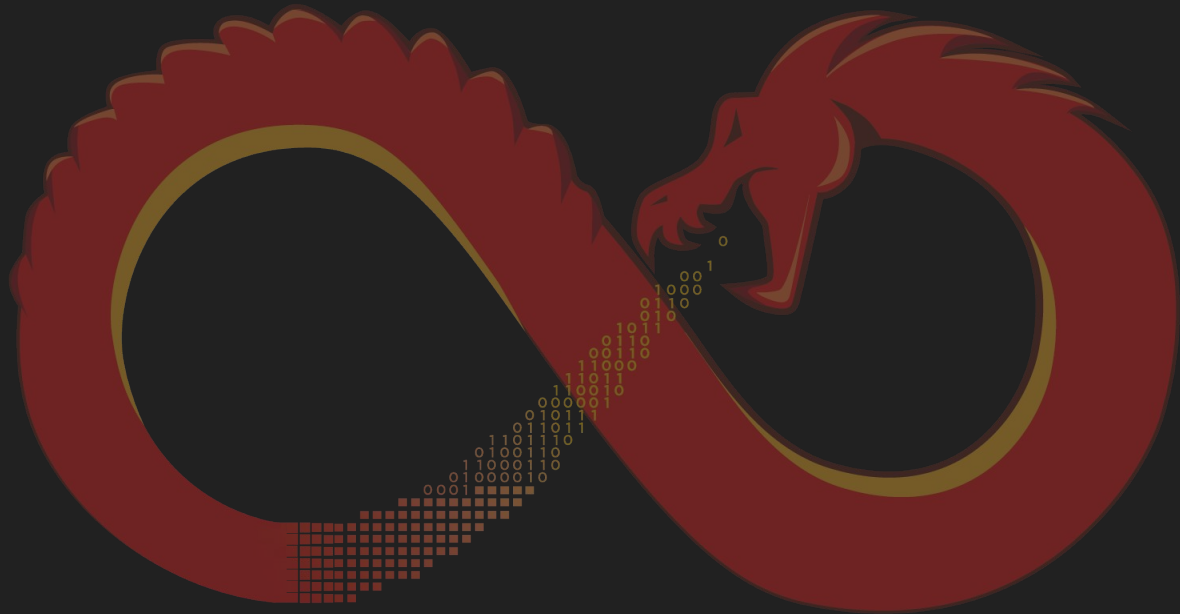


MWS Cup 2023

Static Analysis 解説

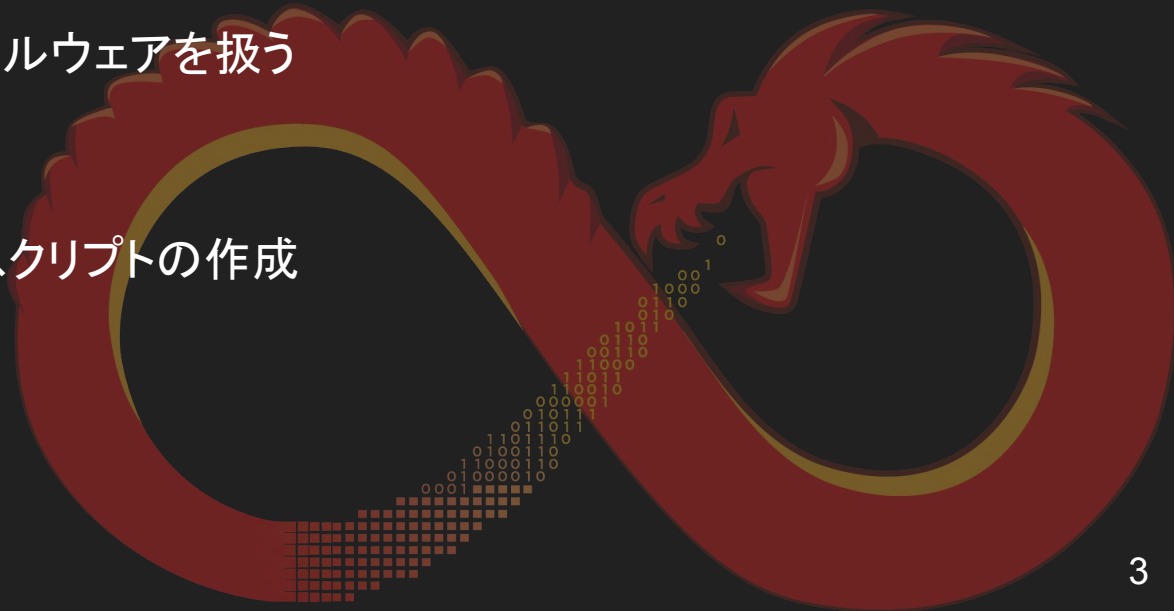


2023の問題担当

- Static Analysis主担当
 - 中島 将太 (株式会社サイバーディフェンス研究所)
- 問題作成委員
 - 石丸 傑 (伊藤忠サイバー&インテリジェンス株式会社)
 - 皆川 諒 (株式会社エヌ・エフ・ラボラトリーズ)
 - 桑原 翼 (株式会社FFRIセキュリティ)

課題2のテーマ

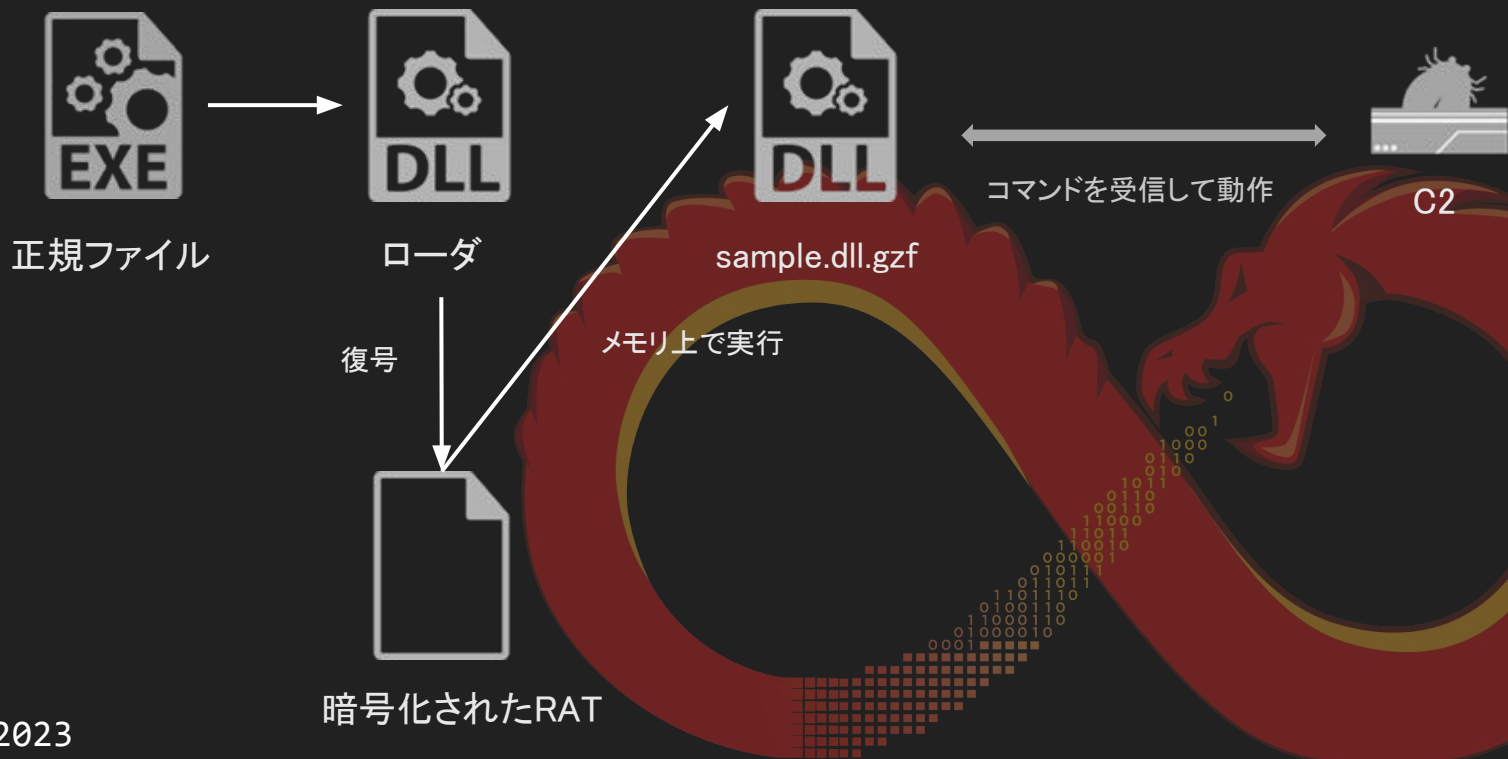
- マルウェアを正しく理解する
 - 課題を通して解析のポイントを学習する
- 最新情報を得る
 - 最近のin-the-wildなマルウェアを扱う
- 実務に近い作業
 - コマンドの分析
 - 静的解析による復号スクリプトの作成
 - コンフィグの理解



ポイント

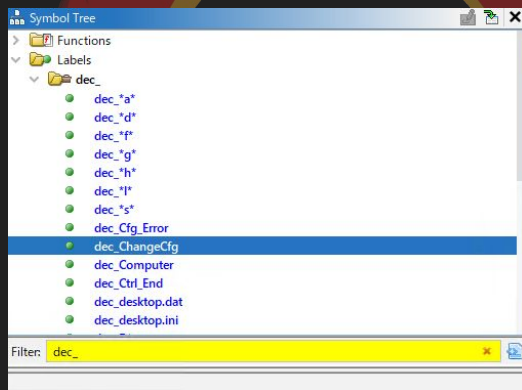
- 積極的に変数名や型を変更する
 - 名前を付けて読みやすくしていく
 - デフォルトでは型情報がないことが多い
- デコンパイラを信用しすぎない
 - アセンブリを確認して整合性を確認する
 - 手動で修正する
- 順番に回答する必要はないので解けそうな問題から解く

マルウェアの動作概要



加工済みのGZF

- MWSの問題作成にあたって以下の点に変更を加えています
 - FLIRTシグネチャを使ったライブラリ関数のシンボルの適用
 - 暗号鍵の変更と暗号文字列への適用
 - 暗号化文字列のラベル変更
 - dec_で始まるラベルは作問者が作成したヒントのための復号後の文字列を示したラベルです



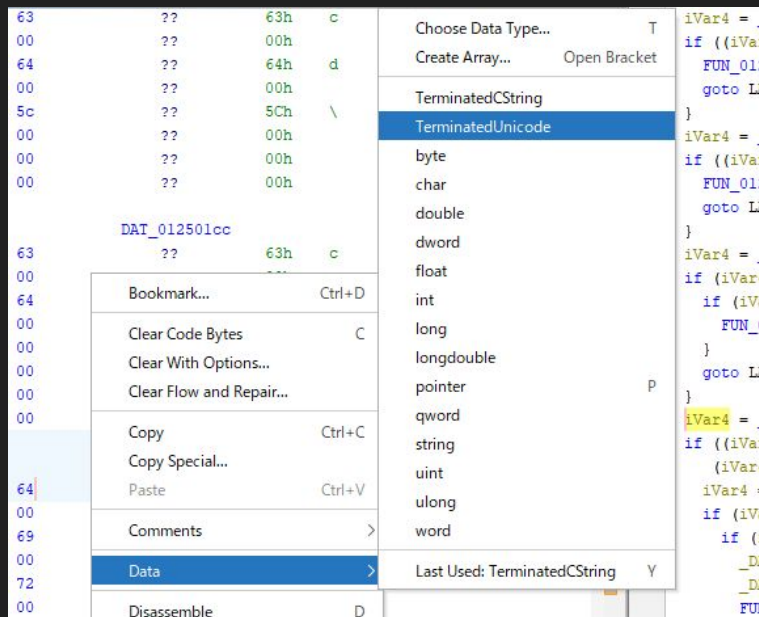
Ghidra Tips: Go To

- Gキーで開くGo Toダイアログにアドレスやlabelを入力
 - 任意の場所に簡単に移動
 - 問題ではアドレスが指定されていることが多いので必須!



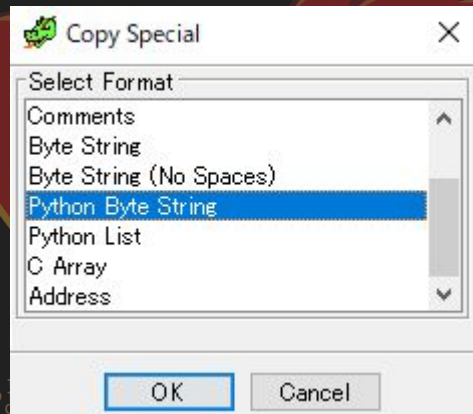
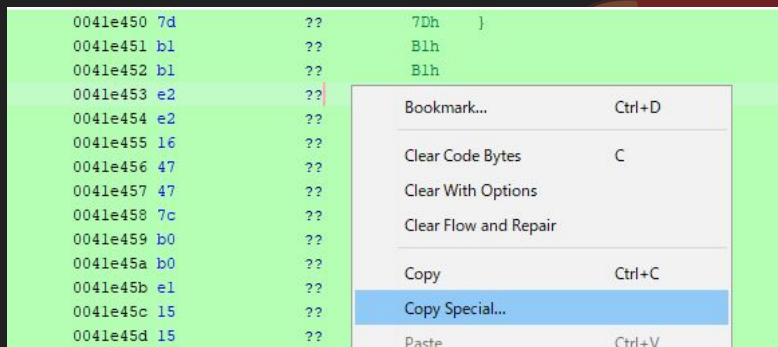
Ghidra Tips: データの定義

- 定義したいデータの先頭を右クリック
 - コンテキストメニューからData → 定義する形式を選択



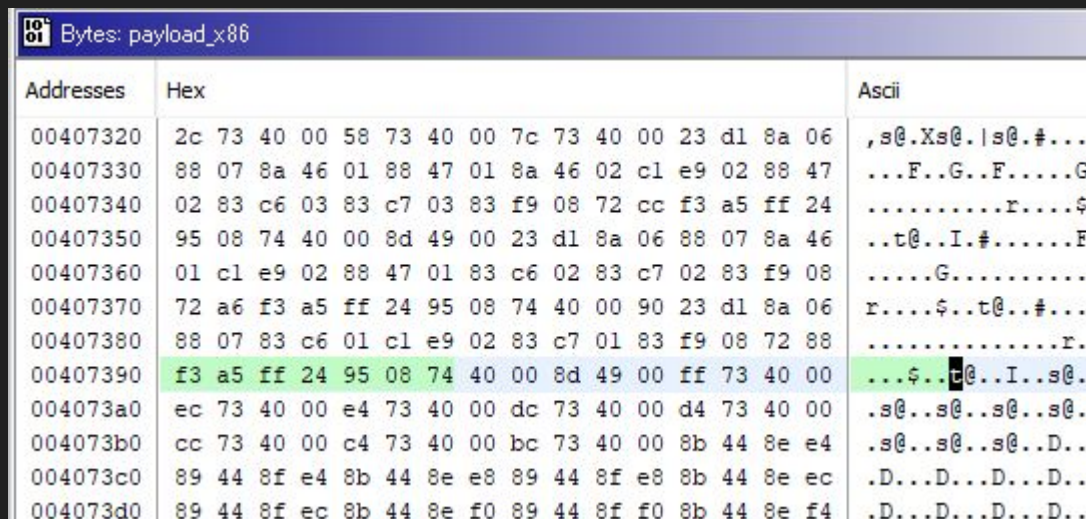
Ghidra Tips: データのコピー1

- コピーしたい範囲を選択
 - コンテキストメニューからCopy Special => 好きなフォーマットを選択



Ghidra Tips: データのコピー2

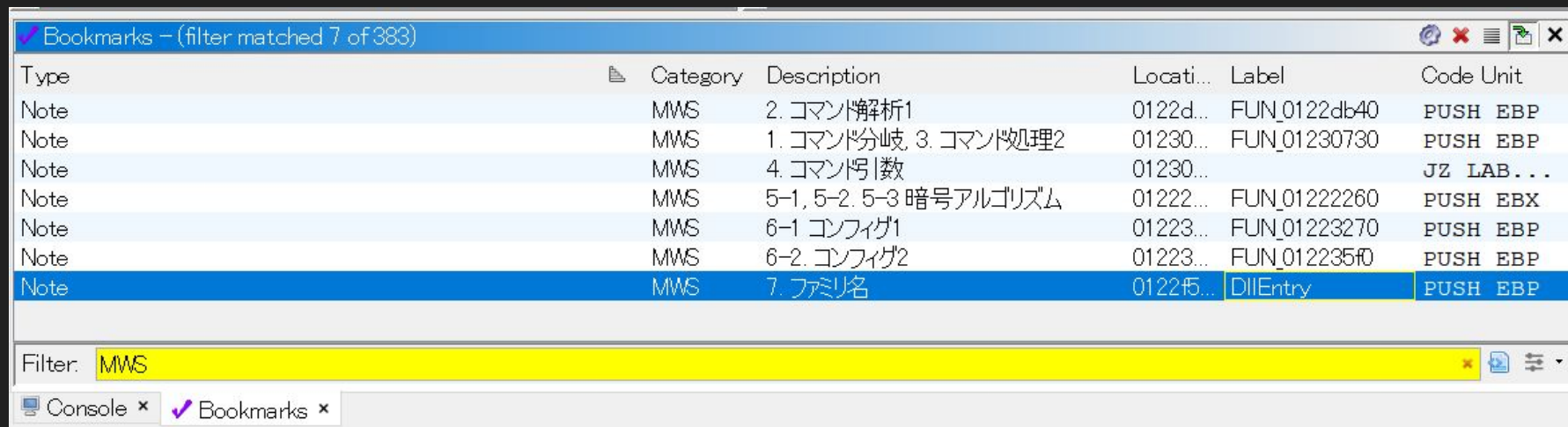
- Bytesウィンドウでコピーする



Addresses	Hex	Ascii
00407320	2c 73 40 00 58 73 40 00 7c 73 40 00 23 d1 8a 06	,s@.Xs@. s@.#...
00407330	88 07 8a 46 01 88 47 01 8a 46 02 c1 e9 02 88 47	...F..G..F.....G
00407340	02 83 c6 03 83 c7 03 83 f9 08 72 cc f3 a5 ff 24r....\$
00407350	95 08 74 40 00 8d 49 00 23 d1 8a 06 88 07 8a 46	..t@..I.#.....F
00407360	01 c1 e9 02 88 47 01 83 c6 02 83 c7 02 83 f9 08G.....
00407370	72 a6 f3 a5 ff 24 95 08 74 40 00 90 23 d1 8a 06	r....\$.t@.#...
00407380	88 07 83 c6 01 c1 e9 02 83 c7 01 83 f9 08 72 88r.
00407390	f3 a5 ff 24 95 08 74 40 00 8d 49 00 ff 73 40 00	...\$.t@..I..s@.
004073a0	ec 73 40 00 e4 73 40 00 dc 73 40 00 d4 73 40 00	.s@..s@..s@..s@.
004073b0	cc 73 40 00 c4 73 40 00 bc 73 40 00 8b 44 8e e4	.s@..s@..s@..D..
004073c0	89 44 8f e4 8b 44 8e e8 89 44 8f e8 8b 44 8e ec	.D...D...D...D..
004073d0	89 44 8f ec 8b 44 8e f0 89 44 8f f0 8b 44 8e f4	.D...D...D...D..

Bookmarks

問題に関連するアドレスをBookmarkとして登録済み



The screenshot shows a debugger window titled "Bookmarks - (filter matched 7 of 383)". The window contains a table of bookmarks with the following columns: Type, Category, Description, Location, Label, and Code Unit. The table is filtered to show only bookmarks with the category "MWS". The last row is selected.

Type	Category	Description	Locati...	Label	Code Unit
Note	MWS	2. コマンド解析1	0122d...	FUN_0122db40	PUSH EBP
Note	MWS	1. コマンド分岐, 3. コマンド処理2	01230...	FUN_01230730	PUSH EBP
Note	MWS	4. コマンド引数	01230...		JZ LAB...
Note	MWS	5-1, 5-2, 5-3 暗号アルゴリズム	01222...	FUN_01222260	PUSH EBX
Note	MWS	6-1. コンフィグ1	01223...	FUN_01223270	PUSH EBP
Note	MWS	6-2. コンフィグ2	01223...	FUN_012235f0	PUSH EBP
Note	MWS	7. ファミリ名	0122f5...	DllEntry	PUSH EBP

Filter: MWS

Console x Bookmarks x

問題一覧

Static Analysis

1. コマンド分岐

2

2. コマンド解析1

2

3. コマンド処理2

2

4. コマンド引数

2

5-1. 暗号アルゴリズム

2

5-2. 暗号鍵

3

5-3. 復号スクリプト作成

3

6-1. コンフィグ1

3

6-2. コンフィグ2

3

7. ファミリ名

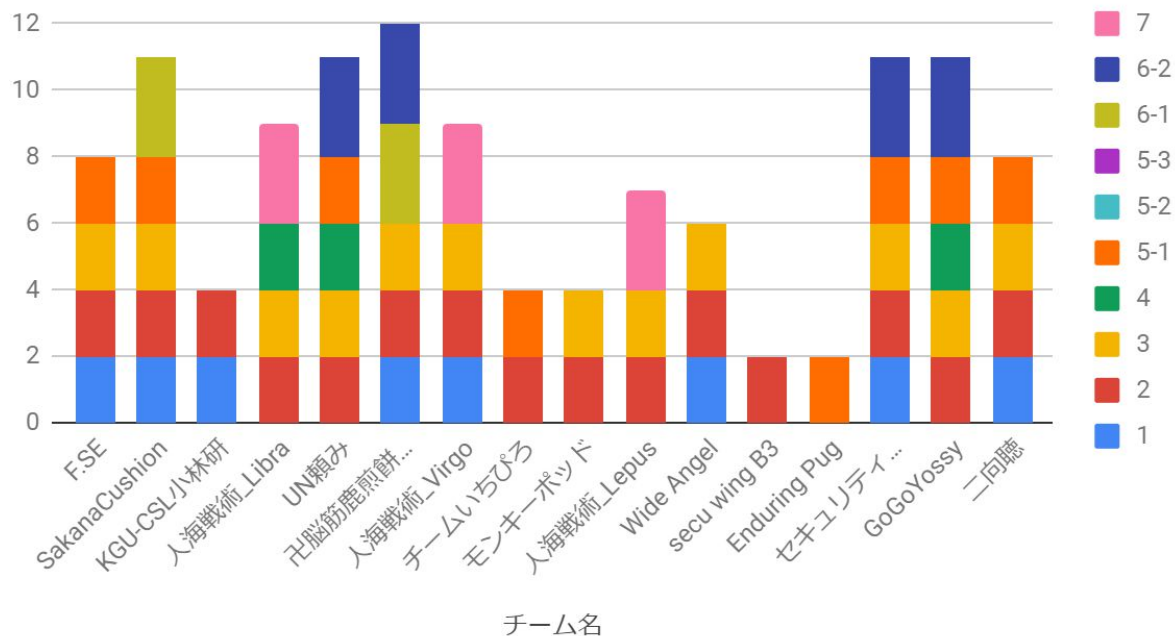
3

競技中はここまで



結果

静的解析 得点



1. コマンド分岐

1. コマンド分岐

2

Static Analysis はマルウェアの静的解析に関する問題です。
問題ファイル `sample.dll.gzf` をダウンロードし、Ghidra で
解析して以降の問いに回答してください。

FUN_01230730 内で、マルウェアに実装されているコマン
ド数を答えよ。コマンド文字列が重複する分岐は1つのコマ
ンドとしてカウントすること。

 `sample.dll.gzf`

0/2 attempts

Flag

Submit



1. コマンド分岐

- 復号した文字列とDAT_01255e78を strcmp で比較する処理がある
 - コマンド文字列の比較処理
- DAT_01255e78の参照を追って コマンド数を数える

```
Decompile: FUN_01230730 - (sample.dll)
40  if (iVar3 == 0) goto LAB_01230c3b;
41  iVar4 = __strcmp((ushort *)&DAT_01255e78, (ushort *)&dec_Sysinfo);
42  if (iVar4 == 0) {
43      FUN_0122db40();
44      goto LAB_01230c3b;
45  }
46  DVar5 = __strcmp((ushort *)&DAT_01255e78, (ushort *)&dec_Download);
47  if ((DVar5 == 0) && (iVar3 == 2)) {
48      local_4024 = DVar5;
49      _memset(&DAT_01255666, 0, 0x800);
50      FUN_012399ec((short *)&DAT_01255666, 0x800, 0x1256278);
51      _DAT_01255560 = param_1;
52      hHandle = CreateThread((LPSECURITY_ATTRIBUTES)0x0, 0, FUN_0122b6c0, &DAT_012555
53      if (hHandle == (HANDLE)0x0) {
54          func_0x01228ef0(param_1, 5, L"Error2:\r\nCreateThread DownloadFile[%s] Error
55      }
56      WaitForSingleObject(hHandle, 500);
57      goto LAB_01230c3b;
58  }
59  iVar4 = __strcmp((ushort *)&DAT_01255e78, (ushort *)&dec_UploadFileOk);
60  if ((iVar4 == 0) && (iVar3 == 2)) {
61      FUN_0122c390(param_1, (undefined4 *)&DAT_01256278);
62      goto LAB_01230c3b;
63  }
64  iVar4 = __strcmp((ushort *)&DAT_01255e78, (ushort *)&dec_RemoteRun);
65  if ((iVar4 == 0) && (iVar3 == 2)) {
66      _memset(local_4024, 0, 0x400);
67      FUN_0122c390(param_1, (undefined4 *)&DAT_01256278);
68      goto LAB_01230c3b;
69  }
70  goto LAB_01230c3b;
```


1. コマンド分岐

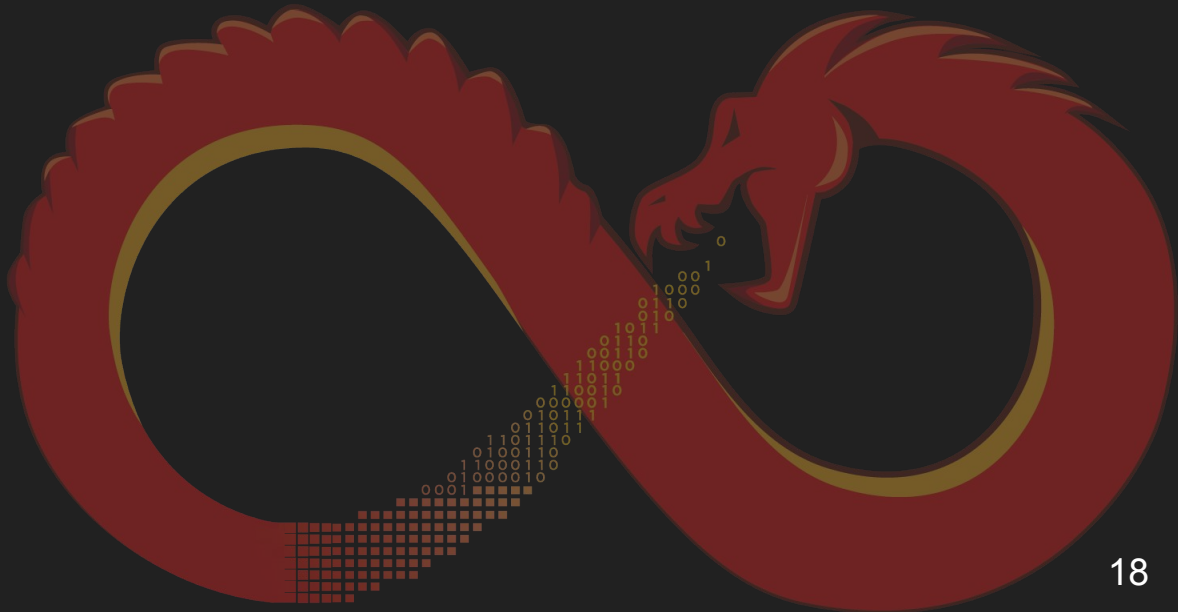
- いくつかの文字列は定義されていないのでデータ型を定義する
 - Data → TerminatedUnicode

```
116 goto LAB_01230c3b;
117 }
118 iVar4 = __stricmp((ushort *)&DAT_01255e78, (ushort *)&DAT_012501bc);
119 if ((iVar4 == 0) && (iVar3 == 1)) {
120     FUN_0122a490((SOCKET)param_1, L".");
121     goto LAB_01230c3b;
122 }
123 iVar4 = __stricmp((ushort *)&DAT_01255e78, (ushort *)&DAT_012501c4);
124 if ((iVar4 == 0) && (iVar3 == 1)) {
125     FUN_0122a490((SOCKET)param_1, (LPCWSTR)&DAT_0124f08c);
126     goto LAB_01230c3b;
127 }
```

00 00				
		i_cd\012501c4		
012501c4	63 00 64	unicode	u"cd\\"	
	00 5c 00			
	00 00			
		DAT_012501cc		
012501cc	63	??	63h	c
012501cd	00	??	00h	
012501ce	64	??	64h	d
012501cf	00	??	00h	
012501d0	00	??	00h	
012501d1	00	??	00h	

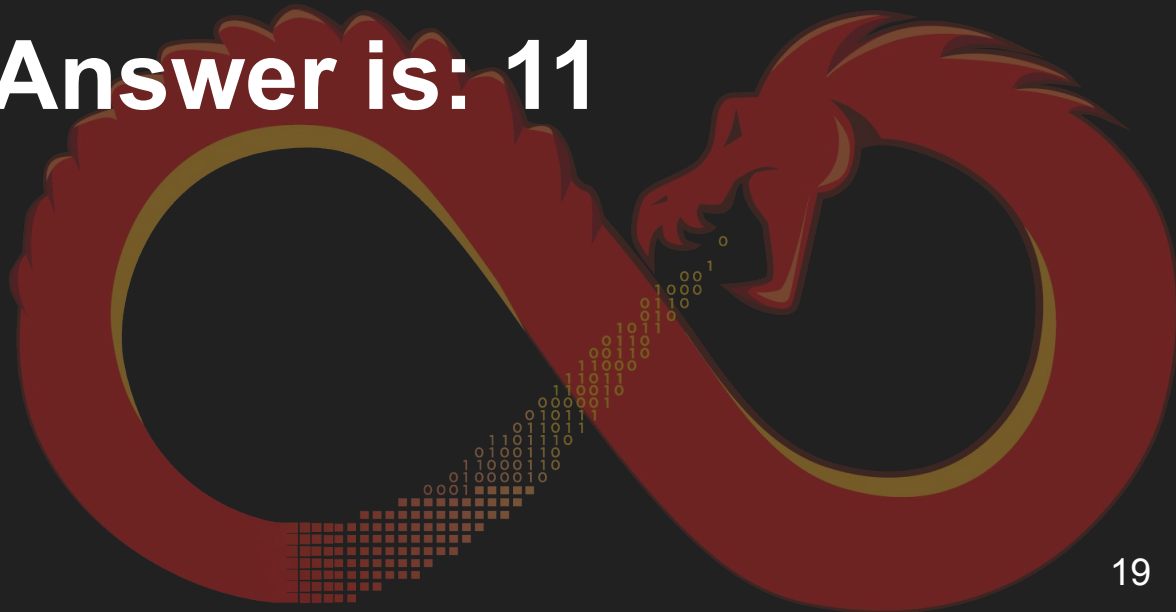
1. コマンド分岐

- Sysinfo
- Download
- UploadFileOk
- RemoteRun
- Computer
- Shell
- cd
- dir
- ls
- del
- Exit



1. コマンド分岐

The Answer is: 11



2. コマンド解析1

Challenge 0 Solves ×

2. コマンド解析1

2

FUN_0122db40の挙動を以下から選択せよ。

- レジストリの登録による永続化
- マルウェアのインストール
- ネットワーク内の他端末の情報取得
- システム情報の取得

0/1 attempt

Submit



2. コマンド解析1

- コマンド文字列から推定
- 出力文字列から推定

```
40 if (iVar3 == 0) goto LAB_01230c3b;
41 iVar4 = __stricmp((ushort *)&DAT_01255e78, (ushort *)&dec_Sysinfo);
42 if (iVar4 == 0) {
43     FUN_0122db40();
44     goto LAB_01230c3b;
45 }
```

```
74 } while (*psVar9 != 0);
75 puVar13 = (undefined4 *)L"\r\nSystem Infomation\r\n-----\r\n";
76 for (iVar6 = 0x1f; iVar6 != 0; iVar6 = iVar6 + -1) {
77     *puVar13 = *puVar13;
```

```
327 FUN_01228c40(s, 1, L"ComputerName:    %s\r\n");
```

```
309 }
310 FUN_01228c40(s, 1, L"RegisterUser:    %s\r\n");
```

```
361 FUN_01228c40(s, 1, L"System Directory: %s\r\n\r\n");
362 FUN_01228c40(s, 1, L"Number of Processors:    %d\r\n");
```

```
410 FUN_01228c40(local_8154, 1, L"RAM:      %dMB Total, %dMB Free.\r\n");
411 FUN_01228c40(local_8154, 1, L"DisplayMode: %d x %d, %dHz, %dbit\r\n");
412 FUN_01228c40(local_8154, 1, L"Uptime:    %d Days %02u:%02u:%02u\r\n");
```

2. コマンド解析1

The Answer is: システム情報の取得

3. コマンド処理2

Challenge 0 Solves ×

3. コマンド処理2

2

FUN_01230730 を読み、指定したファイルを実行するコマンドに関連する比較処理が開始するアドレスを解答せよ。
解答例:FUN_01230730の先頭の場合01230730

0/2 attempts

3. コマンド処理2

- RemoteRunの処理の中でWinExecが実行されている
 - WinExecの引数は初期化されたあとFUN_012221d0に渡される
 - おそらくこの関数でC2からのデータをコピー
- この処理が開始されるアドレスを確認する

```
64 iVar4 = __stricmp((ushort *)&DAT_01255e78, (ushort *)&dec_RemoteRun);
65 if ((iVar4 == 0) && (iVar3 == 2)) {
66     _memset(local_420, 0, 0x400);
67     FUN_012221d0((LPCWSTR)&DAT_01256278, (int)local_420);
68     WinExec(local_420, 0);
69     goto LAB_01230c3b;
70 }
```

```
LAB_01230871 XREF[2]:
01230871 68 98 1e PUSH dec_RemoteRun
                26 01
01230876 68 78 5e PUSH DAT_01255e78
                25 01
0123087b e8 ca 93 CALL __stricmp
                00 00
01230880 83 c4 08 ADD ESP, 0x8
01230883 85 c0 TEST EAX, EAX
01230885 75 3e JNZ LAB_012308c5
01230887 83 fb 02 CMP EBX, 0x2
0123088a 75 39 JNZ LAB_012308c5
0123088c 68 00 04 PUSH 0x400
                00 00
01230891 50 PUSH EAX
01230892 8d 85 e4 LEA EAX=>local_420, [EBP + 0xfffffde4]
                fb ff ff
01230898 50 PUSH EAX
01230899 e8 c2 23 CALL _memset
                00 00
```


3. コマンド処理2

The Answer is: 01230871



4. コマンド引数

Challenge 0 Solves

4. コマンド引数 2

FUN_01230730の第2引数に`dir /c "C:\Users\ghidra master\Downloads"`が与えられたとき、FUN_01229310の終了時のアドレス01230781時点で01256278に格納されている文字列を答えよ。

0/2 attempts

Flag

Submit



4. コマンド引数

- 問題文で指定された文字列はFUN_01229310の第1引数に渡されている

dir /s "C:¥Users¥ghidra master¥Downloads"

```
5 void __fastcall FUN_01230730(HANDLE param_1, LPCWSTR param_2)
```

```
39 iVar3 = FUN_01229310(param_2);
```

```
4 int __fastcall FUN_01229310(LPCWSTR param_1)
5
6 {
7     int iVar1;
8     int iVar2;
9     int iVar3;
10    int iVar4;
11    wchar_t *pwVar5;
12    int local_20;
13    wchar_t *local_1c;
14    int local_14;
15    LPCWSTR local_10;
16    undefined *local_c;
17    int local_8;
18
19    local_20 = 0;
20    local_14 = 0;
21    local_8 = 0;
22    iVar1 = strlenW(param_1);
23    _DAT_01255e78 = 0;
24    if (iVar1 < 0) {
25        return 0;
26    }
27    local_c = &DAT_01255e78;
28    local_10 = param_1;
29    do {
30        pwVar5 = param_1 + local_20;
```

4. コマンド引数

DAT_01256278を参照するmemcpyの処理を追う

```
0122946c 51          PUSH     param_1=>DAT_01255e78
0122946d 89 55 f8     MOV     dword ptr [EBP + local_c],EDX=>DAT_01256278
01229470 e8 7b ac     CALL    FID_conflict:_memcpy
```

```
72          FID_conflict:_memcpy(local_c,local_1c,iVar2 * 2);
```

memcpy (void * _Dst, void * _Src, size_t _Size)

4. コマンド引数

- 引数からスペースを確認し、スペース区切りで文字列を分割する
- 0x400ごとに01255e78にコピー

```
53 local_lc = pwVar5 + 1;
54 iVar2 = __wcsnicmp(pwVar5,L"\"",1);
55 if ((iVar2 == 0) || (local_20 == iVar1)) {
56     if (0x125ae77 < (int)local_c) {
57         return local_8;
58     }
59     local_10 = (LPCWSTR)0x0;
60     iVar2 = local_20;
61     pwVar5 = local_lc;
62     while ((iVar2 <= iVar1 && (iVar3 = __wcsnicmp(pwVar
```

```
28 local_10 = param_1;
29 do {
30     pwVar5 = param_1 + local_20;
31     iVar2 = __wcsnicmp(pwVar5,L" ",1);
32     if ((iVar2 == 0) || (local_20 == iVar1)) {
33         if (0x125ae77 < (int)local_c) {
34             return local_8;
35         }
36         iVar2 = __wcsnicmp(pwVar5 + 1,L" ",1);
37         if (iVar2 != 0) {
38             local_14 = local_20 - local_14;
39             if (0x200 < local_14) {
40                 local_14 = 0x200;
41             }
42             local_8 = local_8 + 1;
43             FID_conflict: memcpy(local_c,local_10,local_14 * 2);
44             local_10 = param_1 + local_20 + 1;
45             local_14 = local_20;
46             local_c = local_c + 0x400;
47         }

```

4. コマンド引数

- 引数からスペースを確認し、スペース区切りで文字列を分割する
- 0x400ごとに01255e78にコピー

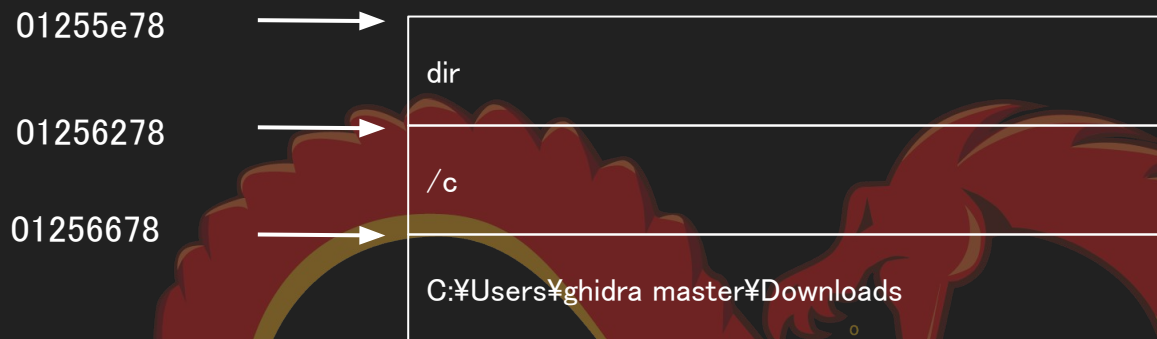
```
28 }
29 local_c = &DAT_01255e78;

37 if (local_14 < 0) {
38     local_14 = local_20 - local_14;
39     if (0x200 < local_14) {
40         local_14 = 0x200;
41     }
42 }

42     local_8 = local_8 + 1;
43     FID_conflict:_memcpy(local_c, local_10, local_14 * 2);
44     local_10 = param_1 + local_20 + 1;
45     local_14 = local_20;
46     local_c = local_c + 0x400;
```

4. コマンド引数

- コピー処理が終わった時点のデータ構造



- $01256278 = 01255e78 + 400 * 1$

4. コマンド引数

The Answer is: /c



5-1. 暗号アルゴリズム

Challenge 0 Solves

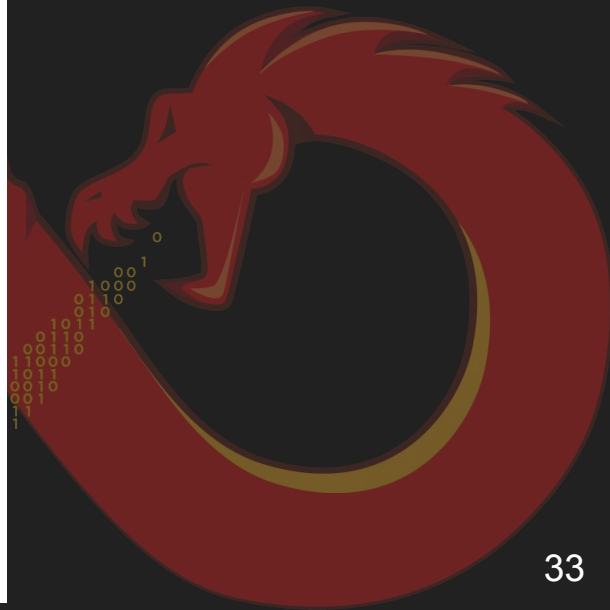
5-1. 暗号アルゴリズム 2

FUN_01222260で利用されている暗号アルゴリズムを以下から選択せよ。

- RC4
- RC5
- RC6
- AES-128
- AES-192
- AES-256
- DES
- 3DES
- Salsa20
- Chacha20

0/1 attempt

Submit



5-1. 暗号アルゴリズム

- 第1引数の復号対象の文字列はFUN_01221530に渡される
- 第2引数はグローバル変数をワイド文字列処理した値がコピーされる

```
64 FUN_01222260(s_DB0FF10DC6FB766D236E9700D79F9DC9_012540c8,0x12633d8);
65 FUN_01222260(s_53FC314787A4FB30D338D22E85160DD8_01254048,0x12632dc);
66 FUN_01222260(s_E5F64CF555549224A005C708BED1C6FD_01254088,0x1263358);
67 FUN_01222260(s_7F22BB27CEB4ACDDE2C815731DEC9116_01253f88,0x1263158);
68 FUN_01222260(s_40E69760490067A0BF902C0A4D2F13D7_01253f48,0x12630dc);
69 FUN_01222260(s_CD4433E064599A3C3E2D30E53A534F95_01253fc8,0x12631d8);
70 FUN_01222260(s_7607C5117D9339A2BF192027CD99549F_01254008,0x1263258);
71 FUN_01222260(s_AD3E5B6223BDD1D293E93D6E1EDB2117_01253e88,0x1262398);
72 FUN_01222260(s_25A37CBF05937FD6B57D0C3054E6876E_01253e48,0x1262318);
73 FUN_01222260(s_C9741371FE0E104A7B7C96316CB9A41E_01253ec8,0x1262418);
```

```
2 void __fastcall FUN_01222260(char *param_1,int param_2)
3
4 {
5     WCHAR WVar1;
6     longlong lVar2;
7     int cchWideChar;
8     LPWSTR lpWideCharStr;
9     LPWSTR pWVar3;
10
11     _memset(&DAT_0126bc98,0,0x400);
12     FUN_01221530(param_1);
13     cchWideChar = MultiByteToWideChar(0,1,&DAT_0126bc98,-1,(LPWSTR)0x0,0);
14     if (cchWideChar != 0) {
15         lVar2 = (ulonglong)(cchWideChar + 1) * 2;
16         lpWideCharStr =
17             (LPWSTR)FUN_01230efa(-(uint)((int)((ulonglong)lVar2 >> 0x20) != 0) | (u
18             MultiByteToWideChar(0,1,&DAT_0126bc98,-1,lpWideCharStr,cchWideChar);
19         lpWideCharStr[cchWideChar] = L'\0';
20         pWVar3 = lpWideCharStr;
21         do {
22             WVar1 = *pWVar3;
23             pWVar3 = pWVar3 + 1;
24             *(WCHAR *)((param_2 - (int)lpWideCharStr) + -2 + (int)pWVar3) = WVar1;
25         } while (WVar1 != L'\0');
26         _free(lpWideCharStr);
```

5-1. 暗号アルゴリズム

- FUN_01221530内ではグローバル変数が参照されている
- 引数のデータを別のFUN_01221900関数で処理している

```
54 }  
55 FUN_01221900(DAT_0126bc94, (int)local_408):  
56 pcVar5 = &DAT_0126bc98;
```

```
45 }  
46 if (iVar4 < 0) break;  
47 param_1 = param_1 + 2;  
48 *pcVar5 = (char)iVar2 * '\x10' + (char)iVar4;  
49 pcVar5 = pcVar5 + 1;  
50 cVar1 = *param_1;  
51 }  
52 *pcVar5 = '\0';  
53 }
```

```
14 pcVar5 = local_408;
```

5-1. 暗号アルゴリズム

- FUN_01221000の固定値とAES:vftableを見つける
 - Rijndael S-box

```
0122100c c7 85 00      MOV      dword ptr [EBP + local_104],0x7b777c63
          ff ff ff
          63 7c 77 7b

01221016 56              PUSH     ESI
01221017 57              PUSH     EDI
01221018 c7 85 04      MOV      dword ptr [EBP + local_100],0xc56f6bf2
          ff ff ff
          f2 6b 6f c5

01221022 c7 03 c4      MOV      dword ptr [EBX],AES::vftable
          ec 24 01

01221028 c7 85 08      MOV      dword ptr [EBP + local_fc],0x2b670130
          ff ff ff
          30 01 67 2b

01221032 c7 85 0c      MOV      dword ptr [EBP + local_f8],0x76abd7fe
          ff ff ff
          fe d7 ab 76

0122103c c7 85 10      MOV      dword ptr [EBP + local_f4],0x7dc982ca
          ff ff ff
          ca 82 c9 7d

01221046 c7 85 14      MOV      dword ptr [EBP + local_f0],0xf04759fa
```

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

The column is determined by the least significant nibble, and the row by the most significant nibble. For example, the value $9a_{16}$ is converted into $b8_{16}$.

5-1. 暗号アルゴリズム

- FUN_01221900のアルゴリズムを調査
 - AESと類似する

High-level description of the algorithm [\[edit \]](#)

round version.

1. `KeyExpansion` – round keys are derived from the cipher key using the [AES key schedule](#). AES requires a separate 128-bit round key block for each round plus one more.
2. Initial round key addition:
 1. `AddRoundKey` – each byte of the state is combined with a byte of the round key using [bitwise xor](#).
3. 9, 11 or 13 rounds:
 1. `SubBytes` – a [non-linear](#) substitution step where each byte is replaced with another according to a [lookup table](#).
 2. `ShiftRows` – a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
 3. `MixColumns` – a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
 4. `AddRoundKey`
4. Final round (making 10, 12 or 14 rounds in total):
 1. `SubBytes`
 2. `ShiftRows`
 3. `AddRoundKey`

5-1. 暗号アルゴリズム

- AESのビット数は？
 - ラウンド数で決まる

The key size used for an AES cipher specifies the number of transformation rounds that convert the input, called the `plaintext`, into the final output, called the `ciphertext`. The number of rounds are as follows:

- 10 rounds for 128-bit keys.
- 12 rounds for 192-bit keys.
- 14 rounds for 256-bit keys.

5-1. 暗号アルゴリズム

- ラウンド数は10

```
42  local_20 = 9;
```

```
73     local_18[iVar8 * 4 + 1] = *(byte *) (local_18[iVar8 * 4 + 1] + 0x104 + (int)this);
74     local_18[iVar8 * 4 + 2] = *(byte *) (local_18[iVar8 * 4 + 2] + 0x104 + (int)this);
75     local_18[iVar8 * 4 + 3] = *(byte *) (local_18[iVar8 * 4 + 3] + 0x104 + (int)this);
76     iVar8 = iVar8 + 1;
77 } while (iVar8 < 4);
78 iVar8 = 0;
79 pbVar7 = local_24;
80 do {
81     local_18[iVar8] = local_18[iVar8] ^ pbVar7[-4];
82     local_18[iVar8 + 4] = local_18[iVar8 + 4] ^ *pbVar7;
83     local_18[iVar8 + 8] = local_18[iVar8 + 8] ^ pbVar7[4];
84     local_18[iVar8 + 0xc] = local_18[iVar8 + 0xc] ^ pbVar7[8];
85     iVar8 = iVar8 + 1;
86     pbVar7 = pbVar7 + 1;
87 } while (iVar8 < 4);
88 if (local_20 != 0) {
89     FUN_01221c60((int)local_18);
90 }
91 local_20 = local_20 + -1;
92 local_24 = local_24 + -0x10;
93 } while (-1 < local_20);
```

5-1. 暗号アルゴリズム

The Answer is: AES-128

5-2. 暗号鍵

Challenge 0 Solves ×

5-2. 暗号鍵

3

FUN_01222260の暗号化処理で使用される鍵を16進数で答えよ。解答例:鍵が `Key`の場合は`4b6579`

Flag Submit

5-2. 暗号鍵

- AESではKeyExpansionで暗号鍵とRijndael S-boxを使って暗号化処理に必要な鍵を生成する

1. KeyExpansion – round keys are derived from the cipher key using the [AES key schedule](#). AES requires a separate 128-bit round key block for each round plus one more.

- FUN_01221b10にRijndael S-boxとFUN_01221000の第1引数を渡している

```
265 local_10c = 0x631469e1;
266 local_108 = 0x7d0c2155;
267 puVar2 = local_204;
268 puVar3 = (undefined4 *) ((int)this + 0x104);
269 for (iVar1 = 0x40; iVar1 != 0; iVar1 = iVar1 + -1) {
270     *puVar3 = *puVar2;
271     puVar2 = puVar2 + 1;
272     puVar3 = puVar3 + 1;
273 }
274 FUN_01221b10(this,param_1,(int)this + 0x204);
```

5-2. 暗号鍵

- FUN_01221000はKeyExpansionの処理のよう

```
42 pbVar1 = (byte *) (pbVar1 + local_24);
43 do {
44     if (iVar8 == 0) {
45         bVar7 = *(byte *) ((CONCAT11(pbVar1[-0xd], pbVar1[-5]) & 0xff) + 4 + (int) this);
46         bVar6 = *(byte *) (pbVar1[-1] + 4 + (int) this);
47         local_15 = *(byte *) (pbVar1[-0xd] + 4 + (int) this);
48         bVar4 = (local_15[local_20] ^ *(byte *) (pbVar1[-9] + 4 + (int) this);
49         pbVar2 = pbVar1;
50     }
51     else {
52         local_15 = pbVar1[iVar8 + 0xb];
53         pbVar2 = pbVar1 + iVar8;
54         bVar4 = pbVar2[-1];
55         bVar7 = pbVar2[3];
56         bVar6 = pbVar2[7];
57     }
58     *pbVar2 = pbVar2[-0x10] ^ bVar4;
59     pbVar2[4] = pbVar2[-0xc] ^ bVar7;
60     bVar4 = pbVar2[-4];
61     pbVar1[iVar8 + 8] = pbVar2[-8] ^ bVar6;
62     pbVar1[iVar8 + 0xc] = bVar4 ^ local_15;
63     iVar8 = iVar8 + 1;
64 } while (iVar8 < 4);
65 local_24 = local_24 + 0x10;
66 local_20 = local_20 + 1;
67 } while (local_24 < 161);
```

The key schedule [\[edit \]](#)

Define:

- N as the length of the key in 32-bit words: 4 words for AES-128, 6 words for AES-192, and 8 words for AES-256
- K_0, K_1, \dots, K_{N-1} as the 32-bit words of the original key
- R as the number of round keys needed: 11 round keys for AES-128, 13 keys for AES-192, and 15 keys for AES-256^[note 4]
- $W_0, W_1, \dots, W_{4R-1}$ as the 32-bit words of the expanded key^[note 5]

Also define RotWord as a one-byte left circular shift:^[note 6]

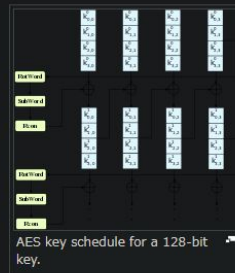
$$\text{RotWord}([b_0 \ b_1 \ b_2 \ b_3]) = [b_1 \ b_2 \ b_3 \ b_0]$$

and SubWord as an application of the AES S-box to each of the four bytes of the word:

$$\text{SubWord}([b_0 \ b_1 \ b_2 \ b_3]) = [S(b_0) \ S(b_1) \ S(b_2) \ S(b_3)]$$

Then for $i = 0 \dots 4R - 1$:

$$W_i = \begin{cases} K_i & \text{if } i < N \\ W_{i-N} \oplus \text{SubWord}(\text{RotWord}(W_{i-1})) \oplus rcon_{i/N} & \text{if } i \geq N \text{ and } i \equiv 0 \pmod{N} \\ W_{i-N} \oplus \text{SubWord}(W_{i-1}) & \text{if } i \geq N, N > 6, \text{ and } i \equiv 4 \pmod{N} \\ W_{i-N} \oplus W_{i-1} & \text{otherwise.} \end{cases}$$



5-2. 暗号鍵

- FUN_01221000の引数を辿っていく
 - DAT_0126c098にたどり着く

```
274 FUN_01221b10(this,param_1,(int)this + 0x204);
```

```
Decompile: FUN_01221000 - (sample.dll)
1
2 undefined4 * __thiscall FUN_01221000(void *this,int param_1)
3
```

```
62 DAT_0126bc94 = FUN_01221000(this,0x126c098);
```

```
57 DAT_0126c0a0 = 0x6e75;
58 _DAT_0126c098 = 0x66726f6667473756a;
```

5-2. 暗号鍵

- アセンブリを確認するとDAT_0126c098には文字列“justforfun”が代入されている

```
0122f30a f3 0f 7e    MOVQ    XMM0, qword ptr [s_justforfun_0124ecc8]
          05 c8 ec
          24 01
0122f312 a3 58 55    MOV     [DAT_01255558], EAX
          25 01
0122f317 66 a1 d0    MOV     AX, [s_un_0124ecc8+8]
          ec 24 01
0122f31d 66 a3 a0    MOV     [DAT_0126c0a0], AX
          c0 26 01
0122f323 a0 d2 ec    MOV     AL, [s_0124ecc8+10]
          24 01
0122f328 68 b4 02    PUSH   0x2b4
          00 00
0122f32d 66 0f d6    MOVQ   qword ptr [DAT_0126c098], XMM0
          05 98 c0
```

```
s_0124ecc8
s_justforfun_0124ecc8
0124ecc8 6a 75 73    ds     "justforfun"
          74 66 6f
          72 66 75
```

5-2. 暗号鍵

- AES-128の鍵長は16バイト(128bit/8bit)なので鍵をヌルで拡張
 - 'justforfun¥x00¥x00¥x00¥x00¥x00¥x00'
- 指定の形式に変換

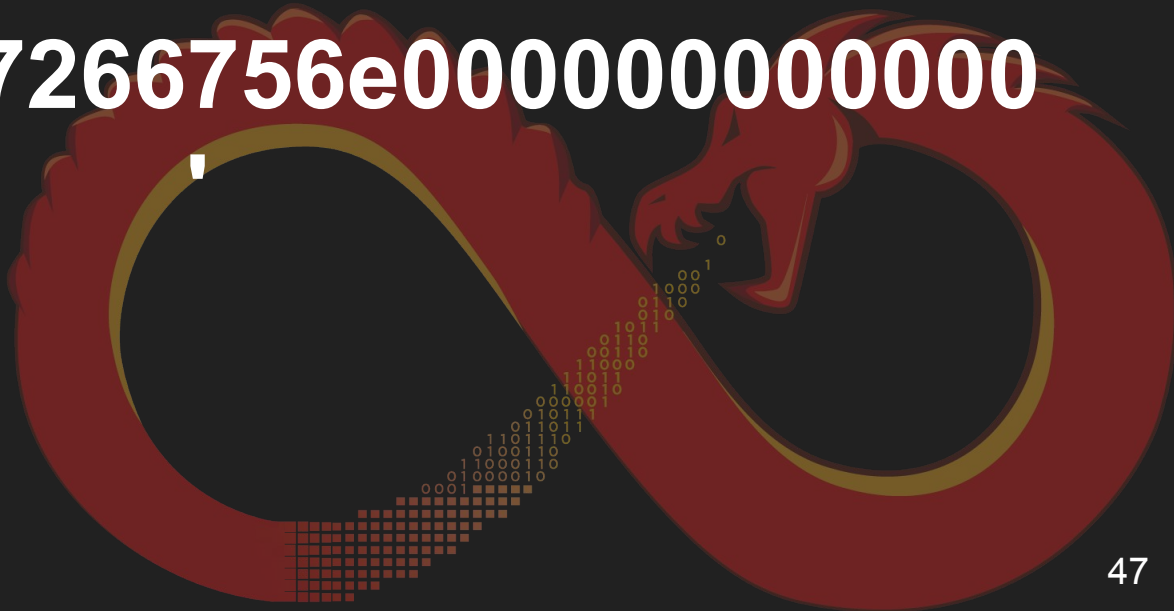
```
>>> key =  
b'justforfun\x00\x00\x00\x00\x00\x00'  
>>> key.hex()  
'6a7573746666f7266756e000000000000'
```



5-2. 暗号鍵

The Answer is:

6a757374666f7266756e000000000000



5-3. 復号スクリプト作成

Challenge 0 Solves ×

5-3. 復号スクリプト作成 3

FUN_0122f250 の一連の処理を読み、FUN_01222260 の処理で以下の文字列を復号した文字列は何になるか答えよ。

`0469cb2edc87cd0557c85273c710cc0e3142a9050eb5ca0618749c05aa57c2491b6ff98ed3cc6cc1e5f09d1a23129e0c`

5-3. 復号スクリプト作成

- 5-1と5-2を基に作成

```
from Crypto.Cipher import AES
key = b'justforfun\x00\x00\x00\x00\x00\x00'
decipher = AES.new(key=key, mode=AES.MODE_ECB)
decipher.decrypt(bytes.fromhex("0469cb2edc87cd0557c85273c710cc0e3142
a9050eb5ca0618749c05aa57c2491b6ff98ed3cc6cc1e5f09d1a23129e0c"))
```

```
>>> decipher.decrypt(bytes.fromhex("0469cb2edc87cd0557c85273c710cc0e314
2a9050eb5ca0618749c05aa57c2491b6ff98ed3cc6cc1e5f09d1a23129e0c"))
b'AES-ECB_is_the_most_basic_of_block_cipher_modes!'
```

5-3. 復号スクリプト作成

The Answer is:

AES-ECB_is_the_most_basic_of_block_cipher_modes!



6-1. コンフィグ

Challenge 0 Solves ×

6-1 コンフィグ1

3

FUN_01223270はコンフィグを復号する関数である。この関数によって復号されるコンフィグデータの先頭8バイトを答えよ 解答例: `000102030A0B0C0D`

0/2 attempts

6-1. コンフィグ1

- デコンパイルでは `_Source+4` 表記
 - `_Source` は `undefined4 *型(4バイト)` なので `4*4` で `0x10`
- Listing View で値を確認
 - `EDI + 0x10` を参照している

```
96 local_8 = 0xffffffff;  
97 _Size = FUN_01223270(this, _Source + 4, (int *)&local_74, (int)local_28, unaff_ESI);  
98 Dest = (byte *)FUN_01230efa(_Size);
```

```
01223762 ff 75 dc    PUSH    dword ptr [EBP + local_28]  
01223765 8d 47 10    LEA    EAX, [EDI + 0x10]  
01223768 51         PUSH    ECX  
01223769 50         PUSH    EAX  
0122376a 8b ca      MOV    ECX, EDX  
0122376c e8 ff fa    CALL   FUN_01223270  
ff ff
```

6-1. コンフィグ1

- EDIの値をたどるとlocal_70のデータを参照している
 - local_70+0x10バイトからが復号対象のデータ
 - リトルエンディアンなのでデータの順番に注意

```
01223698 0f 10 45 94    MOVUPS    XMM0, xmmword ptr [EBP + local_70[0]]
0122369c 8b f8           MOV      EDI, EAX
0122369e 8b 45 d4       MOV      EAX, dword ptr [EBP + local_30]
012236a1 6a 08         PUSH    0x8
012236a3 57           PUSH    EDI
012236a4 89 7d 8c       MOV      dword ptr [EBP + local_78], EDI
012236a7 0f 11 07       MOVUPS    xmmword ptr [EDI], XMM0
```

```
0122361c c7 45 94     MOV      dword ptr [EBP + local_70[0]], 0x30303030
01223623 c7 45 98     MOV      dword ptr [EBP + local_70[4]], 0x32343030
0122362a c7 45 9c     MOV      dword ptr [EBP + local_70[8]], 0x30303030
01223631 c7 45 a0     MOV      dword ptr [EBP + local_70[12]], 0x34333030
01223638 c7 45 a4     MOV      dword ptr [EBP + local_60[0]], 0x23060c80
0122363f c7 45 a8     MOV      dword ptr [EBP + local_60[4]], 0x6ad09821
01223646 c7 45 ac     MOV      dword ptr [EBP + local_60[8]], 0x79e1b36
0122364d c7 45 b0     MOV      dword ptr [EBP + local_60[12]], 0xe050d020
01223654 c7 45 b4     MOV      dword ptr [EBP + local_50[0]], 0xcl92e6a
0122365b c7 45 b8     MOV      dword ptr [EBP + local_50[4]], 0xa0e12207
01223662 c7 45 bc     MOV      dword ptr [EBP + local_50[8]], 0x1a315ccc
01223669 c7 45 c0     MOV      dword ptr [EBP + local_50[12]], 0x883a10c
```

6-1. コンフィグ1

The Answer is: 800C06232198D06A

6-2. コンフィグ2

Challenge 0 Solves ×

6-2. コンフィグ2

3

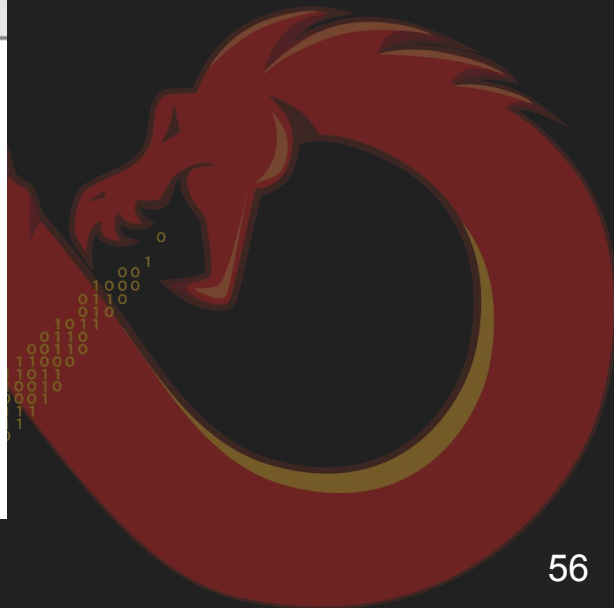
このマルウェアには認証が設定されており、コマンド分岐処理の関数呼び出し前にコンフィグの保持したパスワードを利用する。FUN_012235f0で呼び出されるFUN_0123a6feが復号したコンフィグをグローバル変数にコピーする処理である。パスワードが保存されるグローバル変数のアドレスを答えよ。解答例 DAT_01200000だと01200000

0/2 attempts

6-2. コンフィグ2

- FUN_0123a6feの呼び出しを探すと候補が12個見つかる

```
Decompile: FUN_01235f0 - (sample.dll)
118  if (iVar2 == 0) {
119      _memset(&DAT_01262498, 0, 0xc40);
120      FUN_0123a6fe(&DAT_012624d8, 0x100, 0x125af80);
121      FUN_0123a6fe(&DAT_012625d8, 0x100, 0x125b080);
122      FUN_0123a6fe(&DAT_012626d8, 0x100, 0x125b180);
123      FUN_0123a6fe(&DAT_012627d8, 0x100, 0x125b280);
124      FUN_0123a6fe(&DAT_012628d8, 0x100, 0x125b380);
125      FUN_0123a6fe(&DAT_012629d8, 0x100, 0x125b480);
126      FUN_0123a6fe(&DAT_01262ad8, 0x100, 0x125b580);
127      FUN_0123a6fe(&DAT_01262bd8, 0x100, 0x125b680);
128      FUN_0123a6fe(&DAT_01262cd8, 0x100, 0x125b780);
129      FUN_0123a6fe(&DAT_01262dd8, 0x100, 0x125b880);
130      FUN_0123a6fe(&DAT_01262ed8, 0x100, 0x125b980);
131      FUN_0123a6fe(&DAT_01262fd8, 0x100, 0x125ba80);
132  }
```



6-2. コンフィグ2

- 1. コマンド分岐で参照したFUN_01230730の呼び出し元を確認する
- 認証確認と思われる処理がある
 - `_stricmp`の引数
 - 4. コマンド引数の解答で示したアドレス
 - FUN_01222140でDAT_01262ad8からコピーされた文字列(local_2360)

```
Decompile: FUN_01230200 - (sample.dll)
118 FUN_01222140(&DAT_01262ad8, (int)local_2360);
119 iVar4 = __stricmp((ushort *)&DAT_01255e78, local_2360);
120 if (iVar4 == 0) {
121     iVar5 = 0;
122     local_5374 = 0;
123     FUN_012289c0((SOCKET)param_1, (undefined4 *)&dec_Pw_OK, 0);
```

```
135 FUN_01230730(param_1, local_560);
```

```
Decompile: FUN_01222140 - (sample.dll)
23 * (WCHAR *) ((param_2 - (int)lpWideCharStr) + -2 + (int)pWVar3) = WVar1;
24 } while (WVar1 != L'\0');
```

6-2. コンフィグ2

The Answer is: 01262ad8

7. ファミリ名

Challenge 0 Solves ×

7. ファミリ名

3

これまでの分析結果から、マルウェアファミリの名称を答えよ。

0/2 attempts

7. ファミリ名

- エラー出力文字列をGoogle検索
 - KeyBoyのIoCやYARAルールがヒット

The screenshot shows a Google search interface with the query "UploadFileOK:\r\nReceive". The search results indicate that no exact matches were found. Below the main message, there is a link to a GitHub repository: "malware-indicators/201611_KeyBoy/misp.json at master". The repository information includes the author "Matt Brooks" and a commit message that contains the exact string being searched for: "Receive [%s] ok! Use %2.2f seconds, Average speed %2.2f k/s ...".

```
41
42 strings:
43 //These strings are in ASCII pre-2015 and UNICODE in 2016
44 $error = "Error2" ascii wide
45 //2016 specific:
46 $s1 = "Can't find [%s]!Check the file name and try again!" ascii wide
47 $s2 = "Open [%s] error! %d" ascii wide
48 $s3 = "The Size of [%s] is zero!" ascii wide
49 $s4 = "CreateThread DownloadFile[%s] Error!" ascii wide
50 $s5 = "UploadFile [%s] Error:Connect Server Failed!" ascii wide
51 $s6 = "Receive [%s] Error(Recvd[%d] != Send[%d])!" ascii wide
52 $s7 = "Receive [%s] ok! Use %2.2f seconds, Average speed %2.2f k/s" ascii wide
53 $s8 = "CreateThread UploadFile[%s] Error!" ascii wide
54 //Pre-2016:
55 $s9 = "Ready Download [%s] ok!" ascii wide
56 $s10 = "Get ControlInfo from FileClient error!" ascii wide
57 $s11 = "FileClient has a error!" ascii wide
58 $s12 = "VirtualAlloc SendBuff Error(%d)" ascii wide
59 $s13 = "ReadFile [%s] Error(%d)..." ascii wide
60 $s14 = "ReadFile [%s] Data[Readed(%d) != FileSize(%d)] Error..." ascii wide
61 $s15 = "CreateThread DownloadFile[%s] Error!" ascii wide
62 $s16 = "RecvData MyRecv_Info Size Error!" ascii wide
63 $s17 = "RecvData MyRecv_Info Tag Error!" ascii wide
64 $s18 = "SendData s2ControlInfo_1 Error!" ascii wide
65 $s19 = "SendData s2ControlInfo_3 Error!" ascii wide
66 $s20 = "VirtualAlloc RecvBuff Error(%d)" ascii wide
67 $s21 = "RecvData Error!" ascii wide
68 $s22 = "WriteFile [%s] Error(%d)..." ascii wide
69
70 condition:
71 //MZ header //PE signature
```

7. ファミリー名

- さらに調査するとKeyBoyの亜種EntryShellとわかる

まずはEntryShell内に埋め込まれている文字列を分析してみます。いくつかエラーメッセージを示すような特徴的な文字列が目にとまります。これらの文字列を元に検索してみると、github上に公開されている2016年に作成されたKeyBoyを検知する為のYaraルールが見つかりました。

[rules/malware/APT_KeyBoy.yar at master · Yara-Rules/rules · GitHub](https://github.com/Yara-Rules/rules/blob/master/rules/malware/APT_KeyBoy.yar)

```
text "UTF-16LE", "Ready Download [%s] ok!",0
align 20h
; DATA XREF: sub_2C5F0C+18070
text "UTF-16LE", "Error2!..80h,8ah
text "UTF-16LE", "Can't find [%s]!Check the file name and try a
text "UTF-16LE", "again!..0
; DATA XREF: sub_2C5F0C+24870
text "UTF-16LE", "Error2!..80h,8ah
text "UTF-16LE", "Open [%s] error! Nd",0
align 20h
Format
; DATA XREF: sub_2C5F0C+28870
text "UTF-16LE", "Error2!..80h,8ah
text "UTF-16LE", "The Size of [%s] is zero!",0
align 8
a8rerror
; DATA XREF: sub_2C5F0C+38770
; sub_2C5F6D0+28C7r ...
text "UTF-16LE", "error",0
align 20h
; DATA XREF: backdoor_switch+EB70
text "UTF-16LE", "Error2!..80h,8ah
text "UTF-16LE", "CreateThread DownloadFile[%s] Error!",0
align 20h
; DATA XREF: sub_2C5F6D0+6170
text "UTF-16LE", "Error2!..80h,8ah
text "UTF-16LE", "UploadFile [%s] Error:Connect Server Failed!",0
align 10h
```

2023年5月に発見したEntryShellにあった特徴的なエラーメッセージ

2016年8月に作成されたKeyBoyのYaraルール

図3. 特徴的なエラーメッセージを示す文字列とKeyBoyを検知するためのYaraルールとの比較

virus BULLETIN Covering the global threat landscape

Unveiling activities of Tropic Trooper 2023: deep analysis of Xiangoop Loader and EntryShell payload

Thursday 5 October 16:30 - 17:00, Red room

Suguru Ishimaru (TOCHU Cyber & Intelligence), Hajime Yanagishita (MACNICA) & Yusuke Niwa (TOCHU Cyber & Intelligence)

The Tropic Trooper (also known as KeyBoy and Pirate Panda) is an infamous APT actor that has been highly active since 2011, according to Trend Micro [2]. This group has previously targeted various sectors, including government, healthcare, transportation and high-tech industries in Taiwan, the Philippines, and Hong Kong.

<https://www.virusbulletin.com/conference/vb2023/abstracts/unveiling-activities-tropic-trooper-2023-deep-analysis-xiangoop-loader-and-entryshell-payload/>

標的型攻撃の実態と対策アプローチ

日本を担うサイバーセキュリティの発展2022年度

新しいTTPsやRATなど

ここでは、先に引用させて頂いた公開されている調査報告ではまだ触られていない観測や分析を中心に、少し詳しく紹介します。

— Pirate Panda 中国拠点のスパイフィッシング攻撃

配達されたマクロファイルとEntryShell RAT

中国で人気のチャットツールWeChatで配送されたzipファイルに含まれるExcelファイルのマクロVBAコードは、Base64でエンコードされた値をデコードしてVBScriptのファイルとして保存して実行します。このVBScriptはダウンローダーで、`http[s]://mail.mraden[.]com/win.rar`をダウンロードして; %programdata%win.exeとして実行します。

https://www.macnica.co.jp/business/security/security-reports/pdf/cyberespionage_report_2022.pdf

7. ファミリ名(回答チームからフィードバック)

- PDBもヒントに！
 - マルウェアアナリスト4人で作問したのに抜けていた... 🤪

```
*****  
* Export Library Name *  
*****  
e2 57 6f 72      ds      "WorkDll.dll"  
6b 44 6c  
6c 2e 64 ...
```

```
About sample.dll_ctfd  
-----  
Project File Name:      sample.dll_ctfd  
Last Modified:         Tue Oct 31 12:49:50 JST 2023  
ReadOnly:              false  
Program Name:          sample.dll  
Language ID:           x86:LE:32:default (2.14)  
Compiler ID:           windows  
Processor:             x86  
Endian:               Little  
Address Size:          32  
Minimum Address:       01220000  
Maximum Address:       ffdfffff  
# of Bytes:            322888  
# of Memory Blocks:    8  
# of Instructions:     53307  
# of Defined Data:     5596  
# of Functions:        935  
# of Symbols:          5130  
# of Data Types:       534  
# of Data Type Categories: 36  
Analyzed:              true  
Compiler:              visualstudio:unknown  
Created With Ghidra Version: 10.3.3  
Date Created:          Wed Oct 11 08:27:32 JST 2023  
Executable Format:     Portable Executable (PE)  
Executable Location:   /C:/MWS2023/sample.dll  
Executable MD5:        25d305cb8fc244f0cd1514e8600253bf  
Executable SHA256:     10dc8e8259f286a5ac7dacf6ed7e9cbb1c3be2672503f9f5d30099af3574d5aa  
FSRL:                  file:///C:/MWS2023/sample.dll?MD5=25d305cb8fc244f0cd1514e8600253bf  
PDB Age:               1  
PDB File:              WorkDll.pdb  
PDB GUID:              0709ac9e-laf5-499f-af0a-26d19f205765  
PDB Version:           RSDS  
Preferred Root Namespace Category:  
RTTI Found:           true  
Relocatable:          true  
SectionAlignment:     4096  
Should Ask To Analyze: false
```

7. ファミリ名

The Answer is:
EntryShell or Keyboy

解答まとめ

1. コマンド分岐 FUN_01230730 内で、マルウェアに実装されているコマンド数を答えよ	11
2. コマンド解析1: FUN_0122db40の挙動を以下から選択せよ	システム情報の取得
3. コマンド処理2: FUN_01230730 を読み、指定したファイルを実行するコマンドに関連する処理が開始するアドレスを解答せよ	01230871
4. コマンド引数 FUN_01229310の終了時のアドレス01230781時点で01256278に格納されている文字列を答えよ	/c
5-1. 暗号アルゴリズム FUN_01222260で利用されている暗号アルゴリズムを以下から選択せよ	AES-128
5-2. 暗号鍵: FUN_01222260の暗号化処理で使用する鍵を6進数で答えよ	67573746667266756e000000000000
5-3. 復号スクリプト作成 FUN_0122f250 の一連の処理を読み、FUN_01222260 の処理で以下の文字列を復号した文字列は何になるか答えよ	AES-ECB_is_the_most_basic_of_block_cipher_modes!
6-1 コンフィグ1: FUN_01223270はコンフィグを復号する関数である。この関数によって復号されるコンフィグデータの先頭バイトを答えよ	800C06232198D06A
6-2. コンフィグ2: パスワードが保存されるグローバル変数のアドレスを答えよ	01262ad8
7. ファミリ名: これまでの分析結果から、マルウェアファミリの名称を答えよ	KeyBoy or EntryShell

ご協力お願いします

- 来年度以降の継続のためにもみなさんのフィードバックが重要です！

今後の問題作成の参考としてみなさんの解析手順をみたいので、解析メモが共有可能であればリンク等で共有してもらえると幸いです。（フィードバックのためお願いします・・・）

回答を入力
