

MWS Cup 2017

課題2: 静的解析

中津留 勇
石淵 一三
石丸 傑



課題2 担当



中津留 勇

[ynakatsuru\[at\]secureworks.com](mailto:ynakatsuru[at]secureworks.com)

SecureWorks Japan

株式会社



石淵 一三

[kazumi.ishibuchi.hh\[at\]hitachi.com](mailto:kazumi.ishibuchi.hh[at]hitachi.com)

日立製作所



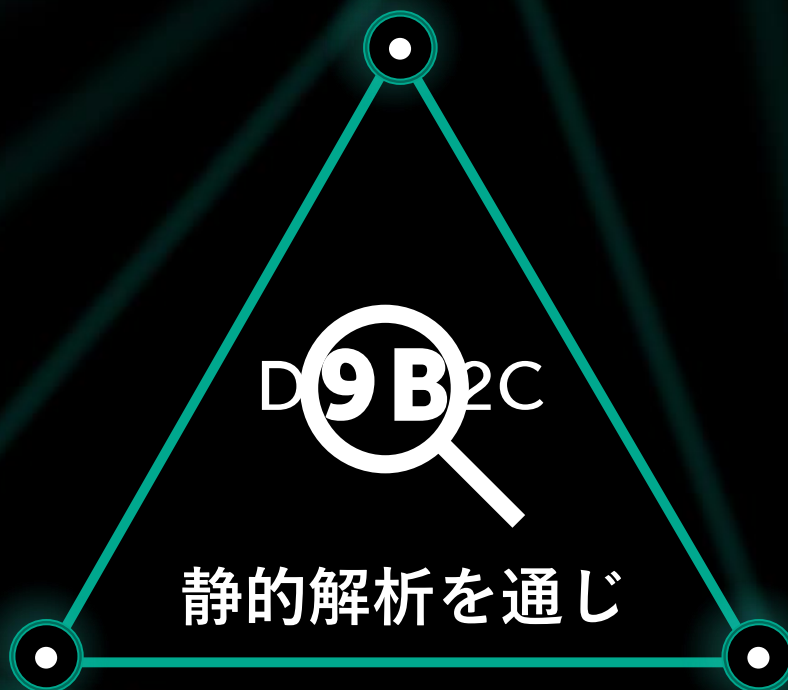
石丸 傑

[suguru.Ishimaru\[at\]kaspersky.com](mailto:suguru.Ishimaru[at]kaspersky.com)

株式会社 カスペルスキー

課題2 変わらぬテーマ

マルウェアを正しく理解する



最新情報を得る

実務に近い作業

MWS Cup 2017

課題2 解説

課題2 oni.exe



製品 サービス パートナー イベント リソース 会社情報



◀ Back to Blog

Don't Miss Another Blog

Email Address

Submit

日本をターゲットにした GlobeImposterの亜種 ("ONI"の正体)

https://www.cylance.com/ja_jp/blog/jp-oni-ransomware-globeimposter.html

KASPERSKY

課題2 設問



設問1

呼び出し規約

当該検体の (Windows API や ライブラリ関数以外の) 関数呼び出しに使用されている呼出規約を以下から選べ (4点)



設問2

関数の挙動解析

関数 `sub_401000` が何をする関数が答えよ (4点)



設問3

被害対象の特定

暗号化の対象となるドライブの種類をすべて選択せよ (4点)



設問4

関数の挙動解析

関数 `sub_406306` が何をする関数が答えよ (4点)



設問5

引数の意味

当該検体が SHA-256 ハッシュ値を求めるために使用される関数 `sub_404C3E` は、4番目の引数によってどのような処理の変化が起きるか答えよ (4点)



設問6

格納データの意味

当該検体のリソース EXDATA に格納されている112番のデータは、復号しても意味不明なデータとなるが、本来どのような情報が格納されているべきか、コードから推測せよ (5点)

課題2 解析のポイント

難しく考えない



アセンブラ言語
そのものは単純

コード全てを
読む必要はない



わかりやすい
文字を探す



- 文字列
- 定数
- API

必要な部分を
必要な分だけ
集中して読む

困った時の
Web検索



Google先生や
MSDNは貴重
な情報元

設問1 呼び出し規約

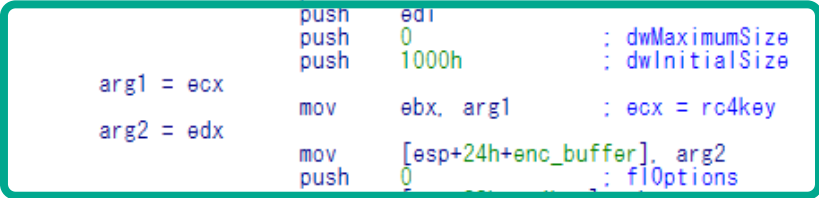


当該検体の (Windows APIやライブラリ関数以外の) 関数呼び出しに使用されている呼出規約を以下から選べ(4点)

- `__cdecl` 引数をスタックに経由で渡す
- `__stdcall` 引数をスタック経由で渡す
- `__fastcall` 引数をECX, EDX,スタックを經由して渡す
- `__thiscall` 引数をスタックに、thisポインターをECXに格納
- `others` 他に適切な答えがある為却下

```
text:00406306          : int __cdecl arg3(DWORD dwBytes)
text:00406306          arg3               proc near
text:00406306          ; CODE XREF: sub_4059AB+311p
text:00406306          ; sub_405A64+8F1p
text:00406306          ; sub_405BAD+5181p
text:00406306          ; sub_405BAD+5531p
text:00406306          enc_buffer         = dword ptr -0Ch
text:00406306          var_8             = dword ptr -8
text:00406306          rc4key            = dword ptr -4
text:00406306          dwBytes           = dword ptr 4
text:00406306          ; arg3
text:00406306          000 83 EC 0C      sub     esp, 0Ch
text:00406309          00C 53           push   ebx
text:0040630A          010 55           push   ebp
text:0040630B          014 56           push   esi
text:0040630C          018 57           push   edi
text:0040630D          01C 6A 00        push   0
text:0040630E          020 68 00 10 00 00 push   1000h
text:0040630F          024 8B D9        mov     ebx, arg1
text:00406310          024 89 54 24 18  mov     [esp+24h+enc_buffer], arg2
text:00406311          028 6A 00        push   0
text:00406312          028 89 5C 24 24  mov     [esp+28h+rc4key], ebx
text:00406313          028 FF 15 B0 10 41 00 call   ds:HeapCreate
text:00406314          01C 8B 6C 24 20  mov     ebp, [esp+1Ch+dwBytes]
text:00406315          028 8B 6C 24 20  mov     ebp, [ebp+1Ch+dwBytes]
text:00406316          028 8B 6C 24 20  mov     ebp, [ebp+1Ch+dwBytes]
text:00406317          028 8B 6C 24 20  mov     ebp, [ebp+1Ch+dwBytes]
```

スタックだけでなくECX, EDX経由で値を渡していることがわかる



設問2 関数の挙動解析

関数 sub_401000 が何をやる関数が答えよ (4点)



全プロセス中からsql、outlook、ssms、postgreに一致したプロセス名のPIDを基にtaskkillを用いてプロセスツリーごと強制終了する関数

関数のアドレスを取得

- Process32FirstW
- Process32NextW
- CreateToolhelp32Snapshot

CreateToolhelp32Snapshotでスナップショットを作成
Process32FirstWでスナップショット内の最初のプロセス情報を取得

<pre> text:0040101F 8B F8 text:00401021 68 78 4E 41 00 text:00401026 57 text:00401027 FF D6 text:00401029 68 88 4E 41 00 text:0040102E 57 text:0040102F A3 44 24 42 00 text:00401034 FF D6 text:00401036 68 98 4E 41 00 text:00401038 57 text:0040103C A3 4C 24 42 00 text:00401041 FF D6 text:00401043 6A 44 text:00401045 5F text:00401046 8B F0 text:00401048 33 DB text:0040104A 57 text:00401048 8D 44 24 20 text:0040104F 89 35 48 24 42 00 text:00401055 53 text:00401056 50 text:00401057 E8 64 76 00 00 text:0040105E 83 04 00 text:00401061 89 7C 24 28 text:00401063 53 text:00401064 6A 02 text:00401066 FF D6 text:00401068 53 text:00401068 8B F8 text:00401071 33 4C 24 14 text:00401072 63 F8 FF text:00401071 2 84 C1 01 00 00 text:0040107E 84 24 58 04 00 00 text:0040107E 07 84 24 58 04 00 00 24+ text:00401089 50 text:0040108A 53 text:00401088 FF 15 44 24 42 00 text:00401091 text:00401091 text:00401091 BD 00 82 41 00 text:00401091 80 83 41 00 </pre>	<pre> mov _Kernel32_dll, eax push offset ProcName, "Process32FirstW" push _Kernel32_dll ; hModule call esi ; GetProcAddress push offset aProcess32nextw, "Process32NextW" push _Kernel32_dll ; hModule mov _Process32FirstW, eax call esi ; GetProcAddress push offset aCreatetoolhelp, "CreateToolhelp32Snapshot" push _Kernel32_dll ; hModule mov _Process32NextW, eax call esi ; GetProcAddress push 44h pop edi _CreateToolhelp32Snapshot = esi mov _CreateToolhelp32Snapshot, eax xor ebx, ebx push edi ; 44h lea eax, [esp+2484h+StartupInfo.dwX] mov dword_422448, _CreateToolhelp32Snapshot push ebx ; 0 push eax call sub_4086C0 add esp, 0Ch mov [esp+2480h+StartupInfo.dwX], edi push ebx push eax call _CreateToolhelp32Snapshot mov hSnapshot, eax mov [esp+2480h+var_2478+4h], hSnapshot cmp hSnapshot, 0FFFFFFFFh jz loc_401238 lea eax, [esp+2488h+var_2030] mov [esp+2488h+var_2030], eax push eax push hSnapshot call _Process32FirstW loc_401091: mov ebp, offset procNameStrList, 0 sql mov ebp, offset procNameStrList, 0 sql </pre>
--	--

設問2 関数の挙動解析

関数 sub_401000 が何をする関数か答えよ (4点)



全プロセス中からsql、outlook、ssms、postgreに一致したプロセス名のPIDを基にtaskkillを用いてプロセスツリーごと強制終了する関数

対象プロセス名と思われる文字列

- sql
- outlook
- ssms
- postgre

Taskkill /F /PID をコマンドとして実行

```
loc_401091:      mov     ebp, offset procNameStrList ; 0 "sql"
                ; 4 "outlook"
                ; 8 "ssms"
                ; c "postgre"
```

```
loc_401096:      lea    ecx, [esp+2490h+WideCharStr] ; lpWideCharStr
```

```
and     ecx, 3
lea     eax, [esp+2498h+CommandLine] ; "taskkill /F /T /PID "
rep movsb
xor     ecx, ecx
push   ecx      ; lpCurrentDirectory
push   ecx      ; lpEnvironment
push   8000000h ; dwCreationFlags
push   ecx      ; bInheritHandles
push   ecx      ; lpThreadAttributes
push   ecx      ; lpProcessAttributes
push   eax      ; lpCommandLine
push   ecx      ; lpApplicationName
call   ds:CreateProcessA
```

```
loc_401216:      hSnapshot = ebx
mov     hSnapshot, [esp+struc_1.hSnapshots]
lea     eax, [esp+2490h+var_2038]
push   eax
push   hSnapshot
call   _Process32NextW
test   eax, eax
jnz    loc_401091
push   hSnapshot ; hObject
call   ds:CloseHandle
```


設問3 被害対象の特定

暗号化の対象となるドライブの種類をすべて選択せよ (4点)



- 固定ドライブ
- 取り外し可能ドライブ
- ネットワークドライブ
- ~~書き込み可能なCD/DVDドライブ~~

GetDriveTypeAを
Importsから探す

Address	Ordinal	Name	Library
00411104		GetCurrentThreadId	KERNEL32
00411058		GetDriveTypeA	KERNEL32
00411120		GetEnvironmentStringsA	KERNEL32

戻り値と比較してい
る値の意味を調べる

```
text:004058B8  
text:004058B8 F6 03 01  
text:004058BB 74 46  
text:004058BD 8D 85 F8 EF FF FF  
text:004058C3 50  
text:004058C4 FF 15 58 10 41 00  
text:004058CA 83 F8 03  
text:004058CD 74 0A  
text:004058CF 83 F8 02  
text:004058D2 74 05  
text:004058D4 83 F8 04  
text:004058D7 75 2A  
text:004058D9
```

```
loc_4058B8: ; CODE XREF: sub_405865+A6↓j  
test bl, 1  
jz short loc_405903  
lea eax, [ebp+String2]  
push eax ; lpRootPathName  
call ds:GetDriveTypeA  
cmp eax, DRIVE_FIXED  
jz short loc_4058D9  
cmp eax, DRIVE_REMOVABLE  
jz short loc_4058D9  
cmp eax, DRIVE_REMOTE  
jnz short loc_405903
```

設問4 関数の挙動解析

関数 sub_406306 が何をする関数か答えよ (4点)



RC4で復号化する関数。第一引数に暗号データへのアドレス、第二引数に鍵、第三引数にデータサイズがそれぞれ入り、戻り値は復号データへのアドレスが入る

00 から ff まで 256 の
バイトのsboxを初期化

第一引数である鍵を
用いてsboxを生成

```
text:00406346
text:00406346 01C 33 C0
text:00406348
text:00406348 01C 88 80 20 E4 41 00
text:00406348 01C 40
text:0040634E 01C 3D 00 01 00 00
text:00406354 01C 75 F2
text:00406356 01C 33 FF
text:00406358
text:00406358 01C 33 ED
text:0040635A
text:0040635A
text:0040635A 01C 33 D2
text:0040635C 01C 8A 9D 20 E4 41 00
text:00406362 01C 8B 05
text:00406364 01C 0F B6 CB
text:00406367 01C F7 F6
text:00406369 01C 8B 44 24 18
text:0040636D 01C 0F B6 04 02
text:00406371 01C 03 F8
text:00406373 01C 03 F9
text:00406375 01C 81 E7 FF 00 00 00
text:0040637B 01C 8A 87 20 E4 41 00
text:00406381 01C 88 85 20 E4 41 00
text:00406387 01C 45
text:00406388 01C 88 9F 20 E4 41 00
text:0040638E 01C 81 FD 00 01 00 00
text:00406394 01C 75 C4
text:00406396 01C 8B 6C 24 20
```

```
cnt = eax
xor cnt, cnt ; cnt = 0

init_sbox:
; CODE XREF: rc4+4E1j
mov sbox_buffer[cnt], al ; sbox_buffer
; = [00 01 02 ... fd fe ff]

inc cnt
cmp cnt, 256 ; compare cnt = 256
jnz short init_sbox ; sbox_buffer
```

```
cnt2 = ebp
xor cnt2, cnt2 ; cnt2 = 0

gen_sbox_using_rc4key:
; CODE XREF: rc4+8E1j
xor edx, edx
mov bl, ss:sbox_buffer[cnt2]
mov eax, cnt2
movzx ecx, bl
div esi
mov eax, [esp+10h+rc4key] ; ecx (arg_0)
movzx eax, byte ptr [edx+eax]
add edi, eax
add edi, ecx
and edi, 0FFh
mov al, sbox_buffer[edi]
mov ss:sbox_buffer[cnt2], al
inc cnt2
mov sbox_buffer[edi], bl
cmp cnt2, 256 ; compare cnt2 = 256
jnz short gen_sbox_using_rc4key
```


設問4 関数の挙動解析

関数 sub_406306 が何をする関数か答えよ (4点)



RC4で復号化する関数。第一引数に暗号データへのアドレス、第二引数に鍵、第三引数にデータサイズがそれぞれ入り、戻り値は復号データへのアドレスが入る

```
text:004063B2 01C 29 7C 24 10      sub     [esp+1Ch+enc_buffer], edi ; enc_data = [esp+1ch] + edi
text:004063B6                          dec_buffer = ebx
text:004063B6 01C 8B DF      mov     dec_buffer, edi
text:004063B8 01C 8B FD      mov     edi, ebp
text:004063BA 01C 8B 6C 24 10     mov     ebp, [esp+1Ch+enc_buffer]
text:004063BE                          rc4_loop:
text:004063BE                          ; CODE XREF: rc4+10B↓j
text:004063BE 01C 42      inc     edx
text:004063BF 01C 81 E2 FF 00 00 00   and     edx, 0FFh
text:004063C5 01C 8A 8A 20 E4 41 00   mov     cl, sbox_buffer[edx]
text:004063C8 01C 0F B6 C1      movzx   eax, cl
text:004063CE 01C 03 F0      add     esi, eax
text:004063D0 01C 81 E6 FF 00 00 00   and     esi, 0FFh
text:004063D6 01C 8A 86 20 E4 41 00   mov     al, sbox_buffer[esi]
text:004063DC 01C 88 8E 20 E4 41 00   mov     sbox_buffer[esi], cl
text:004063E2 01C 88 82 20 E4 41 00   mov     sbox_buffer[edx], al
text:004063E8 01C 0F B6 8E 20 E4 41 00   movzx   ecx, sbox_buffer[esi]
text:004063EF 01C 0F B6 C0      movzx   eax, al
text:004063F2 01C 03 C8      add     ecx, eax
text:004063F4 01C 81 F1 FF 00 00 80   and     ecx, 800000FFh
text:004063FA 01C 79 08      jns     short loc_406404
text:004063FC 01C 49      dec     ecx
text:004063FD 01C 81 C9 00 FF FF FF   or      ecx, 0FFFFFF00h
text:00406403 01C 41      inc     ecx
text:00406404                          loc_406404:
text:00406404                          ; CODE XREF: rc4+F4↑j
text:00406404 01C 8A 81 20 E4 41 00   mov     al, sbox_buffer[ecx]
text:0040640A 01C 32 04 2B      xor     al, [dec_buffer+ebp] ; enc_data
text:0040640D 01C 88 03      mov     [dec_buffer], al
text:0040640F 01C 43      inc     dec_buffer
text:00406410 01C 4F      dec     edi
text:00406411 01C 75 AB      jnz     short rc4_loop
text:00406413 01C 8B 7C 24 14     mov     edi, [esp+1Ch+var 8]
text:00406417 01C 8B 3C 57 17     mov     edi, [ebp+1Ch+var 8]
text:00406419 01C 32 9B      lss     esp, [ebp+1Ch+var 8]
text:0040641D 01C 4E      qec    edi
text:0040641E 01C 4E      qec    edi
text:00406420 01C 8B 03      mov     eax, [dec_buffer]
text:00406423 01C 8B 03      mov     eax, [dec_buffer]
```

1バイトづつXOR
ストリーム暗号



設問5 引数の意味



当該検体が SHA-256 ハッシュ値を求めるために使用される関数 sub_404C3Eは、4番目の引数によってどのような処理の変化が起きるか答えよ (4点)

引数が0の場合はSHA-256のハッシュ値、それ以外の場合にはSHA-224のハッシュ値を求めるように変化する。

4番目の引数はsub_403FFDの2番目の引数として分岐フラグとして用いられる

SHA-256かSHA-224の定数が使用される

```
set_constants_sha256_224 proc near ; CODE XREF: sub_4039A5+7F1p
text:00403FFD .text:00403D70Tj
text:00403FFD .text:00403E23Tj
text:00403FFD sub_404C3E+251p
text:00403FFD sub_4064A9+671p
text:00403FFD 000 93 21 00
text:00403FFD 000 22 31 04
text:00403FFD 000 81 02 80
text:00404004
text:00404006 000 06 00 00 00
text:00404008 000 07 41 08 67 E6 09 6A
text:0040400F 000 07 41 00 85 AE 67 BB
text:00404016 000 07 41 10 72 F3 6E 3C
text:0040401D 000 07 41 14 3A F5 4F A5
text:00404024 000 07 41 18 7F 52 0E 51
text:0040402B 000 07 41 1C 8C 68 05 9B
text:00404032 000 07 41 20 AB D9 83 1F
text:00404039 000 07 41 24 19 CD E0 5B
text:00404040 000 EB 38
text:00404042
text:00404042
text:00404042
text:00404042
text:00404042 000 07 41 08 D8 9E 05 C1
text:00404042 000 07 41 0C 07 D5 7C 36
text:00404050 000 07 41 10 17 0D 70 30
text:00404058 000 07 41 14 39 59 0E F7
text:0040405E 000 07 41 18 31 0B C0 FF
text:00404065 000 07 41 1C 11 15 58 68
text:0040406C 000 07 41 20 A7 8F F9 64
text:00404073 000 07 41 24 A4 4F FA BE
text:0040407A
text:0040407A
text:0040407A 000 89 51 68
text:0040407D 000 C3
loc_404042: ; CODE XREF: set_constants_sha256_224+91Tj
mov [ecx+buffer_1.constants_0], 0C1059ED8h
mov [ecx+buffer_1.constants_4], 367CD507h
mov [ecx+buffer_1.constants_8], 3070DD17h
mov [ecx+buffer_1.constants_c], 0F70E5939h
mov [ecx+buffer_1.constants_10], 0FFC00B31h
mov [ecx+buffer_1.constants_14], 68581511h
mov [ecx+buffer_1.constants_18], 64F98FA7h
mov [ecx+buffer_1.constants_1c], 0BEFA4FA4h
loc_40407A: ; CODE XREF: set_constants_sha256_224+431Tj
mov [ecx+buffer_1.flag_sha256_224], edx ; arg_c of 0x404c3e
retn
```


設問5 引数の意味



当該検体が SHA-256 ハッシュ値を求めるために使用される関数 sub_404C3Eは、4番目の引数によってどのような処理の変化が起きるか答えよ (4点)

引数が0の場合はSHA-256のハッシュ値、それ以外の場合にはSHA-224のハッシュ値を求めるように変化する。

計算部分でも4番目の引数に基づきSHA-256とSHA-224で処理が分岐

```
text:00404C00 010 8A 47 22 mov     al, byte ptr [edi+(buffer_1.constants_18+2)]
text:00404C03 010 88 43 19 mov     [ebx+19h], al
text:00404C06 010 8B 47 20 mov     eax, [edi+buffer_1.constants_18]
text:00404C09 010 C1 E8 08 shr     eax, 8
text:00404C0C 010 88 43 1A mov     [ebx+1Ah], al
text:00404C0F 010 8A 47 20 mov     al, byte ptr [edi+buffer_1.constants_18]
text:00404C12 010 88 43 1B mov     [ebx+1Bh], al
text:00404C15 010 77 58 00 cmp     [edi+buffer_1.flag_sha256_224], 0 ; arg_c of 0x404c3e
text:00404C18 010 58 43 1C pop     ecx
text:00404C1A 018 75 1E jnz     short loc_404C37
text:00404C1D 018 58 43 1C mov     al, byte ptr [edi+(buffer_1.constants_1c+3)]
text:00404C1F 018 88 43 1C mov     [ebx+1Ch], al
text:00404C22 018 88 43 1C mov     al, byte ptr [edi+(buffer_1.constants_1c+2)]
text:00404C25 018 88 43 1C mov     [ebx+1Dh], al
text:00404C28 018 8B 47 24 mov     eax, [edi+buffer_1.constants_1c]
text:00404C2B 018 C1 E8 08 shr     eax, 8
text:00404C2E 018 88 43 1E mov     [ebx+1Eh], al
text:00404C31 018 8A 47 24 mov     al, byte ptr [edi+buffer_1.constants_1c]
text:00404C34 018 88 43 1F mov     [ebx+1Fh], al
text:00404C37
text:00404C37 018 5F loc_404C37: ; CODE XREF: sub_404AD2+1481j
text:00404C37 018 EE pop     edi
text:00404C37 018 EE bbb     eq1
text:00404C37 018 EE loc_404C37: ; CODE XREF: 80404040+1481j
text:00404C37 018 88 43 1E mov     [ebx+1Eh], al
text:00404C37 018 8A 47 24 mov     al, byte ptr [edi+buffer_1.constants_1c]
```

設問5 引数の意味



当該検体が SHA-256 ハッシュ値を求めるために使用される関数 `sub_404C3E` は、4番目の引数によってどのような処理の変化が起きるか答えよ (4点)

引数が0の場合はSHA-256のハッシュ値、それ以外の場合にはSHA-224のハッシュ値を求めるように変化する。



13 4:07 PM

https://tls.mbed.org/api/sha256_8h_source.html

 tls.mbed.org

[sha256.h Source File - API Documentation - mbed TLS \(Previously PolarSSL\)](#)

Doxygen API documentation for sha256.h Source File - API Documentation - mbed TLS (previously PolarSSL)

この127, 128行目

<https://tls.mbed.org/sha-256-source-code>

実装はこちら

 tls.mbed.org

[SHA-256 Source Code \(SHA2\) - mbed TLS \(Previously PolarSSL\)](#)

The source code for the SHA-256 algorithm, is part of the mbed TLS library and represents the most current ve

represents the most current ve

The source code for the SHA-2

SHA-256 source code (SHA2)



you0708 2:37 PM

昨日公開した https://github.com/you0708/ida/tree/master/idadpython_tools/findcrypt 使えば ONI の sha256/sha224 すぐ分かるというオチ

 GitHub

[you0708/ida](#)

ida - IDA related stuff



設問6 格納データの意味



当該検体のリソース EXDATA に格納されている112番のデータは、復号しても意味不明なデータとなるが、本来どのような情報が格納されているべきか、コードから推測せよ(5点)

拡張子リストが“,”(カンマ)を区切りとして格納されているべきと考えられる。また、最初の区切り文字を“ ”(スペース)にも変更することが可能であり、当該拡張子を持つファイルはファイルサイズに関わらずデータ全体を暗号化対象とする

EXDATAからデータを
取得、RC4で復号化

```
text:004059AE 00C 57
text:004059AF 010 8B F2
text:004059B1 010 8B 08 57 41 00
text:004059B6 010 53
text:004059B7 014 0F B7 06
text:004059BA 014 33 ED
text:004059BC 014 50
text:004059BD 018 55
text:004059BE 01C 8B F9
text:004059C0 01C FF 15 40 10 41 00
text:004059C6 010 50
text:004059C7 014 55
text:004059C8 018 15 08 18 41 00
text:004059C9 010 50
text:004059CA 014 8B D3
text:004059D1 014 8B CE
text:004059D3 014 E8 67 FF FF FF
text:004059D8 014 8B D0
text:004059DA 014 8B CF
text:004059DC 014 E8 25 09 00 00
text:004059E1 014 59
text:004059E2 010 8B D8
```

```
push edi
mov esi, edx
mov ebx, offset Type ; ~EXDATA"
push ebx ; lpType
movzx eax, si
xor ebp, ebp
push eax ; lpName
push ebp ; hModule
mov edi, ecx
call ds:FindResourceW
push eax ; hResInfo
push ebp ; hModule
call ds:SizeofResource
push eax ; dwBytes
mov edx, ebx
mov ecx, esi
call get_resource
mov edx, eax
mov ecx, edi
call rc4_dec
pop ecx
mov ebx, eax
mov ebp, esp
bob ecx
call Log_qec
mov ecx, edi
```


設問6 格納データの意味



当該検体のリソース EXDATA に格納されている112番のデータは、復号しても意味不明なデータとなるが、本来どのような情報が格納されているべきか、コードから推測せよ(5点)

拡張子リストが“,”(カンマ)を区切りとして格納されているべきと考えられる。また、最初の区切り文字を“ ”(スペース)にも変更することが可能であり、当該拡張子を持つファイルはファイルサイズに関わらずデータ全体を暗号化対象とする

復号化されたEXDATA112
と暗号化対象となるファイルの拡張子と比較

```
text:0040576E
text:0040576E 4284 8D 44 24 54          loc_40576E:  lea  eax, [esp+4284h+FindFileData_cFileName+4]
text:00405772 4284 50                    push  eax
text:00405773 4288 FF 1 40 24 42 00      call  _PathFindExtensionW
text:00405779 4288 8B 0 mov  ecx, eax           : lpWideCharStr
text:0040577B 4288 E8 3 FC FF FF      call  sub_4053BD       : WideCharToMultiByte
text:00405780 4288 8B 3 C8 8F 41 00    mov  edi, dword_418F08
text:00405786 4288 33 F xor  esi, esi
text:00405788 4288 8B 2 44 F5 41 00    mov  ebp, dec_exdata112 : 0 ext1
text:00405788 4288 8B 2 44 F5 41 00    mov  ebp, 4 ext2
text:00405788 4288 8B 2 44 F5 41 00    mov  ebp, 8 ext3
text:00405788 4288 8B 2 44 F5 41 00    mov  ebp, 0 .oni
text:0040578E 4288 8B D mov  ebx, eax
text:00405790 4288 85 2 test  edi, edi
text:00405792 4288 74 2 jz   short loc_405788
text:00405794
text:00405794 4288 6A F loc_405794:  push  0FFFFFFFh       : cchCount2
text:00405796 428C FF 7 B5 00      push  dword ptr [ebp+esi*4+0] : lpString2
text:0040579A 4290 6A F push  0FFFFFFFh       : cchCount1
text:0040579C 4294 53            push  ebx              : lpString1
text:0040579D 4298 6A 0 push  1                : dwCmpFlags
text:0040579F 429C 68 0 08 00 00      push  800h            : Locale
text:004057A4 42A0 FF 1 98 10 41 00    call  ds:CompareStringA
text:004057AA 4288 83 F 02          cmp  eax, 2
text:004057AD 4288 0F 84 4 00 00 00  jz   loc_405788
text:004057B3 4288 46            inc  esi
text:004057B4 4288 3B F7          cmp  esi, edi
text:004057B6 4288 72 DC          jb   short loc_405794
text:004057B8
```

設問6 格納データの意味



当該検体のリソース EXDATA に格納されている112番のデータは、復号しても意味不明なデータとなるが、本来どのような情報が格納されているべきか、コードから推測せよ(5点)

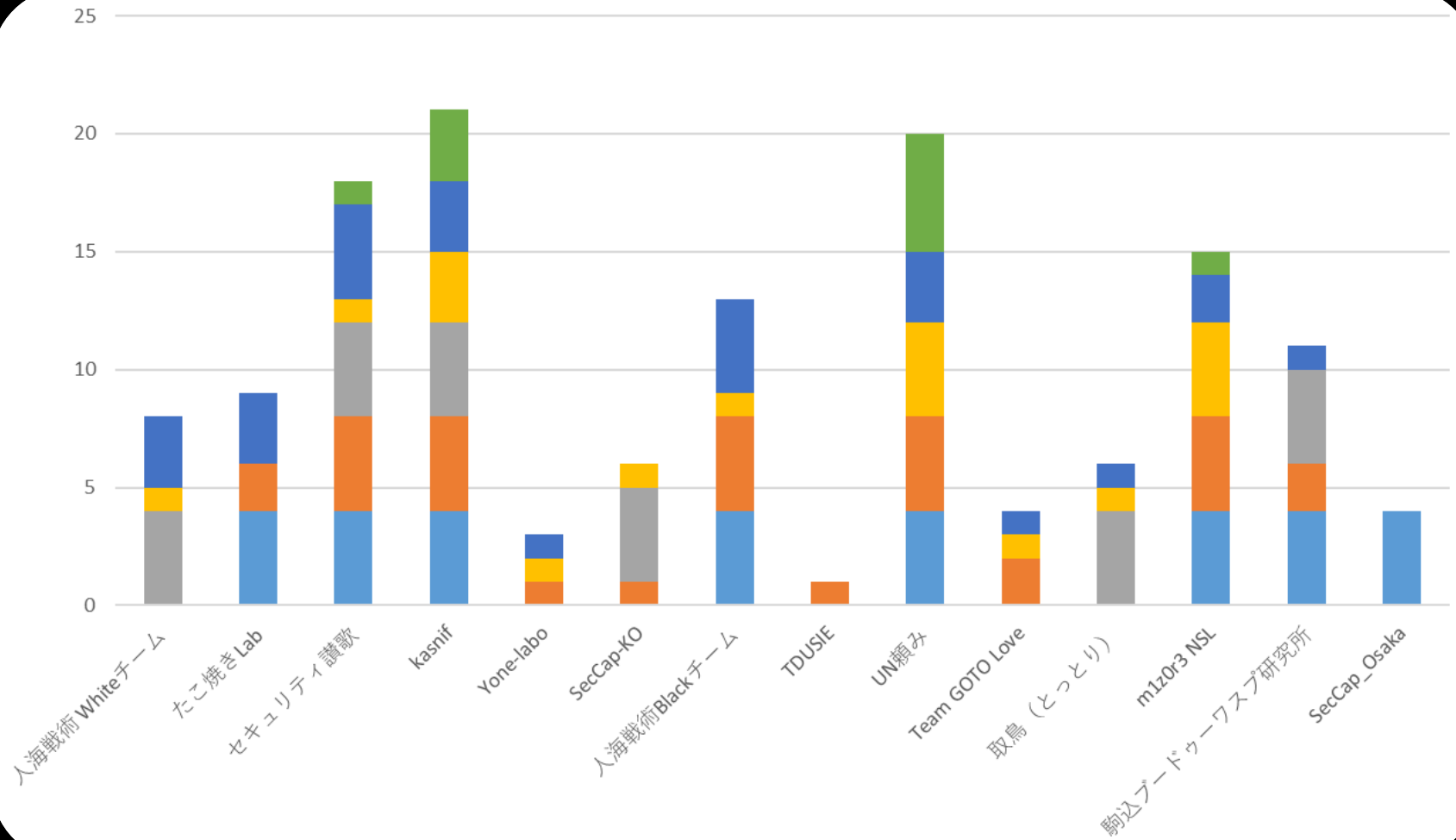
拡張子リストが“,”(カンマ)を区切りとして格納されているべきと考えられる。また、最初の区切り文字を“ ”(スペース)にも変更することが可能であり、当該拡張子を持つファイルはファイルサイズに関わらずデータ全体を暗号化対象とする

EXDATA112の拡張子であった場合ファイルサイズに関係なく全て暗号化する

暗号化対象ファイルのサイズが500000h以上かどうかを確認し、以上の場合は暗号化するサイズを500000hまでに設定する

```
00404EAF 89 5C 24 14  mov     [esp+74h+NumberOfBytesWritten], ebx
00404EB3 39 5C 24 70  cmp     [esp+74h+isFullenc], ebx
00404EB7 75 27      jnz     short loc_404EE0
00404EB9 3E FB     cmp     edi, ebx
00404EBB 7C 23     jl      short loc_404EE0
00404EBD 00 50 00  mov     eax, 500000h
00404EC1 71 0A     jg     short loc_404ECE
00404EC4 8B 35 00 8F 41 00  mov     esi, filesize
00404ECA 3B F0     cmp     esi, eax
00404EC8 76 18     jbe     short loc_404EE6
00404ECE:                                : CODE XREF: sub_404CFA+1C8↑j
00404ECE 8B F0     mov     esi, eax
00404ED0 8F FB     mov     edi, ebx
00404ED2 49 38 00 8F 41 00  mov     filesize, esi
00404ED8 89 3D 04 8F 41 00  mov     dword_418FC4, edi
00404EDE 3B 08     cmp     esi, 8
00404EE0:                                : CODE XREF: sub_404CFA+1BD↑j
00404EE0 3B 38 00 8F 41 00  mov     esi, filesize
00404EE6:                                : CODE XREF: sub_404CFA+1D2↑j
00404EE6 3B 38 00 8F 41 00  mov     esi, filesize
00404EE6:                                : sub_404CFA+1E4↑j
00404EE6 53      push    ebx
00404EE7 68 00 20 00 00  push    2000h
00404EE7 53      push    5000h
00404EE7 59      bnzp   epd
00404EE7 59      bnzp   epd
```


課題 2 採点結果





MWS Cup 2017

課題2 今後の課題

アンケートの意見コメント

今回の課題2に関して良かった点や悪かった点、意見やコメント等あればお願いします

9件の回答

特になし (2)

(すみません静的解析の担当でないのでわかりません。)

関数の動作を解答する問題で、単に関数の動作のみを解答すればよいのか、マルウェアとしてその関数がどのような悪質な動きをするのかを解答すればよいのかわからなかったです。

難易度、問題数的にもちょうど良かったように思います。

解いてないです

現実的な課題でよかったです

競技中のヒントが有用な情報だった。問題文が一部不明確であり、どの程度詳細に回答すればよいかかわからなかった。

近年流行しているランサムウェアを対象とした静的解析を行うことができ、楽しく課題に取り組むことができました。

問題作成の課題

テーマ選定の難しさ

自動化などを静的解析で適用することの困難さ

より実務に近い検体、2時間で解ける
課題として昇華させる難しさ

問題文の出し方

問題文が一部が伝わりにくかった

どの程度の解答を求めているのか
明示できていなかった

作成委員不足

協議中のプレゼンが公開処刑すぎる為
蚤の心臓では耐えられない

問題作成はコードと向き合う必要がある為勉強
になること、なにより楽しい

参加者側の課題

過去問や過去の解答をもとに復習

Write-up書いてみると勉強になる



you0708 11:23 AM

僕からのメッセージとしては

- * アンケートは各チーム最低一人は必ず回答してください。
- * どれだけ作問側がコストをかけてるか分かってますか？フィードバックは絶対にするように。
- * 特に次につながる意見だったり、継続希望とか不要とかそもそも論だったり、そういうのを求めています。
- * この結果を研究に反映させてくれることを大いに期待しています。

アンケート答えてくださいね。お願いします。

勉強する際の参考資料

セキュリティ・キャンプ全国大会2015でのマルウェア分析講義(2015-09-10)

<https://www.jpccert.or.jp/magazine/acreport-seccamp.html>

リバースエンジニアリング入門 - @IT

<http://www.atmarkit.co.jp/ait/series/2614/>

Edomae 2015 - マルウェアを解析してみよう

<http://www.slideshare.net/SatoshiMimura/edomae-2015>



WEB

Practical Malware Analysis | No Starch Press

<https://www.nostarch.com/malware>

O'Reilly Japan - アナライジング・マルウェア

<https://www.oreilly.co.jp/books/9784873114552/>



書籍

MLに投げてるので、それ入手する
データセットに入っている資料を読む



過去問

THANK YOU

ynakatsuru[at]secureworks.com
kazumi.ishibuchi.hh[at]hitachi.com
suguru.Ishimaru[at]kaspersky.com

KASPERSKY LAB



SAVING
THE WORLD
FOR 20 YEARS

Copyright© 2017 Kaspersky Lab
無断複写・転載を禁じます。カスペルスキー、Kaspersky®はKaspersky Labの商標または登録商標です。