    RequiredSize,  
    &RequiredSize) )  
{  
    lpMem_ = lpMem;  
    if ( cmp_strings(lpMem, str_vbox)  
        || cmp_strings(lpMem_, str_qemu)  
        || cmp_strings(lpMem_, str_vmware)  
        || cmp_strings(lpMem_, str_virtualhd) )  
    {  
        v8 = 1;  
    }  
}
```

Call Win32API to get the device name

Check the device name includes specific strings

Anti-Anti-Analysis: Modify the Device Name (VMware)

Modify the device name.

Insert following settings to your .vmx file

```
scsi0:0.productID = "Toshiba SSD"  
scsi0:0.vendorID = "Toshiba"  
scsi1:0.productID = "Toshiba SSD"  
scsi1:0.vendorID = "Toshiba"
```

Recommended setting for Anti-Anti-Analysis

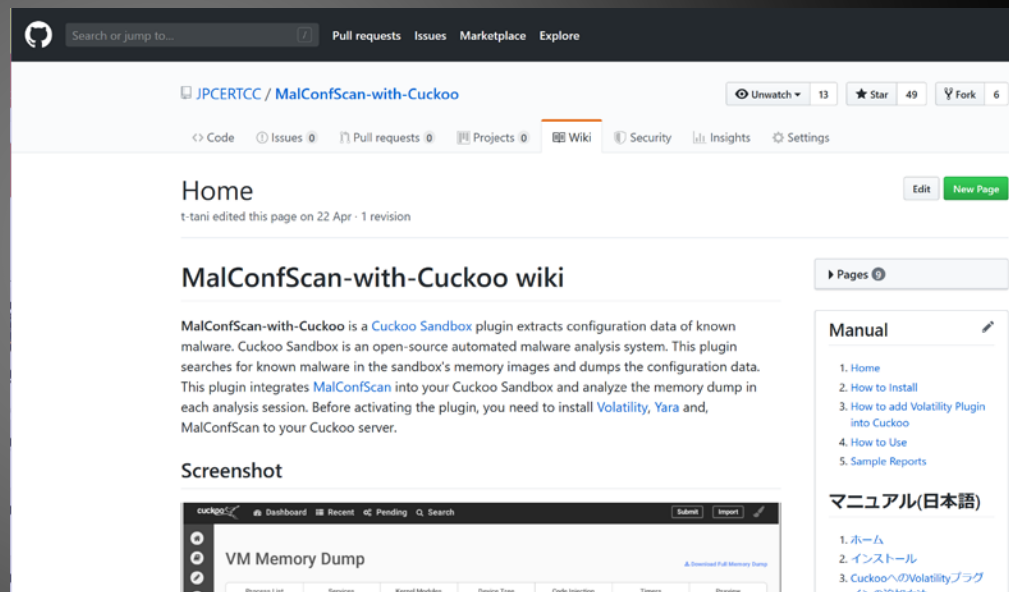
- Do NOT use VMware tools or VirtualBox guest additions.
- Use local language OS for VM
- Modify the CUID response
- Modify the Device name
- Modify the NIC (MAC address)

DEMONSTRATION



How to Use

MalConfScan with Cuckoo wiki



<https://github.com/JPCERTCC/MalConfScan-with-Cuckoo/wiki>

We are going to Volatility3!



MalConfScan will support Volatility3.

JPCERTCC / MalConfScan

27 257 39

<> Code Issues Pull requests Actions Wiki Security Insights Settings

Volatility3 2 branches 6 tags

Go to file Add file Code

This branch is 1 commit ahead of master.

Full request Compare

t-tani Support Volatility3. 2 days ago 40 commits

images	Support Volatility3.	2 days ago
utils	Support Volatility3.	2 days ago
yara	Added support wellness (win and linux)	24 days ago
gitignore	Released v0.0.1	16 months ago
LICENSE.txt	Released v0.0.1	16 months ago
README.md	Support Volatility3.	2 days ago
malconfscan.py	Support Volatility3.	2 days ago
requirements.txt	Released v0.0.1	16 months ago

README.md

MalConfScan

<https://github.com/JPCERTCC/MalConfScan/tree/Volatility3>

Thank you!



<https://github.com/JPCERTCC/MalConfScan-with-Cuckoo>

<https://github.com/JPCERTCC/MalConfScan>

Contact



@jpcert_en



ir-info@jpcert.or.jp

PGP <https://www.jpcert.or.jp/english/pgp/>