paloalto NETWORKS

Teasing the Secrets from Threat Actors

Malware Configuration Extractors

Mark Lim

- Senior Malware Researcher
  at Palo Alto Networks
- Based in Singapore
- Love to go for long jogs

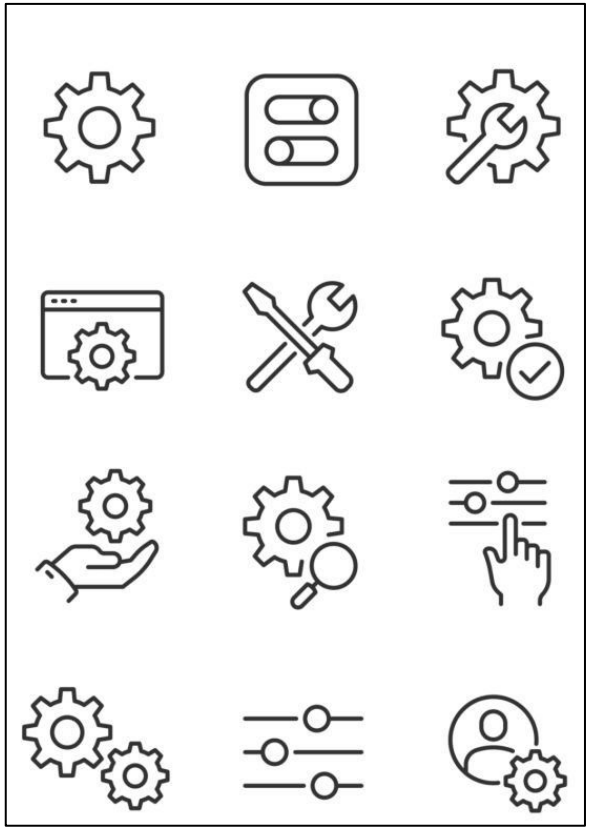
OUTLINE

- Background
- Evolutionary journey of Guloader configuration techniques
  - Ciphertext Splitting
  - Control Flow Obfuscation
- Summary

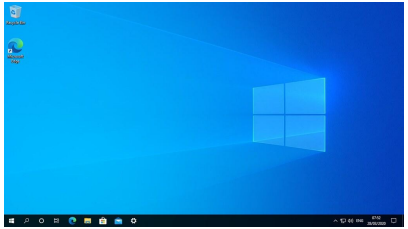
Background

What are malware configurations?

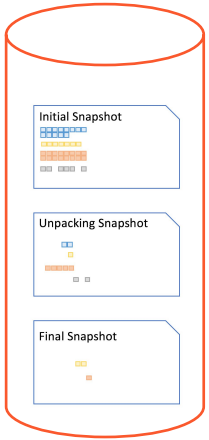
- Similar to ‘settings’ or ‘preferences’ in software
- Malware configuration defines the uniqueness of each instance
- C&C addresses, encryption keys, attack parameters and other IOCs
- <u>Tough to obtain statically</u>
- But can be extracted from process memory.



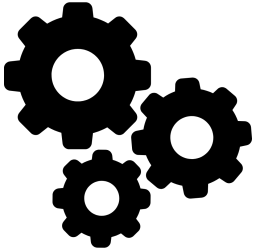
Malware Configuration Extraction Workflow



Sandbox

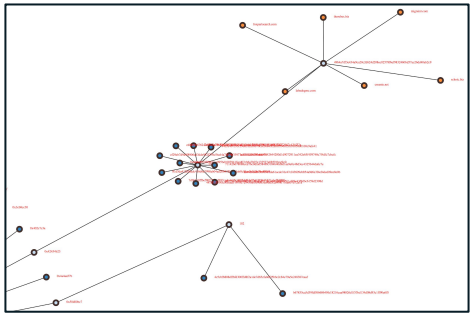


Snapshot Data



Memory Analysis and Malware Configuration Parsing

Malware Analysts Create Malware Config Extractors



Malware Configs indexed

What and Why Guloader ?

- Also known as CloudEye
- Windows Malware downloader
- Shellcode based
- Constantly evolving
- Utilised many anti-analysis techniques

Evolutionary Journey Of Guloader's Configuration Tactics

Evolutionary Journey Of Guloader's Configuration Tactics

2020

Hiding in plain sight

Search for the string "http" would reveal C2 url

2022

C2 url are encrypted and no longer starts with "http"

Q4 2022

Ciphertext Splitting

Ciphertext has to be decoded in blocks from a function before it can be used

Q1 2023

Control flow obfuscation

Control flow obfuscation progressively applied to increase the complexity of retrieving the ciphertext

Q4 2023

???

????

Evolutionary Journey Of Guloader's Configuration Tactics

2020

Hiding in plain sight

Search for the string "http" would reveal C2 url

2022

C2 url are encrypted and no longer starts with "http"

Q4 2022

Ciphertext Splitting

Ciphertext has to be decoded in blocks from a function before it can be used

Q1 2023

Control flow obfuscation

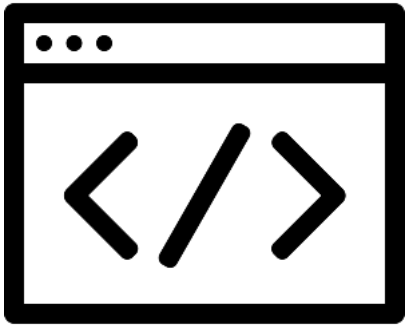
Control flow obfuscation progressively applied to increase the complexity of retrieving the ciphertext

Q4 2023

???

????

Decrypting Malware Configuration



Encryption routine

Encryption key

Ciphertext

Decrypting Malware Configuration

Simple
XOR

Predictable location of
encryption key

???

Ciphertext Splitting

A1CD2379F530FA9458A71CE2EED42D1BA3C125D9

1. Ciphertext splitted into multiple DWORD
2. Each DWORD is encoded with different arithmetic operations
3. Stored as local variables in functions

A1CD2379	F530FA94	58A71CE2	EED42D1B	A3C125D9
{+-&*&}	{^-^*}	{+-*&}	{&-^&}	{^&^*}
BC24D7A1	DE14CFD3	78CD13EF	34DABC34	EF3A32FD

Ciphertext Splitting

```
.text:00417423 ; Attributes: noreturn
.text:00417423
.text:00417423 Func_C2_config proc far
.text:00417423
.text:00417423 arg_0 = dword ptr  4
.text:00417423
.text:00417423        mov     ecx, [esp+arg_0]
.text:00417427        mov     dword ptr [ecx], 0CA49C0B7h
.text:0041742D        jmp     short loc_417434
```

```
.text:00417434
.text:00417434 loc_417434:
.text:00417434        xor     dword ptr [ecx], 6EC23534h
.text:0041743A        jmp     short loc_417446
```

```
.text:00417446
.text:00417446 loc_417446:
.text:00417446        xor     dword ptr [ecx], 6D0F0C6Dh
.text:0041744C        sub     dword ptr [ecx], 0C984F9D2h ; [1st Dword]
.text:00417452        jmp     short loc_41745B
```

```
B = [0×CA49C0B7, 0×6EC23534, 0×6D0F0C6D, 0×C984F9D2]
b1 = (B[0] ^ B[1]) & 0×FFFFFFFF
c1 = (b1 ^ B[2]) & 0×FFFFFFFF
result = (c1 - B[3]) & 0×FFFFFFFF
```

First DWORD is the length of the ciphertext!

Control Flow Obfuscation

```
.text:00401451 E8 6F C4 00+call    sub_40D945
.text:00401451 00
.text:00401456 89 5D 18    mov     [ebp+18h], ebx
.text:00401459 8B 4D 18    mov     ecx, [ebp+18h]
.text:0040145C CC          int     3                               ; Trap to Debugger
.text:0040145D A5          movsd
.text:0040145E 00 8E 7B 8E+add     [esi+6F348E7Bh], cl
.text:0040145E 34 6F
.text:00401464 CD 4C       int     4Ch                             ; Z100 - Slave 8259 - S100 v
.text:00401466 22 B3 BA 57+and     dh, [ebx-7465A846h]
.text:00401466 9A 8B
.text:0040146C A5          movsd
.text:0040146D E8 18 BA 00+call    sub_40CE8A
.text:0040146D 00
.text:00401472 89 85 98 00+mov     [ebp+98h], eax
.text:00401472 00 00
.text:00401478 CC          int     3                               ; Trap to Debugger
.text:00401479 A7          cmpsd
.text:0040147A 7E 00       jle     short $+2
.text:0040147C
.text:0040147C             loc_40147C:                             ; CODE XREF: sub_4013FC+7E↑j
.text:0040147C 72 8D       jb      short loc_40140B
.text:0040147E F6 C8 87    test    al, 87h
.text:00401481 1E          push    ds
.text:00401482 F4          hlt
```

Control Flow Obfuscation

```
pop     eax
xor     dword ptr [ebx], 0D62B9FCCh
int     3                           ; EXCEPTION_BREAKPOINT triggered!
-----------------------------------------------------------------------
db 3,13h,19h,15h,'2',12h,0AFh,'IMp|',0E9h,4Dh,4Dh ; junk bytes
-----------------------------------------------------------------------
add     dword ptr [ebx], 3FA3DA06h
sub     dword ptr [ebx], 0E5EEDCD8h
int     3                           ; EXCEPTION_BREAKPOINT triggered!
-----------------------------------------------------------------------
db 1Dh,'l',0Fh,0B6h,0A3h,8Fh,'o',0A1h,0D5h,34h,0BAh,0AFh,'l',0F6h,8Eh,89h,0B5h ; junk bytes
db 0A9h
-----------------------------------------------------------------------
add     [eax], eax
add     [esi+6B042D54h], bh
push    ecx
```

0xCC bytes triggering
EXCEPTION_BREAKPOINT

```
mov     esi, 0D37212A5h
add     esi, 4C655390h
xor     esi, 1FD76635h              ; esi=0
mov     [esi], esi                  ; EXCEPTION_ACCESS_VIOLATION
-----------------------------------------------------------------------
db 1Ch,0C5h,74h,0C3h,4Ch,0F6h,8Fh,0FBh,0FAh,'36',92h,0Eh,0 ; junk bytes
-----------------------------------------------------------------------
pop     esi
mov     dword ptr [ebx], 70609D36h
push    eax
mov     eax, 0C2520E2Ah
add     eax, 7711832h
xor     eax, 6868751Ah
sub     eax, 0A1AB5346h             ; eax=0
mov     [eax], ecx                  ; EXCEPTION_ACCESS_VIOLATION
-----------------------------------------------------------------------
db 0Ah,0B8h                         ; junk bytes
db 0B8h,'T',0
-----------------------------------------------------------------------
pop     eax
```

Instructions triggering
EXCEPTION_ACCESS_VIOLATION

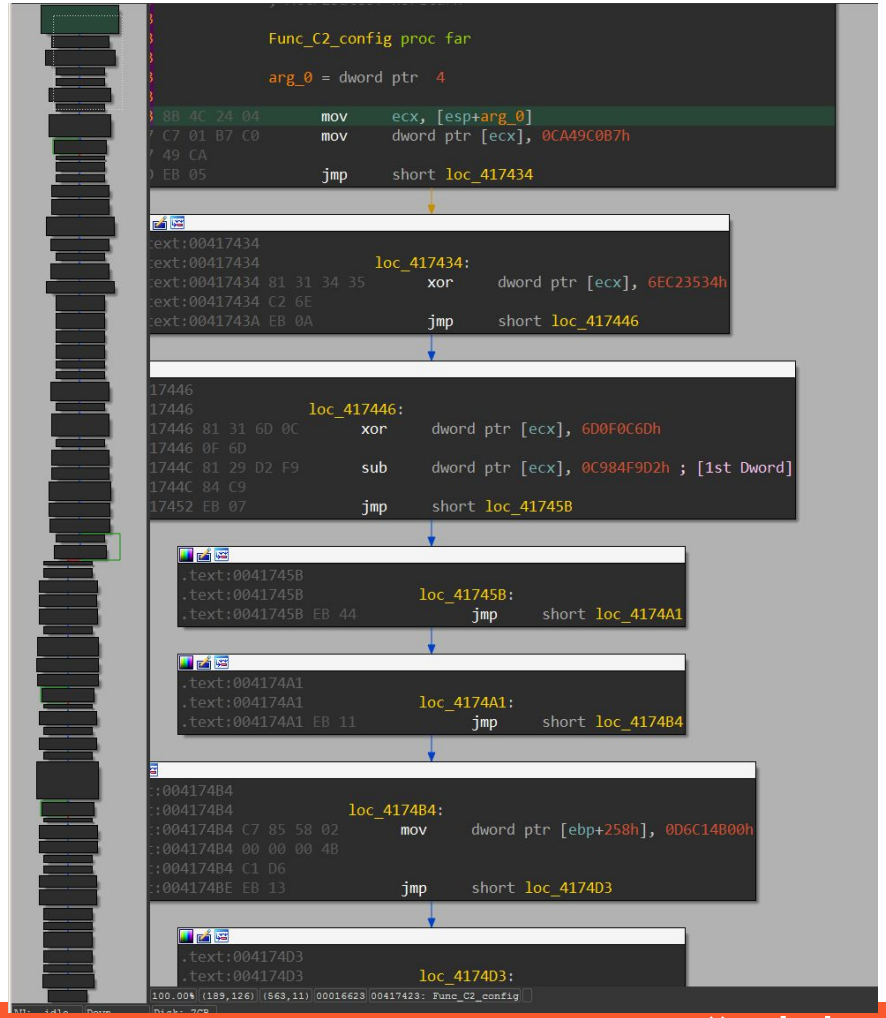
Control Flow Obfuscation

```
def fix_veh_all(data):
    AV_PATTERN = [b"\xBE.{4}\x81.{5}\x81.{5}\x89", b"\xB8.{4}\x05.{4}\x35.{4}\x2d.{4}\x89", b"\xB9.{4}\x81.{5}\x81.{5}\x89",
    b"\x35.{4}\x35.{4}\x35.{4}\x89", b"\x81.{5}\x81.{5}\x81.{5}\x89" ,b"\xBB.{4}\x81.{5}\x81.{5}\x89", b"\xB4.{4}\x2d.{4}\x35.
    {4}\x35.{4}\x89", b"\xB8.{4}\x2d.{4}\x35.{4}\x35.{4}\x89", b"\xBA.{4}\x81.{5}\x81.{5}\x89", b"\xB8.{4}\x2D.{4}\x2D.{4}
    \x05.{4}\x89", b"\x35.{4}\x35.{4}\x2D.{4}\x89", b"\xB8.{4}\x35.{4}\x35.{4}\x89"]
    CC_PATTERN = [b"\x81.{5}\xCC",b"\x8B.{5}\xCC",b"\x5F\x81.{5}\xCC"]
    key = 0×c
    i = 0
    result = dict()
```

Find and patch them all!

Control Flow Obfuscation

```
text:00417423 8B 4C 24 04      mov     ecx, [esp+4]
text:00417427 C7 01 B7 C0      mov     dword ptr [ecx], 0CA49C0B7h
text:00417427 49 CA
text:0041742D CC               int     3                       ; Trap to Debugger
text:0041742D                ; ---------------------------------------------------------
text:0041742E 65               db 65h
text:0041742F 2E               db 2Eh
text:00417430 A3               db 0A3h
text:00417431 19               db 19h
text:00417432 F1               db 0F1h
text:00417433 72               db 72h
text:00417434                ; ---------------------------------------------------------
text:00417434 81 31 34 35      xor     dword ptr [ecx], 6EC23534h
text:00417434 C2 6E
text:0041743A CC               int     3                       ; Trap to Debugger
text:0041743A                ; ---------------------------------------------------------
text:0041743B 6E               db  6Eh ; n
text:0041743C C4               db 0C4h
text:0041743D 78               db  78h ; x
text:0041743E 84               db  84h
text:0041743F C4               db 0C4h
text:00417440 C7               db 0C7h
text:00417441 BD               db 0BDh
text:00417442 2B               db  2Bh ; +
text:00417443 4E               db  4Eh ; N
text:00417444 5A               db  5Ah ; Z
text:00417445 B2               db 0B2h
text:00417446                ; ---------------------------------------------------------
text:00417446 81 31 6D 0C      xor     dword ptr [ecx], 6D0F0C6Dh
text:00417446 0F 6D
text:0041744C 81 29 D2 F9      sub     dword ptr [ecx], 0C984F9D2h
text:0041744C 84 C9
text:00417452 CC               int     3                       ; Trap to Debugger
text:00417452                ; ---------------------------------------------------------
text:00417453 6B               db  6Bh ; k
text:00417454 C6               db 0C6h
text:00417455 84               db  84h
text:00417456 80               db  80h
text:00417457 FA               db 0FAh
```



Sample 1 2023 Q2

Sample 1 2023 Q2

1. 0xCC bytes triggering **EXCEPTION_BREAKPOINT**
2. instructions triggering **EXCEPTION_SINGLE_STEP**
3. Instructions triggering **EXCEPTION_ACCESS_VIOLATION**

```
02B993E3 Func_C2_ciphertext_construct:
02B993E3     mov     eax, [esp+4]
02B993E7     int     3                               ; EXCEPTION_BREAKPOINT triggered!
02B993E7 ; ---------------------------------------------------------------------------
02B993E8     db 0B4h,0A1h,'l',13h,0D8h,0FFh,'`+',0A5h,0E3h,30h,26h,'t'
02B993F6 ; ---------------------------------------------------------------------------
02B993F6     push    eax
02B993F7     mov     eax, 433D4EDBh
02B993FC     xor     eax, 9436930Fh
02B99401     xor     eax, 0A78DF586h
02B99406     sub     eax, 392289D1h
02B9940B     sub     eax, 37639D81h                  ; eax=0x100
02B99410     push    ebx
02B99411     pushf
02B99412     mov     ebx, esp
02B99414     or      [ebx], eax                      ; enable Trap flag
02B99416     popf
02B99417     test    edi, ecx                        ; EXCEPTION_SINGLE_STEP triggered!
02B99417 ; ---------------------------------------------------------------------------
02B99419 aW  db 'w',6,0B7h,0B6h,0CFh,14h,'\',0ACh,0A6h,0B8h,'%',0
02B99425 ; ---------------------------------------------------------------------------
02B99425     cmp     ebx, ecx
02B99427     pop     ebx
02B99428     cmp     eax, edx
02B9942A     pop     eax
02B9942B     mov     dword ptr [eax], 3D6E3A33h
02B99431     push    edi
02B99432     mov     edi, 82A8E946h
02B99437     sub     edi, 9707FB40h
02B9943D     sub     edi, 0EBA0EE06h
02B99443     mov     [edi], edi
02B99445     mov     esp, offset unk_78C8DF
02B9944A     pop     edi
02B9944B     xor     dword ptr [eax], 8344C565h
02B99451     xor     dword ptr [eax], 74FD1098h
02B99457     sub     dword ptr [eax], 0CAD7EF8Ch
02B9945D     push    edi
02B9945E     mov     edi, 0AA2AD49Dh
02B99463     add     edi, 676C9CAAh
02B99469     add     edi, 0EE688EB9h                 ; edi = 0x0
02B9946F     mov     [edi], ecx                      ; EXCEPTION_ACCESS_VIOLATION triggered!
02B9946F ; ---------------------------------------------------------------------------
```

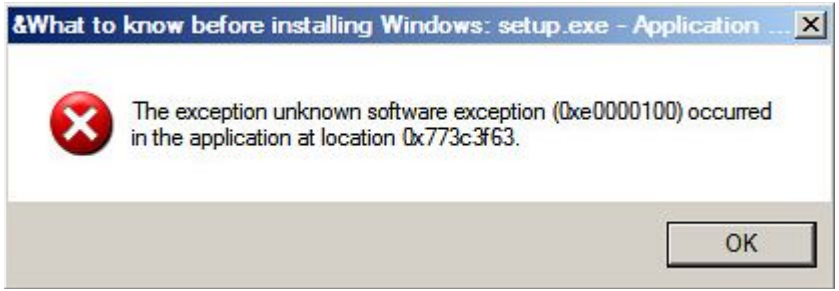
Instructions triggering single step exception!

Sample 1 2023 Q2

```
def fix_veh_all(data):
    AV_PATTERN = [b"\xBE.{4}\x81.{5}\x81.{5}                    {4}                    {4}\x2d.{4}\x89", b"\xB9.{4}\x81.{5}\x81.{5}\x89",
    b"\x35.{4}\x35.{4}\x35.{4}\x89", b"\x81.{5                    \xBB.{4}\x81.{5}\x81.{5}\x89", b"\xB4.{4}\x2d.{4}\x35.
    {4}\x35.{4}\x89", b"\xB8.{4}\x2d.{4}\x35.{4                    {4}\x81.{5}\x81.{5}\x89", b"\xB8.{4}\x2D.{4}\x2D.{4}
    \x05.{4}\x89", b"\x35.{4}\x35.{4}\x2D.{4}\x89",                    {4}\x35.{4}\x89"]
    CC_PATTERN = [b"\x81.{5}\xCC",b"\x8B.{5}\xCC",b"                    CC"]
    key = 0×c
    i = 0
    result = dict()
```

Does not work anymore!

Diving into implementation of control flow obfuscation



```
1  int __stdcall Func_VEH_handler(_EXCEPTION_POINTERS ExceptionInfo)
2  {
3    int ExceptionCode;
4    CONTEXT *Context;
5    _BYTE *Eip;
6    int offset;
7    _BYTE *i;
8
9    ExceptionCode = *(_DWORD *)ExceptionInfo.ExceptionRecord->ExceptionCode;
10   switch ( ExceptionCode )
11   {
12
13         case EXCEPTION_BREAKPOINT:
14             Context = (CONTEXT *)Func_Anti_HW_Breakpoints();      // return Context if no Hardware BP is detected
15             Eip = (_BYTE *)Context->Eip;
16             if ( *Eip == 0xCC )
17             {
18                 offset = Eip[1] ^ 0xBB;                                  // decrypt offset
19                 for ( i = (_BYTE *)(offset + Context->Eip - 1); (_BYTE *)(Context->Eip + 2) != i; --i )
20                 {
21                 if ( *i == 0xCC )                                  // anti debugging loop
22                     return 0;                                                                  // exit from handler if debugger is used
23                 }
24                 Context->Eip += offset;
25                 return EXCEPTION_CONTINUE_EXECUTION;                // exception is handled by Func_VEH_handler()
26             }
27             break;
28         case EXCEPTION_ACCESS_VIOLATION:
29             if ( *(_DWORD *)(ExceptionInfo.ExceptionRecord->ExceptionCode + offsetof(CONTEXT, Dr7)) )
30                 return 0;                                                                      // exit from handler if debugger is used
31             goto EXCEPTION_SINGLE_STEP_ACCESS_VIOLATION;
32
33         case EXCEPTION_SINGLE_STEP:
34
35         EXCEPTION_SINGLE_STEP_ACCESS_VIOLATION:                  //same code for handling both SINGLE STEP and ACCESS VIOLATION exceptions
36             Context = (CONTEXT *)Func_Anti_HW_Breakpoints();      // return Context if no Hardware BP is detected
37             Context->Eip += *(_BYTE *)(Context->Eip + 2) ^ 0xBB;  // add EIP with decrypted offset
38             return EXCEPTION_CONTINUE_EXECUTION;                  // exception is handled by Func_VEH_handler()
39   }
40   return 0;
41 }
```

Diving into implementation of control flow obfuscation

```
case EXCEPTION_BREAKPOINT:
    Context = (CONTEXT *)Func_Anti_HW_Breakpoints();      // return Context if no Hardware BP is detected
    Eip = (_BYTE *)Context->Eip;
    if ( *Eip == 0xCC )
    {
        offset = Eip[1] ^ 0xBB;                                  // decrypt offset
        for ( i = (_BYTE *)(offset + Context->Eip - 1); (_BYTE *)(Context->Eip + 2) != i; --i )
        {
        if ( *i == 0xCC )                                 // anti debugging loop
            return 0;                                                          // exit from handler
        }
        Context->Eip += offset;
        return EXCEPTION_CONTINUE_EXECUTION;                // exception is handled by Func_VEH_handler()
    }
    break;
```

0xCC bytes triggering
EXCEPTION_BREAKPOINT

Diving into implementation of control flow obfuscation

```
case EXCEPTION_BREAKPOINT:
    Context = (CONTEXT *)Func_Anti_HW_Breakpoints();      // return Context if no Hardware BP is detected
    Eip = (_BYTE *)Context->Eip;
    if ( *Eip == 0xCC )
    {
        offset = Eip[1] ^ 0xBB;                                  // decrypt offset
        for ( i = (_BYTE *)(offset + Context->Eip - 1); (_BYTE *)(Context->Eip + 2) != i; --i )
        {
        if ( *i == 0xCC )                                 // anti debugging loop
            return 0;                                                          // exit from handler
        }
        Context->Eip += offset;
        return EXCEPTION_CONTINUE_EXECUTION;                // exception is handled by Func_VEH_handler()
    }
    break;
```

0xCC bytes triggering
EXCEPTION_BREAKPOINT

Diving into implementation of control flow obfuscation

```
case EXCEPTION_BREAKPOINT:
    Context = (CONTEXT *)Func_Anti_HW_Breakpoints();      // return Context if no Hardware BP is detected
    Eip = (_BYTE *)Context->Eip;
    if ( *Eip == 0xCC )
    {
        offset = Eip[1] ^ 0xBB;                                  // decrypt offset
        for ( i = (_BYTE *)(offset + Context->Eip - 1); (_BYTE *)(Context->Eip + 2) != i; --i )
        {
        if ( *i == 0xCC )                                 // anti debugging loop
            return 0;                                                          // exit from handler
        }
        Context->Eip += offset;
        return EXCEPTION_CONTINUE_EXECUTION;                // exception is handled by Func_VEH_handler()
    }
    break;
```

0xCC bytes triggering
EXCEPTION_BREAKPOINT

Diving into implementation of control flow obfuscation

```
case EXCEPTION_BREAKPOINT:
    Context = (CONTEXT *)Func_Anti_HW_Breakpoints();      // return Context if no Hardware BP is detected
    Eip = (_BYTE *)Context->Eip;
    if ( *Eip == 0xCC )
    {
        offset = Eip[1] ^ 0xBB;                                  // decrypt offset
        for ( i = (_BYTE *)(offset + Context->Eip - 1); (_BYTE *)(Context->Eip + 2) != i; --i )
        {
        if ( *i == 0xCC )                                 // anti debugging loop
            return 0;                                                          // exit from handler
        }
        Context->Eip += offset;
        return EXCEPTION_CONTINUE_EXECUTION;                // exception is handled by Func_VEH_handler()
    }
    break;
```

0xCC bytes triggering
EXCEPTION_BREAKPOINT

Diving into implementation of control flow obfuscation

```
case EXCEPTION_BREAKPOINT:
    Context = (CONTEXT *)Func_Anti_HW_Breakpoints();      // return Context if no Hardware BP is detected
    Eip = (_BYTE *)Context->Eip;
    if ( *Eip == 0xCC )
    {
        offset = Eip[1] ^ 0xBB;                                  // decrypt offset
        for ( i = (_BYTE *)(offset + Context->Eip - 1); (_BYTE *)(Context->Eip + 2) != i; --i )
        {
        if ( *i == 0xCC )                                 // anti debugging loop
            return 0;                                                          // exit from handler
        }
        Context->Eip += offset;
        return EXCEPTION_CONTINUE_EXECUTION;                // exception is handled by Func_VEH_handler()
    }
    break;
```

0xCC bytes triggering
EXCEPTION_BREAKPOINT

Diving into implementation of control flow obfuscation

```
    break;
case EXCEPTION_ACCESS_VIOLATION:
    if ( *(_DWORD *)(ExceptionInfo.ExceptionRecord→ExceptionCode + offsetof(CONTEXT, Dr7)) )
        return 0;                                                    // exit from handler if debugger is used
    goto EXCEPTION_SINGLE_STEP_ACCESS_VIOLATION;

case EXCEPTION_SINGLE_STEP:

EXCEPTION_SINGLE_STEP_ACCESS_VIOLATION:          //same code for handling both SINGLE STEP and ACCESS VIOLATION exceptions
    Context = (CONTEXT *)Func_Anti_HW_Breakpoints();   // return Context if no Hardware BP is detected
    Context→Eip += *(_BYTE *)(Context→Eip + 2) ^ 0×BB;  // add EIP with decrypted offset
    return EXCEPTION_CONTINUE_EXECUTION;                // exception is handled by Func_VEH_handler()
```

- Instructions triggering **EXCEPTION_SINGLE_STEP**
- Instructions triggering **EXCEPTION_ACCESS_VIOLATION**

Diving into implementation of control flow obfuscation

- Hiding the single byte key

```
.text:04526FFB test    ax, dx
.text:04526FFE mov     ecx, 0FD80184Ah
.text:04527003 xor     ecx, 888D9523h
.text:04527009 test    cl, bl
.text:0452700B xor     ecx, 0E20CC336h
.text:04527011 cmp     dx, cx
.text:04527014 cmp     edx, ecx
.text:04527016 add     ecx, 6BCEB226h  ; ECX = 0x85 (Key)
.text:0452701C test    dh, ah
.text:0452701E mov     dl, [edx+1]     ; read encrypted offset byte
.text:04527021 test    ecx, ecx
.text:04527023 xor     dl, cl          ; decrypt offset with key
.text:04527025 movzx   edx, dl
.text:04527028 cmp     edx, ebx
.text:0452702A mov     ecx, [eax+CONTEXT._Eip]
.text:04527030 add     ecx, edx        ; add EIP with decrypted offset
.text:04527032 dec     ecx
```

Evolutionary Journey Of Guloader's Configuration Tactics

Sample 1 2023 Q2

1. Ciphertext splitting
2. 0xCC bytes triggering **EXCEPTION_BREAKPOINT**
3. Instructions triggering **EXCEPTION_ACCESS_VIOLATION**
4. Instructions triggering **EXCEPTION_SINGLE_STEP**

Evolutionary Journey Of Guloader's Configuration Tactics

Sample 1 2023 Q2

Evolutionary Journey Of Guloader's Configuration Tactics

Sample 1 2023 Q2

The Solution!

1. Using memory dumps from sandbox execution
2. Locate function containing splitted cipher text using yara
3. Locate the single byte key used in the exception handler via yara
4. Using Unicorn CPU emulator framework
5. Emulate the function containing the DWORD
6. Handle the 3 types of exceptions

Evolutionary Journey Of Guloader's Configuration Tactics

2020

Hiding in plain sight

Search for the string "http" would reveal C2 url

2022

C2 url no longer starts with "http"

Q4 2022

Ciphertext Splitting

Ciphertext has to be decoded in blocks from a function before it can be used

Q1 2023

Control flow obfuscation

progressively applied to increase the complexity of retrieving the ciphertext

Q3 2023

???

????

Evolutionary Journey Of Guloader's Configuration Tactics

The challenger!

Sample 2 Q3 2023

1. Ciphertext splitting
2. C
3. N
4. C

Basic properties

MD5	398168319933805c70238c679be79bdb
SHA-1	8e42c9f4283d9544b63b77fe886abcffea2efe14
SHA-256	617edc0de1e77f1822ef3e93986f705758ba7ee38b59725d01cf6cabd5d98ef1
Vhash	035056655d5c0550d043z8003d7z47z62z3ffz
Authentihash	17ce4b1b7cd05632b82dc6817a5aca5f01a1e89d3747b47bc59b065a682f5e88
Imphash	b78ecf47c0a3e24a6f4af114e2d1f5de
Rich PE header hash	309bdb4314a285296bb545a5382f12fc
SSDEEP	6144:jBe4i9+vu/x+TjVA0ulfeDcGyguqfWrUU4aeZk7v1jEoTEvFGCNE:7i9nuxuI2DhuGsf8W7Nj/EvFlu
TLSH	T10B6401551587947FDC416B70AA3FD265E6FFBE40BD12B2EB13180A63883C4FE9A062C5
File type	Win32 EXE executable windows win32 pe peexe
Magic	PE32 executable (GUI) Intel 80386, for MS Windows, Nullsoft Installer self-extracting archive
TrID	Win32 Executable MS Visual C++ (generic) (47.3%) | Win64 Executable (generic) (15.9%) | Win32 Dynamic Link Library (generic) (9.9%) | Win16 NE executable (generic) (7.6%) | Win32 Executable (generic) (6.8%)
DetectItEasy	PE32 | Installer: Nullsoft Scriptable Install System (3.01) [lzma] | Compiler: Microsoft Visual C/C++ (12.20.9044) [C] | Linker: Microsoft Linker (6.0) | Tool: Visual Studio | Overlay: NSIS data
File size	323.68 KB (331448 bytes)

History

Creation Time	2010-12-11 21:50:38 UTC
First Submission	2023-08-02 12:51:19 UTC
Last Submission	2023-08-04 17:02:00 UTC
Last Analysis	2023-08-29 00:12:09 UTC

Evolutionary Journey Of Guloader's Configuration Tactics

Sample 2 Q3 2023

1. Ciphertext splitting
2. 0xCC bytes triggering **EXCEPTION_BREAKPOINT**
3. Instructions triggering **EXCEPTION_ACCESS_VIOLATION**
4. Instructions triggering **EXCEPTION_SINGLE_STEP**
5. ....?

Evolutionary Journey Of Guloader's Configuration Tactics

Sample 2 Q3 2023

- **TWO** additional exceptions added!

```
if ( var_exception_code != EXCEPTION_ACCESS_VIOLATION )
{
  if ( var_exception_code != EXCEPTION_ILLEGAL_INSTRUCTION// NEW!
    && var_exception_code != EXCEPTION_PRIV_INSTRUCTION// NEW!
    && var_exception_code != EXCEPTION_SINGLE_STEP
    && var_exception_code != EXCEPTION_BREAKPOINT )
  {
    return EXCEPTION_CONTINUE_SEARCH;
  }
```

Evolutionary Journey Of Guloader's Configuration Tactics

Sample 2 Q3 2023

1. Ciphertext splitting
2. 0xCC bytes triggering **EXCEPTION_BREAKPOINT**
3. Instructions triggering **EXCEPTION_ACCESS_VIOLATION**
4. Instructions triggering **EXCEPTION_SINGLE_STEP**
5. Instructions triggering **EXCEPTION_ILLEGAL_INSTRUCTION**
6. Instructions triggering **EXCEPTION_PRIV_INSTRCTION**

Evolutionary Journey Of Guloader's Configuration Tactics

Sample 2 Q3 2023

The Solution!

1. Using memory dumps from sandbox execution
2. Locate function containing splitted cipher text using yara
3. Locate the single byte key used in the exception handler via yara
4. Using Unicorn CPU emulator framework
5. Emulate the function containing the DWORD
6. ✗ Handle the 3 types of exceptions

Evolutionary Journey Of Guloader's Configuration Tactics

The challenger!

1. Ciphertext split
2. 0xCC (INT 3) ins
3. Null pointer acc
4. CPU trap flag in

```
:04526FED guloader_veh_key_4526FED:
:04526FED      mov     edx, [eax+0B8h]
:04526FF3      cmp     [edx], cl
:04526FF5      jnz     loc_45270D3
:04526FFB      test    ax, dx
:04526FFE      mov     ecx, 0FDB0184Ah
:04527003      xor     ecx, 8B8D9523h
:04527009      test    cl, bl
:0452700B      xor     ecx, 0E20CC336h
:04527011      cmp     dx, cx
:04527014      cmp     edx, ecx
:04527016      add     ecx, 6BCEB226h
:0452701C      test    dh, ah
:0452701E      mov     dl, [edx+1]
:04527021      test    ecx, ecx
:04527023      xor     dl, cl
:04527025      movzx   edx, dl
:04527028      cmp     edx, ebx
:0452702A      mov     ecx, [eax+0B8h]
:04527030      add     ecx, edx
:04527032      dec     ecx
:04527033
:04527033 loc_4527033:
:04527033      test    dl, bl
:04527035      push    ebx
:04527036      cmp     cl, dl
:04527038      mov     ebx, [eax+0B8h]
```

```
00000000 CONTEXT struc ; (sizeof=0x2CC,
00000000
00000000 ContextFlags dd ?
00000004 Dr0  dd ?
00000008 Dr1  dd ?
0000000C Dr2  dd ?
00000010 Dr3  dd ?
00000014 Dr6  dd ?
00000018 Dr7  dd ?
0000001C FloatSave FLOATING_SAVE_AREA ?
0000008C SegGs dd ?
00000090 SegFs dd ?
00000094 SegEs dd ?
00000098 SegDs dd ?
0000009C _Edi dd ?
000000A0 _Esi dd ?
000000A4 _Ebx dd ?
000000A8 _Edx dd ?
000000AC _Ecx dd ?
000000B0 _Eax dd ?
000000B4 _Ebp dd ?
000000B8 _Eip dd ?
000000BC SegCs dd ?
000000C0 EFlags dd ?
000000C4 _Esp dd ?
```

Evolutionary Journey Of Guloader's Configuration Tactics

Sample 2 Q3 2023

1. Important offset values are hidden
2. Encoded using arithmetic operations
3. Existing yara rules wont work

```
mov     edx, 605B3469h
xor     edx, 0EABCDDC8h
cmp     ah, ch
add     edx, 594CE36Eh
cmp     al, cl
xor     edx, 0E434CDB7h ; 0x000000B8
cmp     ah, dh
test    cl, cl
push    ecx
mov     ecx, 9Ah
cmp     ecx, 1AAE2F41h
jg      loc_452C36C
pop     ecx
add     eax, edx        ; CONTEXT->EIP
```

Evolutionary Journey Of Guloader's Configuration Tactics

Sample 2 Q3 2023

The Solution!

1. Using memory dumps from sandbox execution
2. Locate function containing splitted cipher text using yara
3. ✗ Locate the single byte key used in the exception handler via yara
4. Using Unicorn CPU emulator framework
5. Emulate the function containing the DWORD
6. ✗ Handle the 3 types of exceptions

Summary

- Malware authors continues to evolve their techniques to hide configurations!

- Combining yara, emulation and scripting to automate malware configurations extractions

More interesting details in our paper!

Family	Protection
Guloader	Ciphertext splitting and control flow obfuscation
IcedID	XOR cipher
Trickbot	Erasing configuration after use, mixing decoys with actual C2 IP addresses and XOR cipher
Emotet – v5 in mid-2020	Plaintext C2 list and XOR cipher
Emotet – v6 in late-2021	O-LLVM protected and emulation is required
RedLine	Base64 and XOR cipher
WarZone RAT	RC4 and a variant with customized RC4+

Table 1: Malware families and their key protections.

Thank you

paloalto NETWORKS