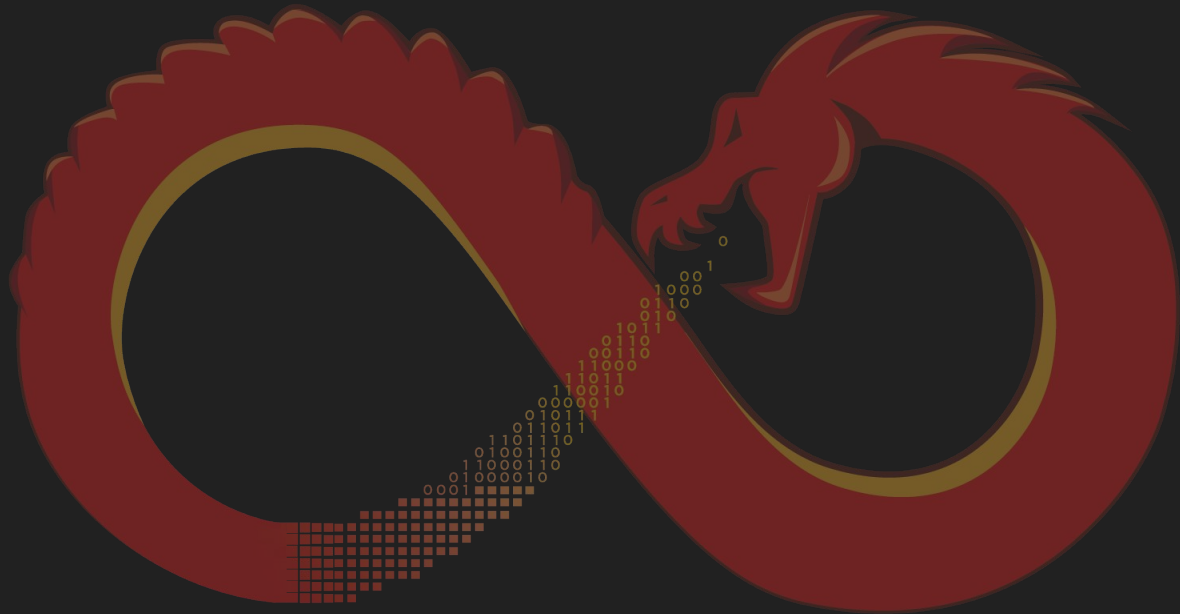


# MWS Cup 2022

## 課題2 解説



# 2022の問題担当

---

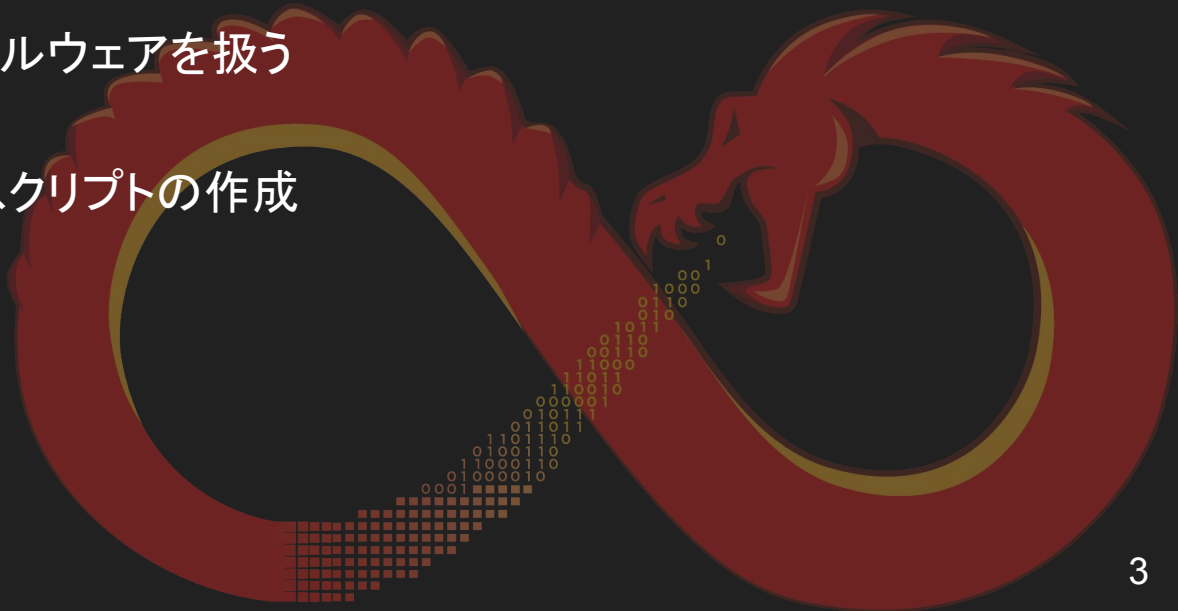
- 課題2主担当
  - 株式会社サイバーディフェンス研究所 中島 将太
- 問題作成委員
  - 株式会社 エヌ・エフ・ラボラトリーズ 皆川 諒
  - 株式会社サイバーディフェンス研究所 森 瑞穂
  - 学生、若手募集中！



# 課題2のテーマ

---

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



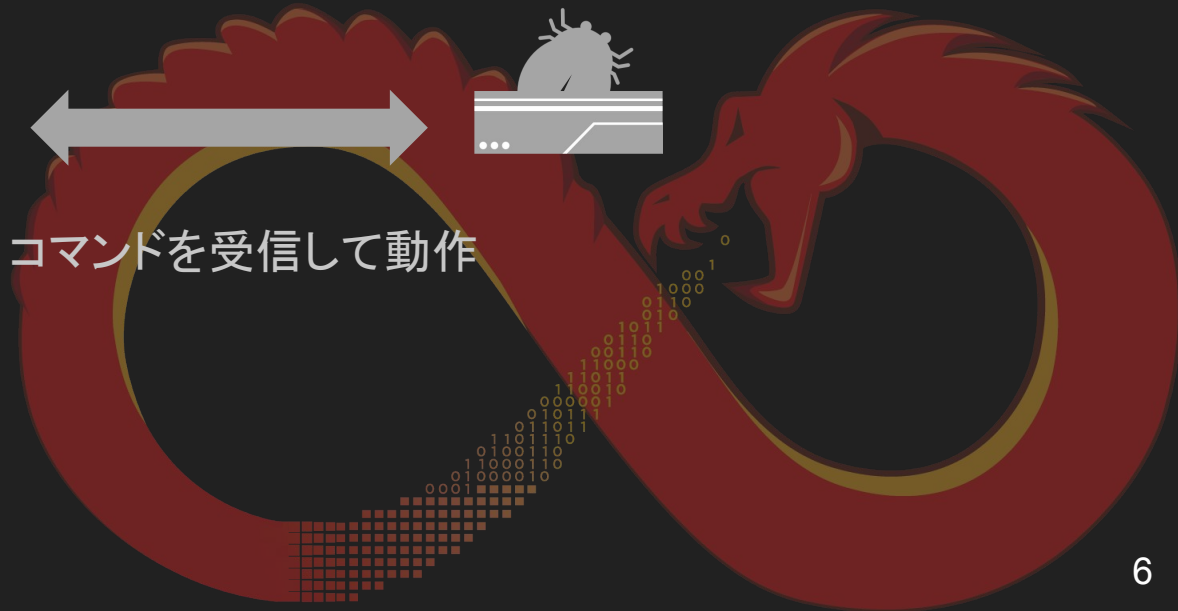
# ポイント

---

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



# マルウェアの動作概要



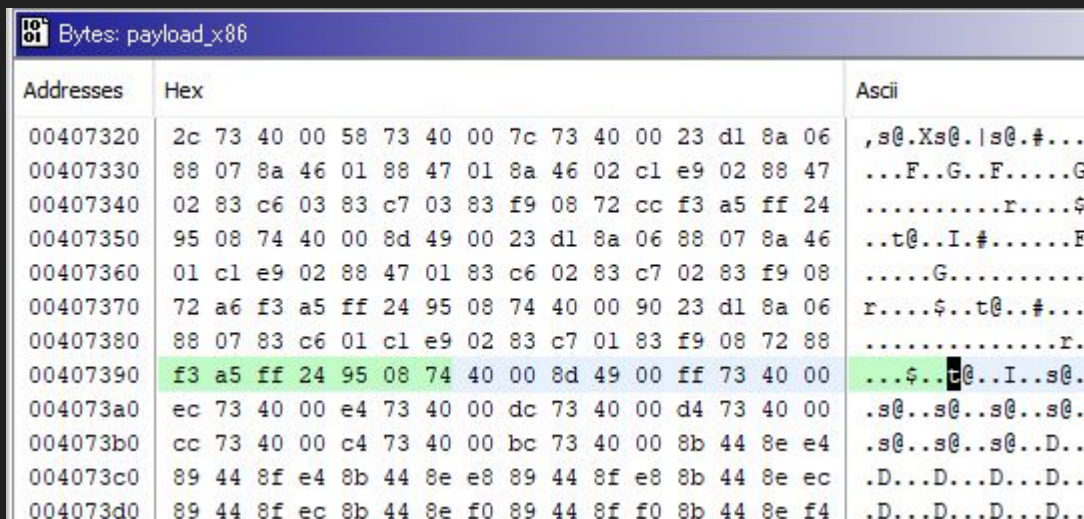
# Ghidra Tips: Go To

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



# Ghidra Tips: データのコピー1

- Bytesウィンドウでコピーする

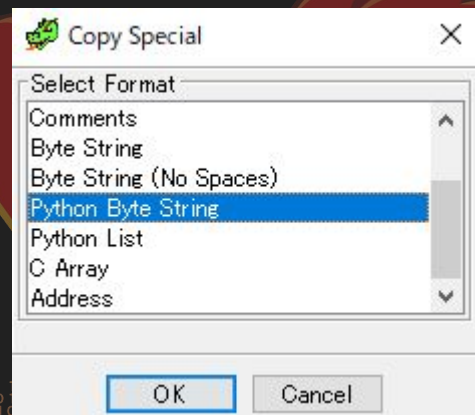
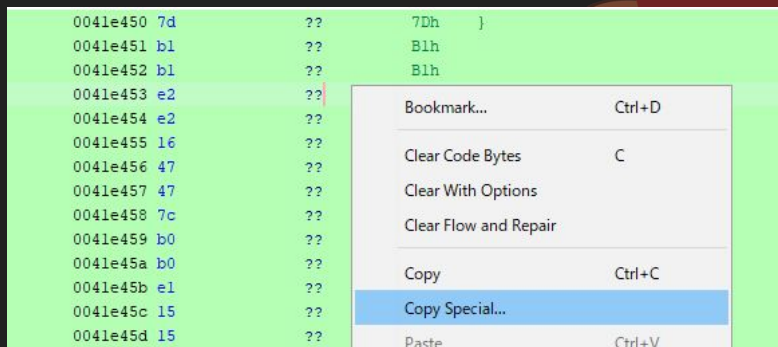


Addresses	Hex	Ascii
00407320	2c 73 40 00 58 73 40 00 7c 73 40 00 23 d1 8a 06	,s@.Xs@. s@.#...
00407330	88 07 8a 46 01 88 47 01 8a 46 02 c1 e9 02 88 47	...F..G..F.....G
00407340	02 83 c6 03 83 c7 03 83 f9 08 72 cc f3 a5 ff 24	.....r....\$
00407350	95 08 74 40 00 8d 49 00 23 d1 8a 06 88 07 8a 46	..t@..I.#.....F
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..



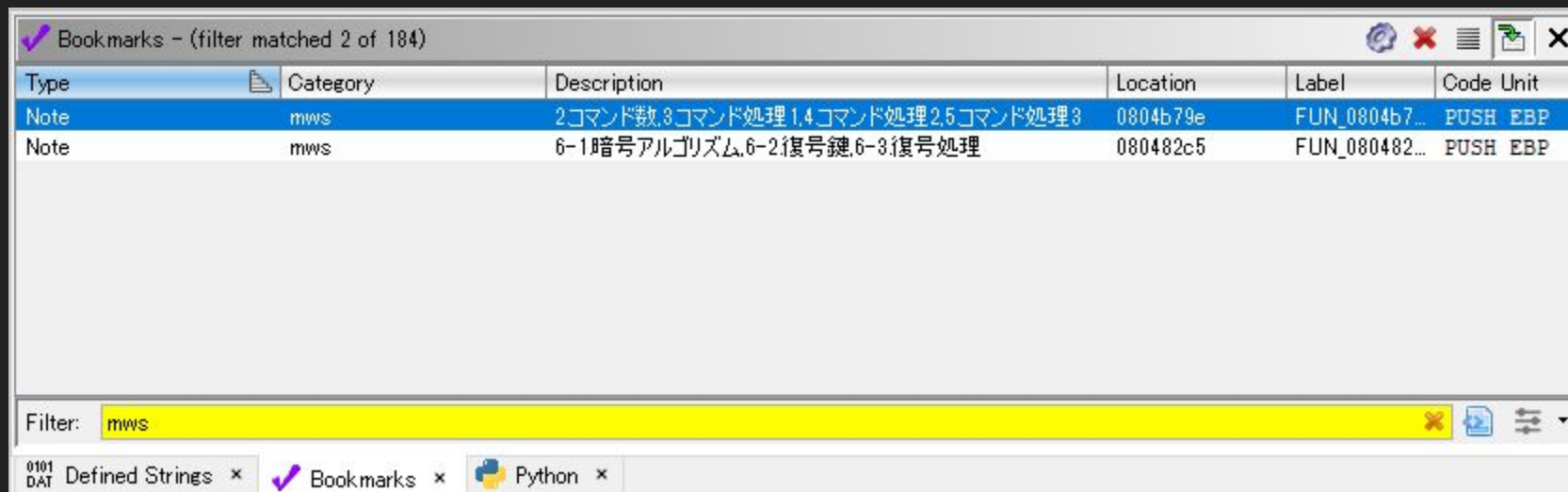
# Ghidra Tips: データのコピー2

- 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



# Bookmarks

問題に関連するアドレスをBookmarkとして登録済み



Bookmarks - (filter matched 2 of 184)

Type	Category	Description	Location	Label	Code Unit
Note	mws	2コマンド数,3コマンド処理,4コマンド処理,5コマンド処理,3	0804b79e	FUN_0804b7...	PUSH EBP
Note	mws	6-1暗号アルゴリズム,6-2復号鍵,6-3復号処理	080482c5	FUN_080482...	PUSH EBP

Filter: mws

0101 DAT Defined Strings x ✓ Bookmarks x Python x

# Hint!!

swi(software interrupt)がある場合は、Listingウィンドウを見る

```
Decompile: FUN_080635d0 - (sample)
1
2 uint FUN_080635d0(void)
3
4 {
5     code *pcVar1;
6     uint uVar2;
7     int *in_GS_OFFSET;
8
9     pcVar1 = (code *)swi(0x80);
10    uVar2 = (*pcVar1)();
```

FUN_080635d0			XREF[2]:	FUN_08049e6
080635d0	89 da	MOV	EDX,EBX	
080635d2	8b 4c 24 08	MOV	ECX,dword ptr [ESP + param_2]	
080635d6	8b 5c 24 04	MOV	EBX,dword ptr [ESP + param_1]	
080635da	b8 27 00 ...	MOV	EAX,0x27	
080635df	cd 80	INT	0x80	
080635e1	89 d3	MOV	EBX,EDX	
080635e3	3d 01 f0 ...	CMP	EAX,0xfffff001	
080635e8	0f 83 52 ...	JNC	FUN_08053c40	
080635ee	c3	RET		

# Hint!!

main()関数の前には何があるのか (7)

## Linuxカーネルに見る、システムコール番号と引数、システムコール・ラッパーとは

(1/2 ページ)

C言語の「Hello World!」プログラムで使われる、「printf()」「main()」関数の中身を、デバッガによる解析と逆アセンブル、ソースコード読解などのさまざまな側面から探る連載。前回から、printf()内のwrite()やint \$0x80の呼び出しについてLinuxカーネルのソースコード側から探ってきた。今回は、さらにシステムコールについて学ぶ。

© 2017年06月29日 05時00分 公開

【坂井弘亮, 著】



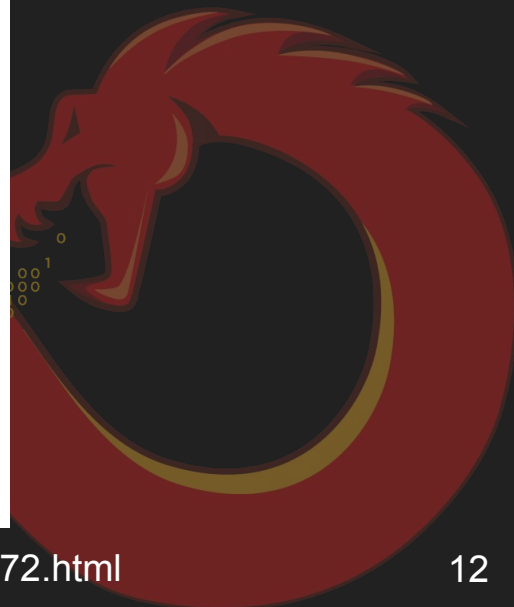
インデックス ●●● 連載目次



書籍の中から有用な技術情報をピックアップして紹介する本シリーズ。今回は、秀和システム発行の書籍『ハロー“Hello, World” OSと標準ライブラリのシグトとしくみ (2015年9月11日発行)』からの抜粋です。

ご注意：本稿は、著者及び出版社の許可を得て、そのまま転載したものです。このため用語用語の統一ルールなどは@ITのそれとは一致しません。あらかじめご了承ください。

※編集部注：前回事「Linuxカーネルのソースコードを読んで、システムコールを探る」はこちら



# Hint!!

## システムコール番号の参考資料

project logo Chromium OS Docs

[Home](#) [Sitemap](#) [Getting Started](#) [Developer Guide](#) [Contact](#) [Bugs](#) [Glossary](#) [Style Guide \(Markdown\)](#) [Gitiles \(Markdown Format\)](#) [Old Docs](#)

### Linux System Call Table

**Warning: This document is old & has moved. Please update any links:**  
<https://chromium.googlesource.com/chromiumos/docs/+HEAD/constants/syscalls.md>

These are the system call numbers (NR) and their corresponding symbolic names.  
These vary significantly across architectures/ABIs, both in mappings and in actual name.  
This is a quick reference for people debugging things (e.g. seccomp failures).  
For more details on syscalls in general, see the `syscall(2)` man page.

#### Contents

- Random Names
- Kernel Implementations
- Calling Conventions
- Tables
  - x86\_64 (64-bit)
  - arm (32-bit/EABI)
  - arm64 (64-bit)
  - x86 (32-bit)
  - Cross-arch Numbers

x86 (32-bit)

Compiled from Linux 4.14.0 headers.

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%es)
0	restart_syscall	<a href="#">man/ cs/</a>	0x00	-	-	-	-
1	exit	<a href="#">man/ cs/</a>	0x01	int error_code	-	-	-
2	fork	<a href="#">man/ cs/</a>	0x02	-	-	-	-
3	read	<a href="#">man/ cs/</a>	0x03	unsigned int fd	char *buf	size_t count	-
4	write	<a href="#">man/ cs/</a>	0x04	unsigned int fd	const char *buf	size_t count	-
5	open	<a href="#">man/ cs/</a>	0x05	const char *filename	int flags	umode_t mode	-
6	close	<a href="#">man/ cs/</a>	0x06	unsigned int fd	-	-	-
7	waitpid	<a href="#">man/ cs/</a>	0x07	pid_t pid	int *stat_addr	int options	-
8	creat	<a href="#">man/ cs/</a>	0x08	const char *pathname	umode_t mode	-	-
9	link	<a href="#">man/ cs/</a>	0x09	const char *oldname	const char *newname	-	-
10	unlink	<a href="#">man/ cs/</a>	0x0a	const char *pathname	-	-	-
11	execve	<a href="#">man/ cs/</a>	0x0b	const char *filename	const char *const *argv	const char *const *envp	-
12	chdir	<a href="#">man/ cs/</a>	0x0c	const char *filename	-	-	-

[https://chromium.googlesource.com/chromiumos/docs/+master/constants/syscalls.md#x86-32\\_bit](https://chromium.googlesource.com/chromiumos/docs/+master/constants/syscalls.md#x86-32_bit)

競技中はここまで



# 1-1. コンフィグサイズ

Challenge ×

## 1-1. コンフィグサイズ 2

課題2はマルウェアの静的解析に関する問題です。問題ファイル `sample.gzf` をダウンロードし、Ghidraで解析して回答してください。

このマルウェアは複数のIPアドレスとポートをコンフィグとして持つことができる。1つのコンフィグのサイズを10進数で答えよ。

 `sample.gzf`

# 1-1. コンフィグサイズ

Defined StringsウィンドウのIPアドレスを起点に解析を進める

The image shows a debugger interface with two main windows. The left window displays a memory dump for address 080d4098, showing a series of bytes (00, 0a, 00, 00, 00, 00, 00, 00) and their corresponding hex values. The right window shows the 'Defined Strings' window, which lists 695 items. The string '172.16.123.133' is highlighted in blue, indicating it is the current selection.

Location	String Value	String Representation
080d3019	zR	zR
080d30d1		""
080d3185	zR	"zR"
080d31f1		""
080d32c9	zR	"zR"
080d3309	zR	"zR"
080d337d		""
080d350d	zR	"zR"
080d35a5	zR	"zR"
080d37c9		""
080d37fd	zR	"zR"
080d3941	zR	"zR"
080d39b5		""
080d3a01	zR	"zR"
080d3b11		""
080d3b7d	zR	"zR"
080d3bf9	zR	"zR"
080d3ccd	zR	"zR"
080d40a0	172.16.123.133	"172.16.123.133"



# 1-1. コンフィグサイズ

- FUN\_0804851の引数に利用
- FUN\_0804851内のループで操作

```
63 do {
64     local_24 = FUN_08048515(s_172.16.123.133_080d40a0);
65     if (local_24 != -1) {
28     for (; DAT_080d4b00 < 10; DAT_080d4b00 = DAT_080d4b00 + 1) {
29         if ((*char *) (DAT_080d4b00 * 0x3e + param_1) == '\0') ||
30             (*(short *) (DAT_080d4b00 * 0x3e + param_1 + 0x3c) == 0) {
31             DAT_080d4b00 = 0;
32             if (0 < local_lc) {
33                 FUN_0804ed10(local_lc);
34             }
35             return -1;

```

# 1-1. コンフィグサイズ

- 引数で受け取ったアドレスを0x3e(62)毎に進める

```
27 local_lc = FUN_080e4ba0(2,2,0);
28 for (; DAT_080d4b00 < 10; DAT_080d4b00 = DAT_080d4b00 + 1) {
29     if ((* (char *) (DAT_080d4b00 + 0x3e) param_1) == '\0') ||
30         (*(short *) (DAT_080d4b00 + 0x3e) param_1 + 0x3c) == 0) {
31         DAT_080d4b00 = 0;
32         if (0 < local_lc) {
33             FUN_0804ed10(local_lc);
34         }
35         return -1;

```

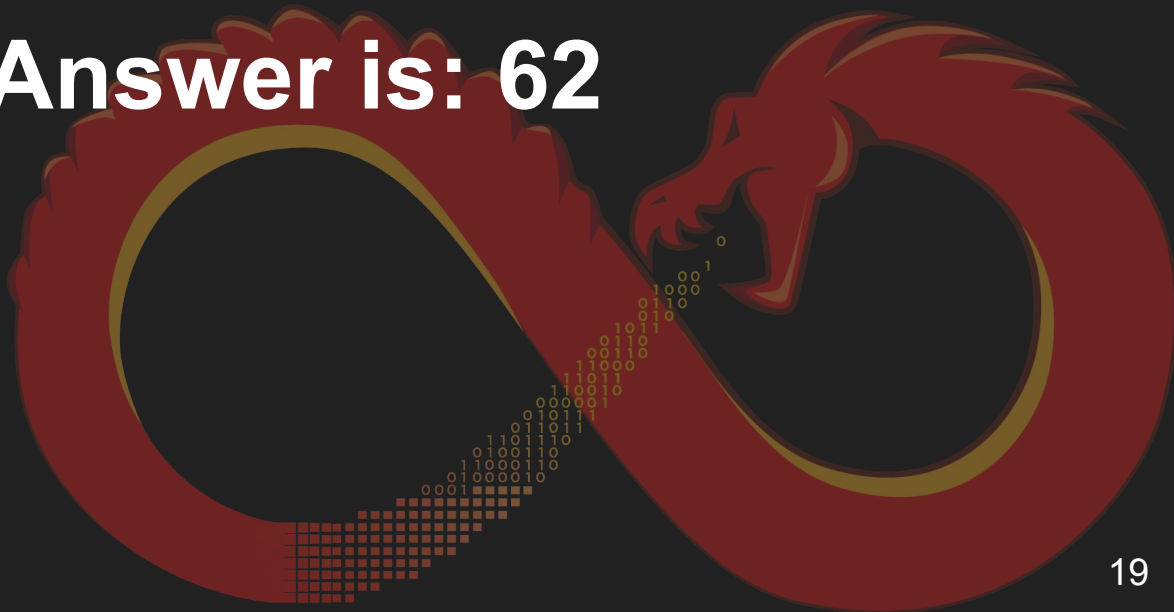
62

172.16.123.133

# 1-1. コンフィグサイズ

---

The Answer is: 62



# 1-2. コンフィグ数

Challenge ×

## 1-2. コンフィグ数

### 2

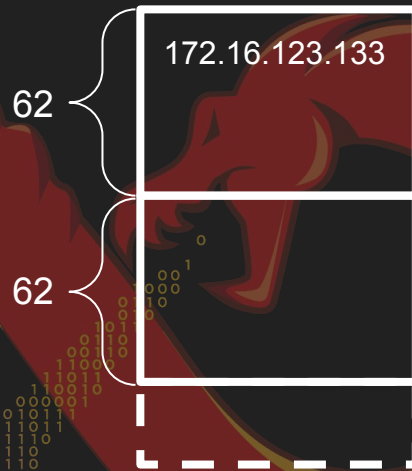
このマルウェアが保有できるIPアドレスとポートの組の最大数を10進数で答えよ。

## 1-2. コンフィグ数

- 10回ループする
  - 10組のIPアドレスとポート情報を保持可能

```
27 FUN_0804ed10(local_1c, param_1, param_2, param_3, param_4, param_5, param_6, param_7, param_8, param_9, param_10, param_11, param_12, param_13, param_14, param_15, param_16, param_17, param_18, param_19, param_20, param_21, param_22, param_23, param_24, param_25, param_26, param_27, param_28, param_29, param_30, param_31, param_32, param_33, param_34, param_35, param_36, param_37, param_38, param_39, param_40, param_41, param_42, param_43, param_44, param_45, param_46, param_47, param_48, param_49, param_50, param_51, param_52, param_53, param_54, param_55, param_56, param_57, param_58, param_59, param_60, param_61, param_62, param_63, param_64, param_65, param_66, param_67, param_68, param_69, param_70, param_71, param_72, param_73, param_74, param_75, param_76, param_77, param_78, param_79, param_80, param_81, param_82, param_83, param_84, param_85, param_86, param_87, param_88, param_89, param_90, param_91, param_92, param_93, param_94, param_95, param_96, param_97, param_98, param_99, param_100);
28 for (; DAT_080d4b00 < 10; DAT_080d4b00 = DAT_080d4b00 + 1) {
29     if ((* (char *) (DAT_080d4b00 * 0x3e + param_1) == '\0') ||
30         (*(short *) (DAT_080d4b00 * 0x3e + param_1 + 0x3c) == 0)) {
31         DAT_080d4b00 = 0;
32         if (0 < local_1c) {
33             FUN_0804ed10(local_1c);
34         }
35         return -1;

```



## 1-2. コンフィグ数

---

The Answer is: 10



# 1-3.通信先

Challenge ×

## 1-3.通信先

### 3

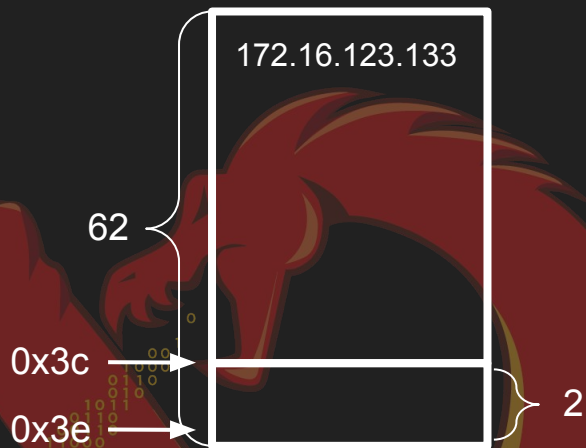
本検体の通信先(C2サーバ)を"IPアドレス:ポート番号"の形式で答えよ。複数存在する場合はカンマ(,)区切りで連続させて、IPアドレスの第4オクテットが小さい順に並べること。

例: 192.168.2.2:80,192.168.2.150:8080

# 1-3.通信先

1つのコンフィグは0x3cと0x3eの2つに分割される

```
28 for (; DAT_080d4b00 < 10; DAT_080d4b00 = DAT_080d4b00 + 1) {
29     if ((* (char *) (DAT_080d4b00 * 0x3e + param_1) == '\0') ||
30         (*(short *) (DAT_080d4b00 * 0x3e + param_1 + 0x3c) == 0)) {
31         DAT_080d4b00 = 0;
32         if (0 < local_1c) {
33             FUN_0804ed10(local_1c);
34         }
35     }
```





# 1-3.通信先

- 0x1BB → 443

```
ip_addr XREF[1]: FUN_0804bb96:0804bc98 (*)
080d40a0 31 37 32 ... ??[60]
├── 080d40a0 [0] 31h 1,37h 7,32h 2,2Eh .
├── 080d40a4 [4] 31h 1,36h 6,2Eh .,31h 1
├── 080d40a8 [8] 32h 2,33h 3,2Eh .,31h 1
├── 080d40ac [12] 33h 3,33h 3, 00h, 00h
├── 080d40b0 [16] 00h, 00h, 00h, 00h
├── 080d40b4 [20] 00h, 00h, 00h, 00h
├── 080d40b8 [24] 00h, 00h, 00h, 00h
├── 080d40bc [28] 00h, 00h, 00h, 00h
├── 080d40c0 [32] 00h, 00h, 00h, 00h
├── 080d40c4 [36] 00h, 00h, 00h, 00h
├── 080d40c8 [40] 00h, 00h, 00h, 00h
├── 080d40cc [44] 00h, 00h, 00h, 00h
├── 080d40d0 [48] 00h, 00h, 00h, 00h
├── 080d40d4 [52] 00h, 00h, 00h, 00h
├── 080d40d8 [56] 00h, 00h, 00h, 00h
└── 080d40dc bb 01 dw 1BBh
```

# 1-3.通信先

---

The Answer is:  
172.16.123.133:443



## 2. コマンド数

Challenge ×

## 2. コマンド数

### 2

`FUN_0804b79e` はC2サーバから受信したコマンドをもとに処理を決定する関数である。コマンド数を答えよ。

## 2. コマンド数

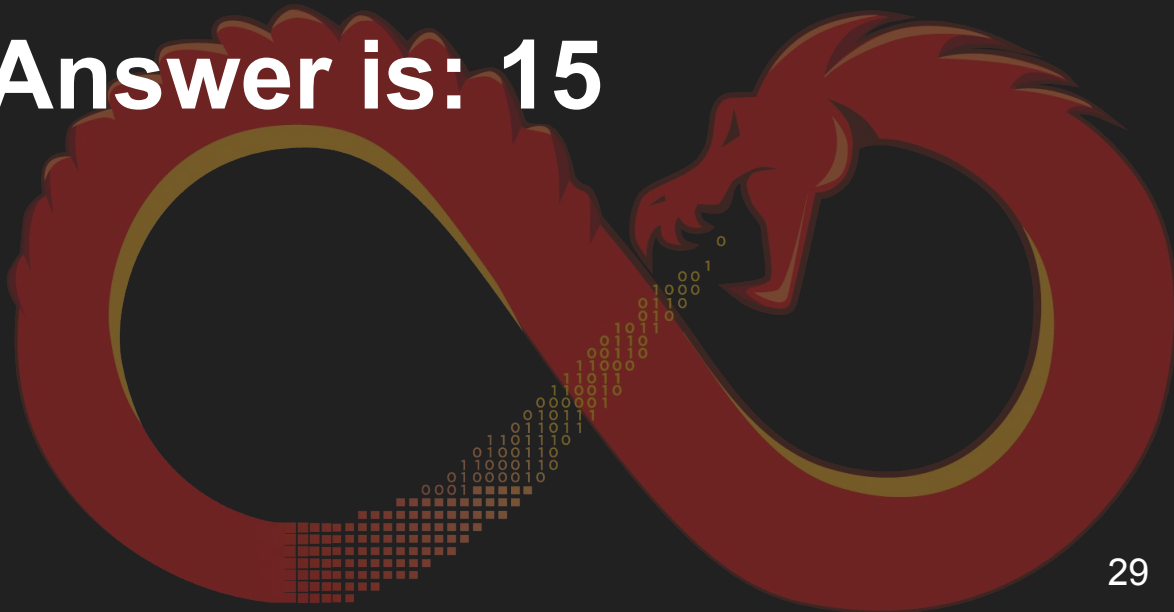
FUN\_0804b79e内のコマンドIDを使った  
分岐処理を数える

```
Decompile: FUN_0804b79e - (sample)
4 {
5   undefined4 uVar1;
6   undefined4 local_c;
7   byte cmd_id;
8
9   if ((param_1 == 0) || (param_1 == -1)) {
10    local_c = 0;
11  }
12  else if (param_2 == (byte *)0x0) {
13    local_c = 0;
14  }
15  else if (param_3 < 1) {
16    local_c = 0;
17  }
18  else {
19    cmd_id = *param_2;
20    if (cmd_id == 0x89) {
21      local_c = FUN_08049e65(param_1,param_2,param_3);
22    }
23    else {
24      if (cmd_id < 0x8a) {
25        if (cmd_id == 0x84) {
26          uVar1 = FUN_080493e5(param_1,param_2,param_3);
27          return uVar1;
28        }
29        if (cmd_id < 0x85) {
30          if (cmd_id == 0x82) {
31            uVar1 = FUN_08048e74(param_1,param_2,param_3);
32            return uVar1;
33          }
34          if (0x82 < cmd_id) {
```

## 2. コマンド数

---

The Answer is: 15



# 3.コマンド処理1

Challenge ×

## 3.コマンド処理1

### 2

コマンド0x85の処理はどれか。以下から選択せよ。

※1度しか回答できません。注意して回答してください。

- ファイル読み込み
- ファイル書き込み
- ファイル削除
- ファイルの名前変更

0/1 attempt



# 3. コマンド処理1

コマンド分岐が難しい。2つの重要なシステムコールを用いる

```
43     else {
44         if (bVar1 == 0x86) {
45             uVar2 = FUN_08049747(param_1,param_2,param_3);
46             return uVar2;
47         }
48         if (bVar1 == 0x87) {
49             uVar2 = FUN_08049626(param_1,param_2,param_3);
50             return uVar2;
51         }
52         if (bVar1 == 0x87) {
53             uVar2 = FUN_08049c69(param_1,param_2,param_3);
54             return uVar2;
55         }
56     }
```



```
undefined FUN_08049626(int param_1,int param_2,int param_3)
{
    undefined uVar1;
    undefined local_e [2];
    int fd;
    int local_8;

    if ((param_1 == -1) || (param_1 == 0)) {
        uVar1 = 0;
    }
    else if (param_2 == 0) {
        uVar1 = 0;
    }
    else if (param_3 < 1) {
        uVar1 = 0;
    }
    else {
        fd = open(sDAT_080d8780,0x442,0x1f8);
        if (fd != -1) {
            uVar1 = 0;
        }
        else {
            FUN_0804_000(param_2,param_3);
            local_8 = write(fd,param_2 + 1,param_3 + -1);
            FUN_0804_000(param_2,param_3);
        }
    }
}
```

# 3.コマンド処理1

0804efca	53	PUSH	EBX
0804efcb	8b 54 24 10	MOV	EDX,dword ptr [ESP + param_3]
0804efcf	8b 4c 24 0c	MOV	ECX,dword ptr [ESP + param_2]
0804efd3	8b 5c 24 08	MOV	EBX,dword ptr [ESP + param_1]
0804efd7	b8 05 00	MOV	EAX,0x5
	00 00		
0804efdc	cd 80	INT	0x80
0804efde	5b	POP	EBX
0804efdf	3d 01 f0	CMP	EAX,0xfffff001
	ff ff		
0804efe4	0f 83 56	JNC	FUN_08053c40
	4c 00 00		
0804efea	c3	RET	

```
2 uint open(void)
3
4 {
5     code *pcVar1;
6     uint uVar2;
7     undefined4 uVar3;
8     int *in_GS_OFFSET;
9
10    if (in_GS_OFFSET[3] == 0) {
11        pcVar1 = (code *)swi(0x80);
12        uVar2 = (*pcVar1)();
13        if (uVar2 < 0xfffff001) {
14            return uVar2;
15        }
16    }
17    else {
18        uVar3 = FUN_0804e8e0();
19        pcVar1 = (code *)swi(0x80);
20        uVar2 = (*pcVar1)(uVar3);
```

swi関数。中身はopenシステムコール



# 3.コマンド処理1

```
0804ec63 8b 5c 24 08    MOV     EBX,dword ptr [ESP + param_1]
0804ec67 b8 04 00        MOV     EAX,0x4
0804ec6c cd 80          INT     0x80
0804ec6e 5b            POP     EBX
0804ec6f 3d 01 f0       CMP     EAX,0xfffff001
0804ec74 0f 83 c6       JNC     FUN_08053c40
0804ec7a c3            RET

                                LAB_0804ec7b
0804ec7b e8 60 fc       CALL   FUN_0804e8e0
0804ec80 50            PUSH   EAX
0804ec81 53            PUSH   EBX
0804ec82 8b 54 24 14    MOV     EDX,dword ptr [ESP + param_3]
0804ec86 8b 4c 24 10    MOV     ECX,dword ptr [ESP + param_2]
0804ec8a 8b 5c 24 0c    MOV     EBX,dword ptr [ESP + param_1]
0804ec8e b8 04 00        MOV     EAX,0x4
0804ec93 cd 80          INT     0x80
```

```
1
2 uint write(void)
3
4 {
5     code *pcVar1;
6     uint uVar2;
7     undefined4 uVar3;
8     int *in_GS_OFFSET;
9
10    if (in_GS_OFFSET[3] == 0) {
11        pcVar1 = (code *)swi(0x80);
12        uVar2 = (*pcVar1)();
13        if (uVar2 < 0xfffff001) {
14            return uVar2;
15        }
16    }
17    else {
18        uVar3 = FUN_0804e8e0();
19        pcVar1 = (code *)swi(0x80);
20        uVar2 = (*pcVar1)(uVar3);
21        FUN_0804e8a0();
22        if (uVar2 < 0xfffff001) {
23            return uVar2;
24        }
25    }
26 }
```

swi関数。writeシステムコールを呼ぶ。

## 3.コマンド処理1

---

The Answer is: ファイル書き込み

# 4.コマンド処理2



## 4. コマンド処理2: コマンドID 0xF6の動作を説明せよ

```
0804af0b c7 04 24 MOV dword ptr [ESP]=>local_102c,s/_b
          46 85 0b 08
0804af12 e8 f9 79 CALL FUN_08062910
          01 00
0804af17 85 c0 TEST EAX,EAX
0804af19 79 0f JNS LAB_0804af2a
0804af1b c7 85 ec MOV dword ptr [EBP + local_1018],0x0
```

```
47 iVar1 = FUN_080553b0(PTR_DAT_080d47fc,0,1,0);
48 if (iVar1 == 0) {
49     FUN_08063630(&DAT_080b8544);
50     iVar1 = FUN_08062910("/bin/sh", "/bin/sh", 0);
51     if (1 < iVar1) {
52         return local_101c;
53     }
54     local_1018 = 0;
```

```
08087750 55 PUSH EBP
08087751 89 e5 MOV EBP,ESP
08087753 8b 4d 0c MOV ECX,dword ptr [EBP + param_2]
08087756 53 PUSH EBX
08087757 8b 55 10 MOV EDX,dword ptr [EBP + param_3]
0808775a 8b 5d 08 MOV EBX,dword ptr [EBP + param_1]
0808775d b8 0b 00 MOV EAX,0xb
          00 00
08087762 cd 80 INT 0x80
```

```
7 int *in_GS_OFFSET;
8
9 pcVar1 = (code *)swi(0x80);
10 uVar2 = (*pcVar1)();
11 if (0xfffff000 < uVar2) {
12     *(uint *)(&in_GS_OFFSET + -0x30) = -uVar2;
13     uVar2 = 0xffffffff;
14 }
15 return uVar2;
```

/bin/shという文字列からシェルを開くことを想定。  
システムコールはexecveである。

## 4. コマンド処理2: コマンドID 0xF6の動作を説明せよ

```
0804af0b c7 04 24 MOV dword ptr [ESP]=>local_102c,s_/b
          46 85 0b 08
0804af12 e8 f9 79 CALL FUN_08062910
          01 00
0804af17 85 c0 TEST EAX,EAX
0804af19 79 0f JNS LAB_0804af2a
0804af1b c7 85 ec MOV dword ptr [EBP + local_1018],0x0
```

```
47 iVar1 = FUN_080553b0(PTR_DAT_080d47fc,0,1,0);
48 if (iVar1 == 0) {
49     FUN_08063630(&DAT_080b8544);
50     iVar1 = FUN_08062910("/bin/sh", "/bin/sh", 0);
51     if (1 < iVar1) {
52         return local_101c;
53     }
54     local_1018 = 0;
```

```
08087750 55 PUSH EBP
08087751 58 PUSH EBX
08087752 59 PUSH ECX
08087753 5a PUSH EDI
08087754 5b PUSH ESI
08087755 5c PUSH EBP
08087756 5d PUSH EBX
08087757 5e PUSH ECX
08087758 5f PUSH EDI
08087759 60 PUSH ESI
0808775a 8b 5d 08 MOV EBX, dword ptr [EBP + param_1]
0808775d b8 0b 00 MOV EAX, 0xb
          00 00
08087762 cd 80 INT 0x80
```

offset	syscall	arg1	arg2	arg3	arg4	arg5
11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp

```
12 *(uint *)(&in_GS_OFFSET + -0x30) = -uVar2;
13 uVar2 = 0xffffffff;
14 }
15 return uVar2;
```

/bin/shという文字列からシェルを開くことを想定。  
システムコールはexecveである。

## 4. コマンド処理2: コマンドID 0xF6の動作を説明せよ

---

The Answer is:

侵害されたマシンのリモートシェルを開く

```
execve("/bin/sh", "/bin/sh", [0]);
```

# 5.コマンド処理3

Challenge ×

## 5.コマンド処理3

### 2

ファイル名の変更をするコマンドIDを16進数で答えよ。

例えば、コマンドIDが0x1234ABCDだと判明した場合、**0x1234ABCD**の形式で回答せよ。

※回答回数が制限されています。注意して回答してください。

0/3 attempts

# 5.コマンド処理3

基本方針:

RAT のメイン関数 FUN\_0804b79e に  
実装されているコマンドを全て読み、  
最も適当なコマンドIDを探す

```
2 undefined4 FUN_0804b79e(int param_1,byte *param_2,int param_3)
3
4 {
5     byte bVar1;
6     undefined4 uVar2;
7     undefined4 local_c;
8
9     if ((param_1 == 0) || (param_1 == -1)) {
10         local_c = 0;
11     }
12     else if (param_2 == (byte *)0x0) {
13         local_c = 0;
14     }
15     else if (param_3 < 1) {
16         local_c = 0;
17     }
18     else {
19         bVar1 = *param_2;
20         if (bVar1 == 0x89) {
21             local_c = FUN_08049e65(param_1,param_2,param_3);
22         }
23         else {
24             if (bVar1 < 0x8a) {
25                 if (bVar1 == 0x84) {
26                     uVar2 = FUN_080493e5(param_1,param_2,param_3);
27                     return uVar2;
28                 }
29                 if (bVar1 < 0x85) {
30                     if (bVar1 == 0x82) {
31                         uVar2 = FUN_08048e74(param_1,param_2,param_3);
32                         return uVar2;
33                     }
34                     if (0x82 < bVar1) {
35                         uVar2 = FUN_08048ef8(param_1,param_2,param_3);
36                         return uVar2;
37                     }
38                 }
39             }
40         }
41     }
42 }
```



# 5.コマンド処理3

FUN\_0804a456 で使用されている system call “38” が最もそれらしい

```
call_rename XREF[2]: FUN_0804a456:0804a598
08055220 89 da MOV EDX,EBX
08055222 8b 4c 24 08 MOV ECX,dword ptr [E
08055226 8b 5c 24 04 MOV EBX,dword ptr [E
0805522a b8 26 00 ... MOV EAX,38
0805522f cd 80 INT 0x80
08055231 89 d3 MOV EBX,EDX
08055233 3d 01 f0 ... CMP EAX,0xffffffff
08055238 0f 83 02 ... JNC FUN_08053c40
0805523e c3 RET
0805523f 90 ?? 90h
```

38	rename	man/ cs/	0x26
----	--------	----------	------

# 5.コマンド処理3

あとは FUN\_0804a456 に分岐するIDの条件を参照から辿れば答え

```
else {  
    if (bVar1 == 0xc6) {  
        return 2;  
    }  
    if (bVar1 < 199) {  
        if (bVar1 == 0x8b) {  
            uVar2 = FUN_0804a2fa(param_1,param_2,param_3);  
            return uVar2;  
        }  
        if (bVar1 < 0x8b) {  
            uVar2 = FUN_08049fca(param_1,param_2,param_3);  
            return uVar2;  
        }  
        if (bVar1 == 0x8f) {  
            uVar2 = FUN_0804a456(param_1,param_2,param_3);  
            return uVar2;  
        }  
    }  
}
```



# 5.コマンド処理3

## RAT コマンド処理一覧

コマンドID	処理内容
0x15	ランダムデータ送信
0x82	検体固有のIDを送信
0x83	ファイル一覧情報取得
0x84	ファイルの属性情報取得
0x85	ファイル書き込み
0x86	ファイル読み込み
0x87	ファイルディスクリプタのクローズ
0x89	新規ディレクトリの作成

コマンドID	処理内容
0x8a	ファイルの削除
0x8b	ディレクトリの削除
0x8f	ファイル名の変更
0xc6	プロセスキル
0xf6	リモートシェルの起動
0xf7	リモートシェルにコマンドを送信
0xf8	リモートシェルの終了

## 5.コマンド処理3

---

The Answer is: 0x8f



# 6-1.暗号アルゴリズム

## 6-1.暗号アルゴリズム

2

FUN\_080482c5 はマルウェアが通信の暗号化のために使用する関数である。

暗号のベースとなっているアルゴリズムを以下から選択せよ。

※1度しか回答できません。注意して回答してください。

- RC4
- xor
- AES
- XXTEA

0/1 attempt



# 6-1.暗号アルゴリズム

## 毎年恒例の暗号アルゴリズム特定問題です

### 2020年: AES

#### ラウンド処理部分を探す

- sbx が使われている関数の回りを探すと、004017b2 が AES のメイン処理に該当
- ラウンド数は 10

```
2 void __cdecl AES_round(undefined4 param_1)
3 {
4 {
5     byte bVar1;
6     byte *unaff_ESI;
7
8     aa_AddRoundKey(param_1);
9     bVar1 = i;
10    do {
11        aa_SubBytes(unaff_ESI);
12        aa_ShiftRows();
13        aa_MixColumns((int)unaff_ESI);
14        aa_AddRoundKey(param_1);
15        bVar1 = bVar1 + 1;
16    } while (bVar1 < 10);
17    aa_SubBytes(unaff_ESI);
18    aa_ShiftRows();
19    aa_AddRoundKey();
20    return;
21 }
```

MWS Cup 2020

34

### 2021年: XOR

#### 2-1.エンコード [loader\_x86] Answer

- XORでデコードするループ

```
35 i = 0;
36 j = 0;
37 do {
38     k = key + 1;
39     i = i + 1;
40     (sdec_str)[j] = *k ^ enc_str[j];
41     if (i == 0xf) {
42         __DAT_0041c210 = iVar2 + iVar1;
43         i = 0;
44     }
45     j = j + 1;
46 } while (j < 0xb);
47 pRVar3 = GetModuleHandleA(sdec_str);
```

MWS Cup 2021

14

# 6-1.暗号アルゴリズム

- 指定された関数を読み、暗号アルゴリズムの特徴と一致する部分がないかを探す
- 冒頭に  $0x100$  (= 256) バイトの配列を作り、 $0 \sim 256$ の値で初期化する処理が入っている
  - さらにこの配列は、初期化処理後関数の第3引数の値を使いながら値がスワップされる

```
Decompile: FUN_080482c5 - (bifrose_mws)
1
2 void FUN_080482c5(int param_1,int param_2,int param_3,int param_4,byte param_5)
3
4 {
5     byte abStack532 [256];
6     byte abStack276 [256];
7     int local_14;
8     uint local_10;
9     uint local_c;
10    byte local_7;
11    byte local_6;
12    byte local_5;
13
14    for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + 1) {
15        abStack276[local_14] = (byte)local_14;
16    }
17    for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + param_4) {
18        for (local_10 = 0; ((int)local_10 < param_4 && ((int)(local_10 + local_14)
19            local_10 = local_10 + 1) {
20            abStack532[local_14 + local_10] = *(byte *) (local_10 + param_3);
21        }
22    }
23    local_10 = 0;
24    for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + 1) {
25        local_5 = abStack276[local_14];
26        local_6 = abStack532[local_14];
27        local_10 = local_5 + local_10 + (uint)abStack532[local_14] & 0xff;
28        local_6 = abStack276[local_10];
29        abStack276[local_14] = abStack276[local_10];
```

# 6-1.暗号アルゴリズム

## RC4 のアルゴリズムをググるとその特徴と合致してますね

### RC4と特定するための特徴

ぱっと見でRC4だと特定するための特徴と思われるものをまとめてみます。

- 256バイトのSという状態を持つ
- 最初に鍵を元にKSAによってSの状態が初期化される（初回のみ呼ばれる）
- Sの状態変化はswapで行われる
- データの暗号化はPRGAから出力された1バイト値をXORによって暗号化する
- KSAとPRGAの計算は似ている（けど鍵の計算がないので少し違う）
- 暗号化と復号が同じ処理（送信と受信で同じ関数を使っている場合）

『アセンブリで書かれたRC4は見た瞬間分かるか』, yasulib memo,  
<https://yasulib.hatenablog.jp/entry/20180127/1517041821>



## 6-1.暗号アルゴリズム

---

The Answer is: RC4



## 6-2.復号鍵

### 6-2.復号鍵

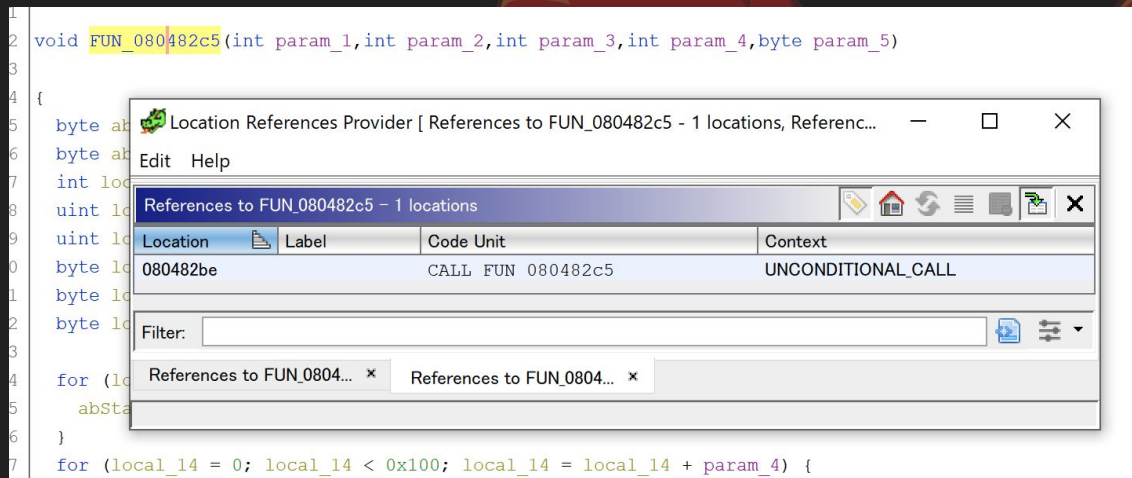
#### 3

マルウェアが暗号化で使用する鍵を16進数で答えよ。  
例えば、鍵が文字列“ABCD”だった場合、  
`\x41\x42\x43\x44` の形式で回答せよ。

## 6-2.復号鍵

- 6-1で暗号アルゴリズムがRC4だとわかれば、この関数の第3引数として使われたデータが鍵である可能性が高い
- 暗号関数の参照を辿る。使用されているのはアドレス **0x080482be**

```
1 void FUN_080482c5(int param_1,int param_2,int param_3,int param_4,byte param_5)
2
3
4 {
5   byte abSt
6   byte abSt
7   int loc_10
8   uint loc_10
9   uint loc_10
10  byte loc_10
11  byte loc_10
12  byte loc_10
13
14  for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + param_4) {
15    abSt
16  }
17  for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + param_4) {
```



Location	Label	Code Unit	Context
080482be		CALL FUN_080482c5	UNCONDITIONAL_CALL

## 6-2.復号鍵

- アドレス 0x080482be へいくと FUN\_08048294 というwrapper関数らしき関数へ飛ぶ。
- FUN\_080482c5 の第3引数は FUN\_08048294 の第1引数として使われているので、次は FUN\_08048294 の参照を辿り第1引数として使われている値を探す。

```
2 void FUN_08048294(undefined4 param_1,undefined4 param_2,undefined4 param_3)
3
4 {
5     FUN_080482c5(param_2,param_3,param_1,0x10,0x16);
6     return;
7 }
8
```

Location	Label	Code Unit	Context
08048b38		CALL FUN 08048294	UNCONDITIONAL_CALL
0804bd98		CALL FUN 08048294	UNCONDITIONAL_CALL

## 6-2.復号鍵

- 参照されているアドレスのいずれかを確認すると、FUN\_08048294 の第1引数には \xa3 ~ \x00 までの16 バイトが使用されていることがわかる。

→ これがRC4で使用される 鍵

```
else {
    local_4c = 0xa3;
    local_4b = 0x78;
    local_4a = 0x26;
    local_49 = 0x35;
    local_48 = 0x57;
    local_47 = 0x32;
    local_46 = 0x2d;
    local_45 = 0x60;
    local_44 = 0xb4;
    local_43 = 0x3c;
    local_42 = 0x2a;
    local_41 = 0x5e;
    local_40 = 0x33;
    local_3f = 0x34;
    local_3e = 0x72;
    local_3d = 0;
    piVar2 = (int *)FUN_0805d000(param_3);
    if (piVar2 == (int *)0x0) {
        local_e4 = 0;
    }
    else {
        local_3c = piVar2;
        if (DAT_080d4b1c == 1) {
            iVar3 = FUN_08048256(param_3);
            *piVar2 = iVar3;
        }
        else if (DAT_080d4b1c == 0) {
            *piVar2 = param_3;
        }
        piVar2 = local_3c + 1;
        for (iVar3 = param_3; iVar3 != 0; iVar3 = iVar3 + -1) {
            *(undefined *)piVar2 = *param_2;
            param_2 = param_2 + 1;
            piVar2 = (int *)((int)piVar2 + 1);
        }
        FUN_08048294(6local_4c,local_3c + 1,param_3);
    }
}
```

References to FUN\_08048294 - 2 locations [CodeBrowser]

Edit Help

References to FUN\_08048294 - 2 locations

Location	Label	Code Unit
08048b38		CALL FUN_08048294
0804bd98		CALL FUN_08048294

Filter:

## 6-2.復号鍵

### TIPs: 鍵の長さの予想

- 0x00はNull終端であって鍵として使用していないのではないかと考えた人がいるかもしれないので補足
- **FUN\_080482c5** の第3引数が鍵の本体ならば、**第4引数には鍵の長さが使われているのではないかと予想ができる** (これは、**FUN\_080482c5** を解析しても確認できる)
- 本検体の場合、第4引数は 0x10 (= 16)固定。そのため、**鍵としては \xa3 から始まる16バイト分が正解**

```
2 void FUN_08048294(undefined4 param_1,undefined4 param_2,undefined4 param_3)
3
4 {
5     FUN_080482c5(param_2,param_3,param_1,0x10,0x16);
6     return;
7 }
```



## 6-2.復号鍵

---

The Answer is:

`\xa3\x78\x26\x35\x57\x32\x2d\x60\xb4\x3c\x2a\x5e\x33\x34\x72\x00`



## 6-3.復号処理

### 6-3.復号処理

#### 4

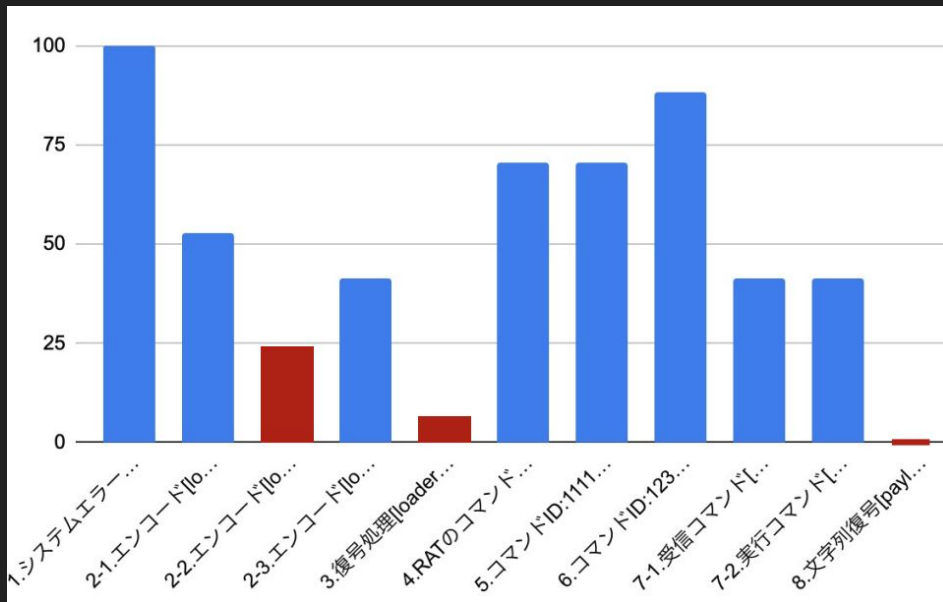
FUN\_080482c5 のアルゴリズムで暗号化されたファイルを `encrypted.bin` として取得した。  
解析結果をもとにファイルを復号し、得られた平文を答えよ。

なお、ベースとなっている暗号アルゴリズムのコードは `encrypt_hint.py` として実装されている。必要なチームは本コードをベースに復号コードを書いてもよい。

📄 encrypted.bin

📄 encrypt\_hint....

## 6-3.復号処理



去年は復号問題解けたチーム **0** だったので、  
今年には復号に使えるベースのコードを用意しました!!

## 6-3.復号処理

基本方針: 用意されたコード(RC4)とデコンパイル結果を比較して処理を追加する

```
def encrypt(data: bytes, key: bytes) -> bytearray:
    result: list[int] = []
    S: list[int] = [0] * 0x100
    for i in range(0, 0x100, 1):
        S[i] = i
    j: int = 0
    for i in range(0, 0x100, 1):
        S_i_tmp: int = S[i]
        k: int = key[i % len(key)]
        j = (S_i_tmp + j + k) & 0xFF
        S[i] = S[j]
        S[j] = S_i_tmp

    j: int = 0
    for i in range(0, len(data), 1):
        S_i_tmp: int = S[(i + 1) & 0xFF]
        j = (S[(i + 1) & 0xFF] + j) & 0xFF
        S[(i + 1) & 0xFF] = S[j]
        S[j] = S_i_tmp
        index: int = (S[(i + 1) & 0xFF] + S[j]) & 0xFF
        xor_key: int = S[index]
        calc: int = data[i] ^ xor_key
        result.append(calc)

    return bytearray(result)
```

```
for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + 1) {
    abStack276[local_14] = (byte)local_14;
}
for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + param_4) {
    for (local_10 = 0; ((int)local_10 < param_4 && ((int)(local_10 + local_14) < 0x100));
        local_10 = local_10 + 1) {
        abStack532[local_14 + local_10] = *(byte *) (local_10 + param_3);
    }
}
local_10 = 0;
for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + 1) {
    local_5 = abStack276[local_14];
    local_6 = abStack532[local_14];
    local_10 = local_5 + local_10 + (uint)abStack532[local_14] & 0xff;
    local_6 = abStack276[local_10];
    abStack276[local_14] = abStack276[local_10];
    abStack276[local_10] = local_5;
}
local_c = (uint)param_5;
local_10 = 0;
for (local_14 = 0; local_14 < param_2; local_14 = local_14 + 1) {
    local_7 = abStack276[local_14 + 1U & 0xff];
    local_10 = abStack276[local_14 + 1U & 0xff] + local_10 & 0xff;
    abStack276[local_14 + 1U & 0xff] = abStack276[local_10];
    abStack276[local_10] = local_7;
    local_6 = abStack276[local_14 + 1U & 0xff] + local_7;
    local_7 = abStack276[(byte)(abStack276[local_14 + 1U & 0xff] + local_7)];
    if ((local_c & 0x80) == 0) {
        *(char *) (local_14 + param_1) = *(char *) (local_14 + param_1) + (char)local_c;
        *(byte *) (local_14 + param_1) = *(byte *) (local_14 + param_1) ^ local_7;
    }
    else {
        local_7 = local_7 ^ *(byte *) (local_14 + param_1);
        *(byte *) (local_14 + param_1) = (char)local_c + local_7;
    }
}
return;
```

## 6-3.復号処理

### 変更点1: 256 バイトの S-box が 2つ使用されている

```
def encrypt(data: bytes, key: bytes) -> bytearray:
    result: list[int] = []
    S: list[int] = [0] * 0x100
    for i in range(0, 0x100, 1):
        S[i] = i
    j: int = 0
    for i in range(0, 0x100, 1):
        S_i_tmp: int = S[i]
        k: int = key[i % len(key)]
        j = (S_i_tmp + j + k) & 0xFF
        S[i] = S[j]
        S[j] = S_i_tmp

    j: int = 0
    for i in range(0, len(data), 1):
        S_i_tmp: int = S[(i + 1) & 0xFF]
        j = (S[(i + 1) & 0xFF] + j) & 0xFF
        S[(i + 1) & 0xFF] = S[j]
        S[j] = S_i_tmp
        index: int = (S[(i + 1) & 0xFF] + S[j]) & 0xFF
        xor_key: int = S[index]
        calc: int = data[i] ^ xor_key
        result.append(calc)

    return bytearray(result)
```

```
for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + 1) {
    abStack276[local_14] = (byte)local_14;
}
for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + param_4) {
    for (local_10 = 0; ((int)local_10 < param_4 && ((int)(local_10 + local_14) < 0x100));
        local_10 = local_10 + 1) {
        abStack532[local_14 + local_10] = *(byte *) (local_10 + param_3);
    }
}
local_10 = 0;
for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + 1) {
    local_5 = abStack276[local_14];
    local_6 = abStack532[local_14];
    local_10 = local_5 + local_10 + (uint)abStack532[local_14] & 0xff;
    local_6 = abStack276[local_10];
    abStack276[local_14] = abStack276[local_10];
    abStack276[local_10] = local_5;
}
local_c = (uint)param_5;
local_10 = 0;
for (local_14 = 0; local_14 < param_2; local_14 = local_14 + 1) {
    local_7 = abStack276[local_14 + 10 & 0xff];
    local_10 = abStack276[local_14 + 10 & 0xff] + local_10 & 0xff;
    abStack276[local_14 + 10 & 0xff] = abStack276[local_10];
    abStack276[local_10] = local_7;
    if (local_10 < local_7) {
        *(char)local_c;
        *(char)local_7;
    }
    else {
        local_7 = local_7 ^ *(byte *) (local_14 + param_1);
        *(byte *) (local_14 + param_1) = (char)local_c + local_7;
    }
}
return;
```

KSA による S-box の初期化で  
独自のアルゴリズムで作成した  
S-box も追加で使用している

## 6-3.復号処理

### 変更点2: XOR の処理後に第5引数の値だけ加算する処理がある

```
def encrypt(data: bytes, key: bytes) -> bytearray:
    result: list[int] = []
    S: list[int] = [0] * 0x100
    for i in range(0, 0x100, 1):
        S[i] = i
    j: int = 0
    for i in range(0, 0x100, 1):
        S_i_tmp: int = S[i]
        k: int = key[i % len(key)]
        j = (S_i_tmp + j + k) & 0xFF
        S[i] = S[j]
        S[j] = S_i_tmp

    j: int = 0
    for i in range(0, len(data), 1):
        S_i_tmp: int = S[(i + 1) & 0xFF]
        j = (S[(i + 1) & 0xFF] + j) & 0xFF
        S[(i + 1) & 0xFF] = S[j]
        S[j] = S_i_tmp
        index: int = (S[(i + 1) & 0xFF] + S[j]) & 0xFF
        xor_key: int = S[index]
        calc: int = data[i] ^ xor_key
        result.append(calc)

    return bytearray(result)
```

```
for (local_14 = 0; local_14 < 0x100; local_14 = local_14 + 1) {
    abStack276[local_14] = (byte)local_14;
}

for (local_14 = 0; local_14 < param_4; local_14 = local_14 + param_4) {
    local_7 = abStack276[local_14 + 1U & 0xff];
    local_10 = abStack276[local_14 + 1U & 0xff] + local_10 & 0xff;
    abStack276[local_14 + 1U & 0xff] = abStack276[local_10];
    abStack276[local_10] = local_7;
    local_6 = abStack276[local_14 + 1U & 0xff] + local_7;
    local_7 = abStack276[(byte)(abStack276[local_14 + 1U & 0xff] + local_7)];
    if ((local_c & 0x80) == 0) {
        *(char *) (local_14 + param_1) = *(char *) (local_14 + param_1) + (char)local_c;
        *(byte *) (local_14 + param_1) = *(byte *) (local_14 + param_1) ^ local_7;
    }
    else {
        local_7 = local_7 ^ *(byte *) (local_14 + param_1);
        *(byte *) (local_14 + param_1) = (char)local_c + local_7;
    }
}

return;
```

第5引数の値が  
- 0x80 (=128) 未満で "加算 => XOR"  
- 0x80 (=128) 以上で "XOR => 加算"  
RC4にシーザー暗号のような処理  
が追加されている

## 6-3.復号処理

第5引数で 사용되는値を確認

```
2 void FUN_08048294(undefined4 param_1,undefined4 param_2,undefined4 param_3)
3
4 {
5     FUN_080482c5(param_2,param_3,param_1,0x10,0x16);
6     return;
7 }
```

0x16 (= 22) なので 加算 => XOR

## 6-3.復号処理

解析結果を元に、コードを加筆・修正する

- 修正点1: S-box の追加
- 修正点2: KSA 処理の変更
- 修正点3: **減算処理**の追加
  - 暗号化コードが”加算”なので  
”減算”処理をしないと復号できない点に注意

# 6-3.復号処理

## 実装例

```
def RC4_custom(data: bytes, key: bytes, shift: int) -> bytearray:
    result: list[int] = []
    # KSA
    S: list[int] = [0] * 0x100
    S_extend: list[int] = [0] * 0x100
    for i in range(0, 0x100, 1):
        S[i] = i
    for i in range(0, 0x100, 16):
        for i2 in range(0, 16, 1):
            if i + i2 > 0x100:
                break
            S_extend[i + i2] = key[i2]
    j: int = 0
    for i in range(0, 0x100, 1):
        S_tmp: int = S[i]
        S_extend_tmp: int = S_extend[i]
        j = (S_tmp + S_extend_tmp + j) & 0xFF
        S[i] = S[j]
        S[j] = S_tmp

    # PRGA
    j: int = 0
    for i in range(0, len(data), 1):
        S_i_tmp: int = S[(i + 1) & 0xFF]
        j = (S[(i + 1) & 0xFF] + j) & 0xFF
        S[(i + 1) & 0xFF] = S[j]
        S[j] = S_i_tmp
        index: int = (S[(i + 1) & 0xFF] + S_i_tmp) & 0xFF
        xor_key: int = S[index]
        calc: int = 0
        if shift & 0x80 == 0:
            calc = ((data[i] ^ xor_key) - shift) & 0xFF
        else:
            calc = ((data[i] - shift) ^ xor_key) & 0xFF
        result.append(calc)

    return bytearray(result)
```

S-box 追加

KSA の変更

減算処理の追加



## 6-3.復号処理

### 実行結果

```
~/Documents/work via 🐍 v3.9.0  
> python bifrose_rc4.py encrypted.bin  
bytearray(b'MWS{w3lc0m3_t0_d33p_1n_th3_rc4_c1ph3r!!}')  

```

```
def RC4_custom(data: bytes, key: bytes, shift: int) -> bytearray:  
    result: list[int] = []  
    # KSA  
    S: list[int] = [0] * 0x100  
    S_extend: list[int] = [0] * 0x100  
    for i in range(0, 0x100, 1):  
        S[i] = i  
    for i in range(0, 0x100, 16):  
        for i2 in range(0, 16, 1):  
            if i + i2 > 0x100:  
                break  
            S_extend[i + i2] = key[i2] S-box 追加  
    j: int = 0  
    for i in range(0, 0x100, 1):  
        S_tmp: int = S[i] KSA の変更  
        S_extend_tmp: int = S_extend[i]  
        j = (S_tmp + S_extend_tmp + j) & 0xFF  
        S[i] = S[j]  
        S[j] = S_tmp  
  
    # PRGA  
    j: int = 0  
    for i in range(0, len(data), 1):  
        S_i_tmp: int = S[(i + 1) & 0xFF]  
        j = (S[(i + 1) & 0xFF] + j) & 0xFF  
        S[(i + 1) & 0xFF] = S[j]  
        S[j] = S_i_tmp  
        index: int = (S[(i + 1) & 0xFF] + S_i_tmp) & 0xFF  
        xor_key: int = S[index] 減算処理の追加  
        calc: int = 0  
        if shift & 0x80 == 0:  
            calc = ((data[i] ^ xor_key) - shift) & 0xFF  
        else:  
            calc = ((data[i] - shift) ^ xor_key) & 0xFF  
        result.append(calc)  
  
    return bytearray(result)
```

## 6-3.復号処理

---

The Answer is:

MWS{W3lc0m3\_t0\_d33p\_1n\_th3\_rc4\_c1ph3r!!}

# 解答まとめ

1-1. コンフィグサイズ このマルウェアは複数のPアドレスとポートをコンフィグとして持てる1つのコンフィグのサイズを求めよ	62
1-2. コンフィグ数 このマルウェアが保有できるPアドレスとポートの組の最大数(0進数)で答えよ。	10
1-3. 通信先: 本検体の通信先(C2サーバ)を"IPアドレス:ポート番号"の形式で答えよ。	172.16.123.133:443
2. コマンド数 FUN_0804b79e はC2サーバから受信したコマンドをもとに処理を決定する関数である。コマンド数を答えよ。	15
3. コマンド処理1: コマンド0x85の処理を選択せよ。	ファイル書き込み
4. コマンド処理2: コマンドID 0xF6 の動作を説明せよ。	侵害されたマシンのリモートシェル起動
5. コマンド処理3: ファイル名の変更をするコマンドIDを16進数で答えよ。	0x8f
6-1. 暗号アルゴリズム FUN_080482c5 はマルウェアが通信の暗号化のために使用する関数であるベースのアルゴリズムを選択せよ。	C2 サーバから受信したデータを別スレッドで実行する
6-2. 復号鍵: マルウェアが暗号化で使用する鍵を6進数で答えよ。	<code>\xa3\x78\x26\x35\x57\x32\x2d\x60\xb4\x3c\x2a\x5e\x33\x34\x72\x00</code>
6-3. 復号処理: FUN_080482c5 のアルゴリズムで暗号化されたファイルを復号し、得られた平文を答えよ。	<code>.MWS{W3lc0m3_t0_d33p_1n_th3_rc4_c1ph3r!!}</code>

# マルウェアファミリー

- ELF bifrose
  - BlackTechが利用
- 2020年以降Linux環境も狙う

## おわりに

攻撃グループBlackTechは、引き続き活動を続けており、今後も注意が必要です。今回解説した検体の通信先に関しては、Appendix Cに記載していますので、アクセスしている端末がないかご確認ください。なお、今回紹介したマルウェアが発見されたサーバー上では、その他のマルウェア（ダウンローダー、バックドア、ELF Bifrose）や、攻撃ツールを確認しています。次にサーバー上で保存されている攻撃ツールを列挙します。これらのツールは、攻撃グループBlackTechによって利用されている可能性があることにご注意ください。

<https://blogs.jpCERT.or.jp/ja/2021/09/gh0sttimes.html>

技術分析 RSS

## 中国のハッカーHUAPIのバックドア型マルウェア「BiFrost」の解析

2020.4.15 | Global Support & Service Share: [f](#) [in](#) [t](#)

キーワード: HUAPI, PLEAD, GhostCat, CVE-2020-1938, Linux, BiFrost, RC4, RAT

### はじめに

最近、TeamT5は台湾のある学術情報ネットワーク図書館のWebサイトでマルウェアが見つかったという情報を得ました。TeamT5の研究員は分析と調査により、同WebサイトのシステムがTomcat 7.0.73をWebサーバとして使用し、8009番ポートが開放されていることを発見しました。TeamT5の研究員は、WebサイトにGhostcat (CVE-2020-1938) の脆弱性があることを確認しました。詳細は下図のとおりです。

Nmap scan report for [redacted].edu.tw [redacted]

<https://teamt5.org/jp/posts/technical-analysis-on-backdoor-bifrost-of-the-chinese-apt-group-huapi/>

## 3.12. ELF\_Bifrose

Linux版のBifroseマルウェアです。過去の分析記事[14]に記載されている検体と大きく機能は変わっていませんが、私たちが確認した検体について紹介します。

### 3.12.1. 特徴

fileコマンドやreadelf -p .commentコマンドの実行結果は以下の通りです。古い環境でコンパイル・静的リンクされており、攻撃者は環境依存の問題を減らしたいものと考えられます。

```
$ file a914c729e4816fb9c8b9838694be385466c2cc366b71ab1410e84295cfa9946
a914c729e4816fb9c8b9838694be385466c2cc366b71ab1410e84295cfa9946: ELF 32-bit LSB executable, Intel 80386,
version 1 (SYSV), statically linked, for GNU/Linux 2.6.9, stripped
$ readelf -p .comment a914c729e4816fb9c8b9838694be385466c2cc366b71ab1410e84295cfa9946
```

```
String dump of section '.comment':
[ 1] GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-44)
[ 2] GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-44)
[ 3] gcc: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-44)
```

図 42 ELF\_Bifrose fileコマンド実行結果