

[Rev] Pack Program Solution

表層解析

問題のプログラム「challenge」がなんのファイルか調査する。

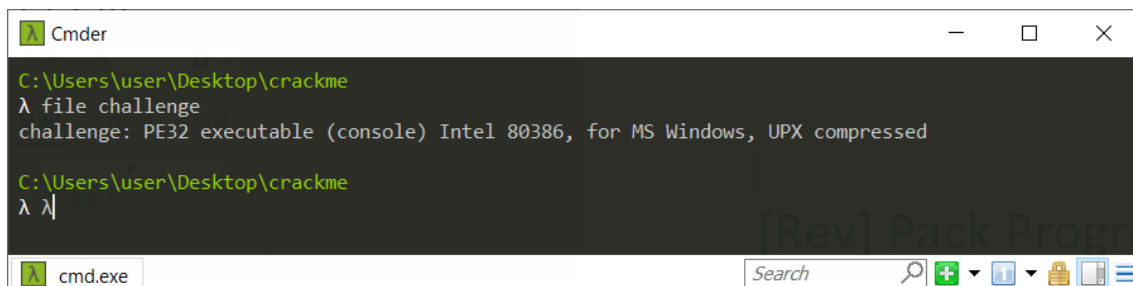


図 1 : file コマンドを使用した場合

「challenge」ファイルは Windows のバイナリファイル(x86)であることがわかる。試しに実行してみると次のようになる。

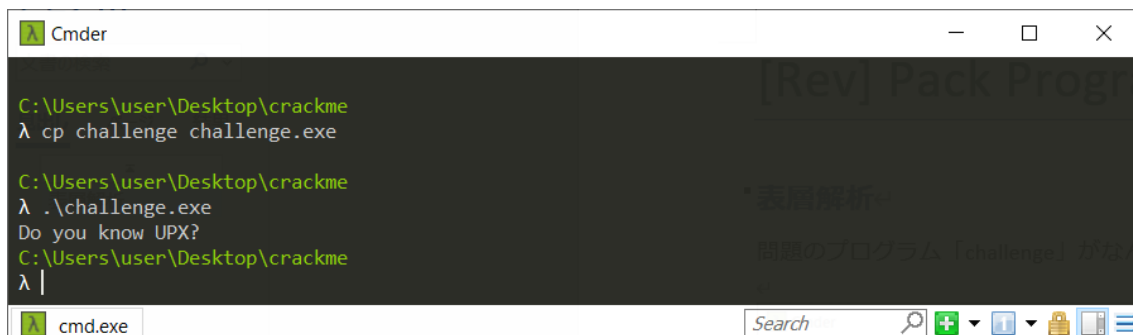


図 2 : challenge ファイルを実行した結果

「Do you know UPX?」と表示されるだけで、特に入力を要求されることもない。Strings コマンドで見ても、flag っぽい文字列はないので表示される UPX について調べる。

「UPX pack」等で調べると、大体次のような記事が見つかる。

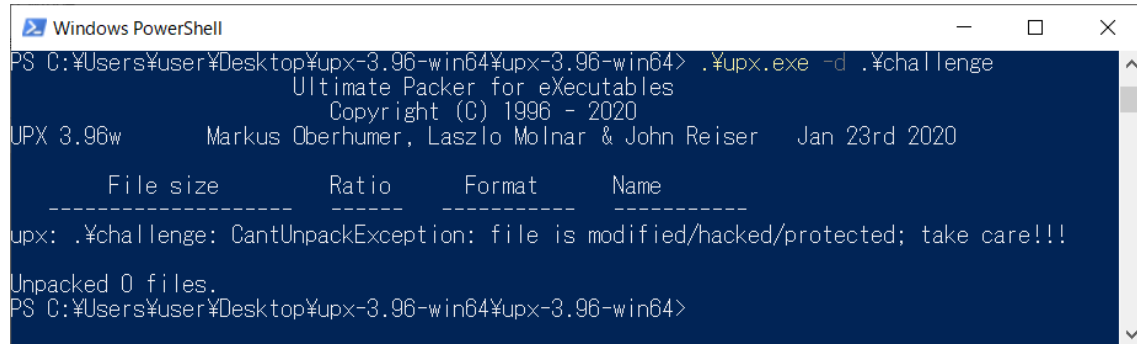
ESET : パッカー(Packer)

https://eset-info.canon-its.jp/malware_info/term/detail/00084.html

「file」コマンド時にも出力されていたが、このプログラムは UPX というパッカーでパックされている。

プログラムのアンパック

UPX はオープンソースなので、Github(<https://github.com/upx/upx/releases>)からダウンロードしてアンパックを試してみる。しかし、ファイルは修正されていると表示され、アンパックすることはできない。



```
Windows PowerShell
PS C:\Users\User\Desktop\upx-3.96-win64\upx-3.96-win64> .\upx.exe -d .\challenge
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2020
UPX 3.96w      Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

  File size      Ratio      Format      Name
  -----
upx: .\challenge: CantUnpackException: file is modified/hacked/protected; take care!!!

Unpacked 0 files.
PS C:\Users\User\Desktop\upx-3.96-win64\upx-3.96-win64>
```

図 3 : UPX の実行結果

そのため、今回はツールによるパックではなく、マニュアルアンパックを行う。調べるといろいろサイトでやり方が出てくる (<https://malware.news/t/the-basics-of-packed-malware-manually-unpacking-upx-executables/35961>)。ESP トリックを使う。

ESP トリックを使ったアンパック

Pushad 命令から 1 つ命令を進めたタイミングの ESP に対して、ハードウェアブレイクポイントを設定する。そのまま処理を継続させると popad 命令後で止まってくれる。

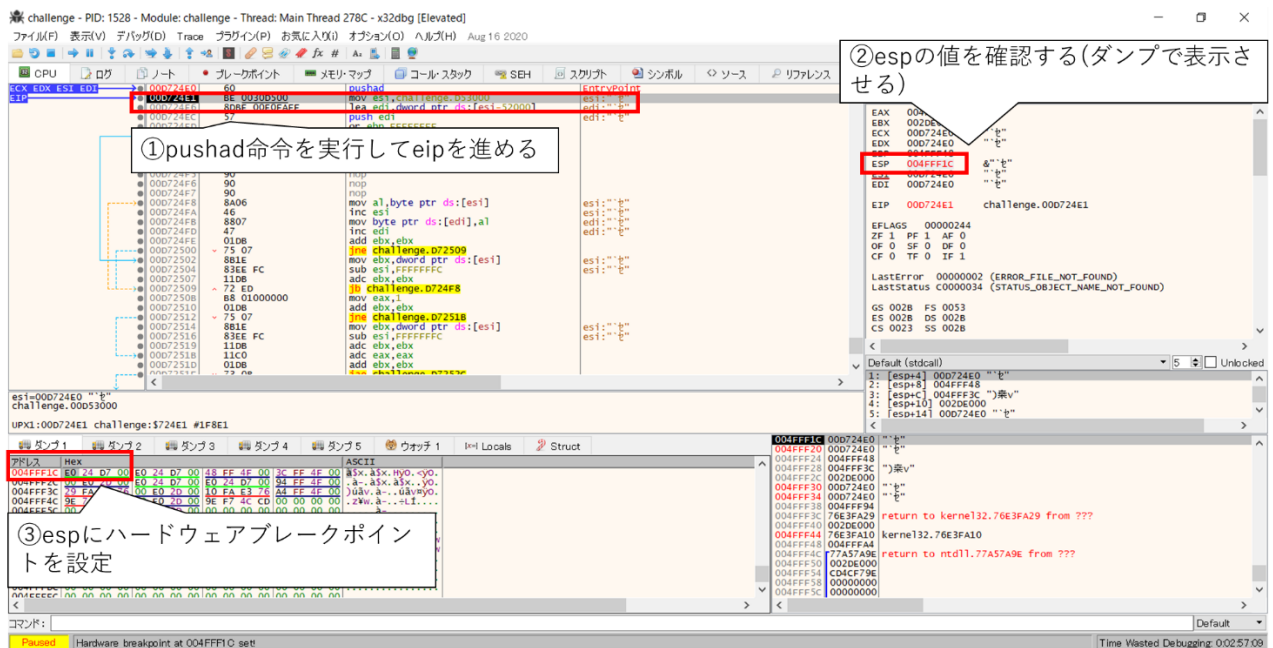
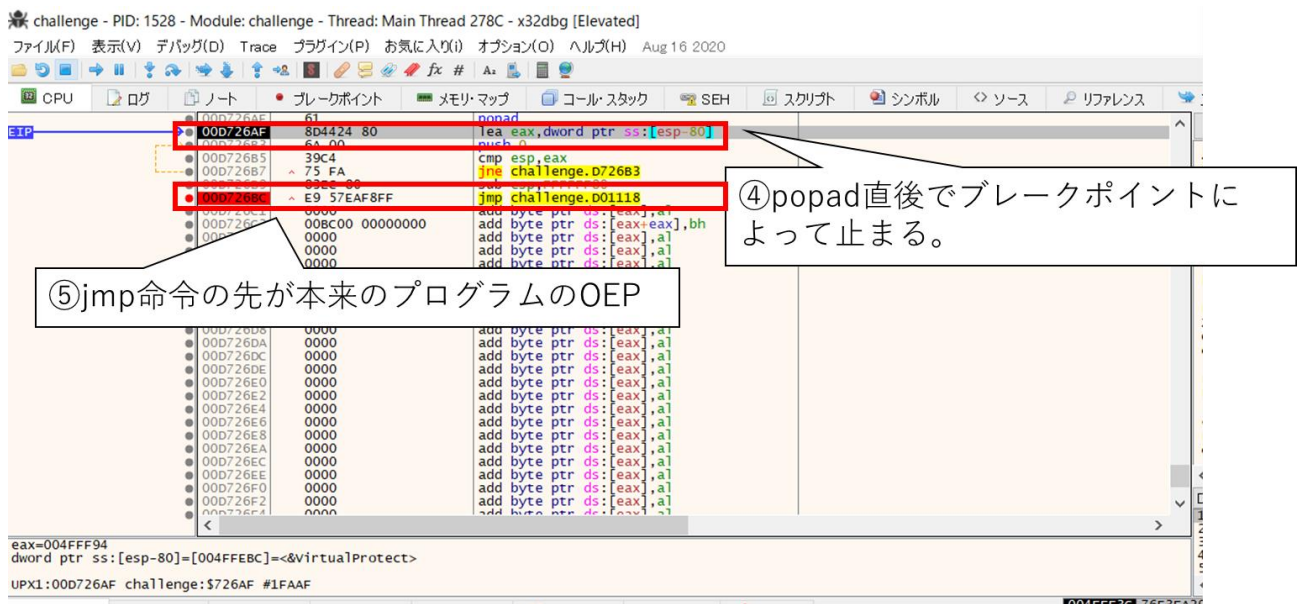
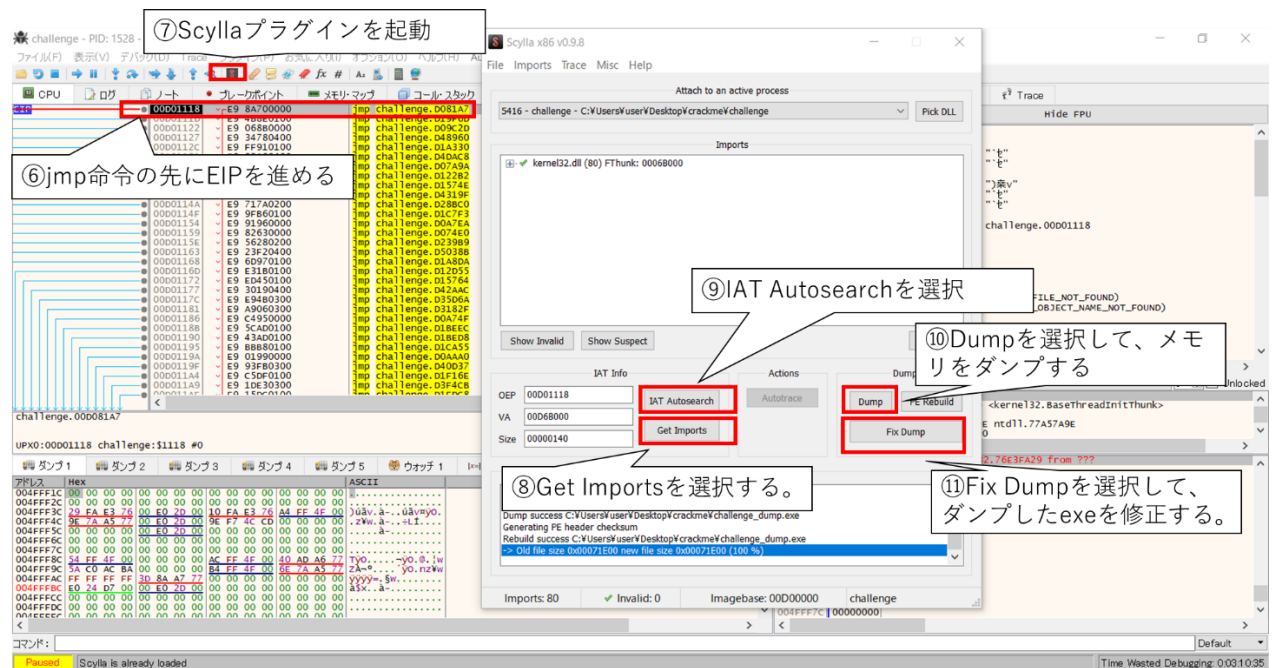


図 4 : PUSHAD 直後にハードウェアブレークポイントの設定



popad 命令後の jmp 命令によって遷移する先が本来のプログラムの OEP(Original Entry Point)となる。この後、jmp 命令を実行し EIP を jmp したところで止める。



Scylla プラグインを起動し、図のような手順で実行ファイルをダンプすることにより、Ghidra 等での静的解析が可能になる。

Ghidra を使った静的解析によるフラグ入手

ダンプしたファイルを Ghidra に読み込ませて、文字列とかを見てみると flag is %s という文字列が見つかる。

```
Decompile: FUN_00d076a0 - (challenge-dump_SCY.exe)
1
2 undefined4 FUN_00d076a0(void)
3
4 {
5     void *_Dst;
6     size_t sVar1;
7     char *pcVar2;
8     int iVar3;
9
10    _Dst = (void *)thunk_FUN_00d2b53d(0x40);
11    _memset(_Dst,0,0x10);
12    sVar1 = _strlen(s_6jTJ+b/RSJZxBLGcVcbglt==_00d6801c);
13    pcVar2 = (char *)thunk_FUN_00d070d0((int)s_6jTJ+b/RSJZxBLGcVcbglt==_00d6801c,sVar1);
14    iVar3 = thunk_FUN_00d07590(s_n96t6tPFEElhk0uSjcoeJasW_00d68000,pcVar2,(int)_Dst);
15    if (iVar3 == 0) {
16        thunk_FUN_00d07a20((wchar_t *)s_%.16s_00d68038);
17    }
18    else {
19        iVar3 = thunk_FUN_00d2b53d(0x78);
20        sVar1 = _strlen(s_5jqb5bvFEphcP4DHcvWM9rr46tVjjpxX_00d68040);
21        pcVar2 = (char *)thunk_FUN_00d070d0((int)s_5jqb5bvFEphcP4DHcvWM9rr46tVjjpxX_00d68040,sVar1);
22        thunk_FUN_00d07590(s_n96t6tPFEElhk0uSjcoeJasW_00d68000,pcVar2,iVar3);
23        thunk_FUN_00d07a20((wchar_t *)s_flag_is_%s_00d68080);
24    }
25    return 0;
26 }
27
```

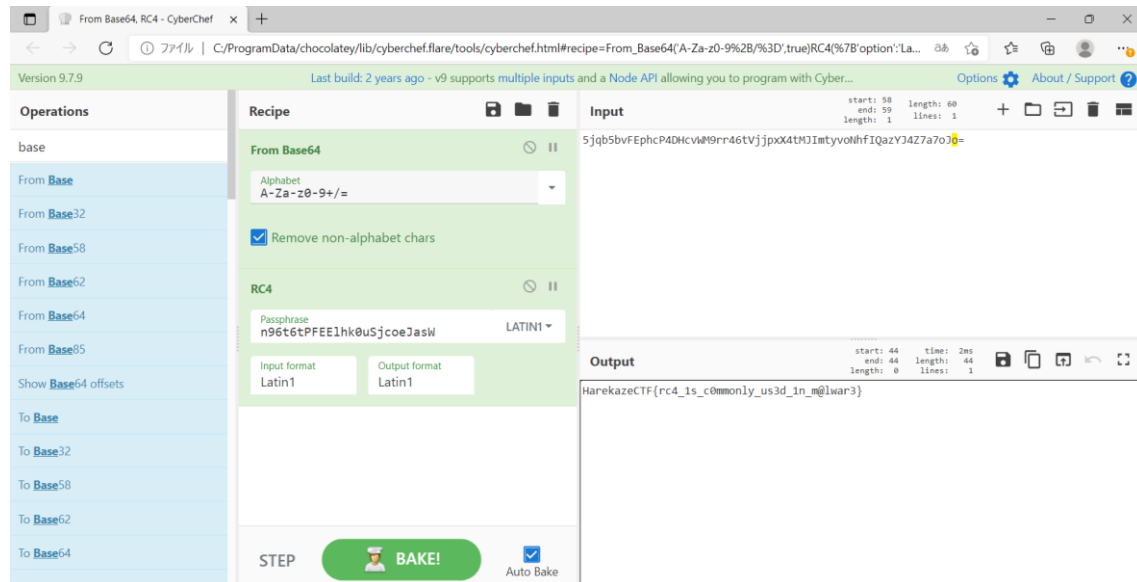
また、興味深い base64 文字列も見つかるが、base64 をデコードしても、意味のある情報は見つからない。

関数を上から折ってくと、次の関数が見つかる。

```
Decompile: FUN_00d074e0 - (challenge-dump_SCY.exe)
1
2 undefined4 __cdecl FUN_00d074e0(char *param_1,int param_2)
3
4 {
5     size_t sVar1;
6     uint local_10;
7     int local_c;
8     int local_8;
9
10    sVar1 = _strlen(param_1);
11    local_10 = 0;
12    local_c = 0;
13    while (local_c < 0x100) {
14        *(char *) (param_2 + local_c) = (char)local_c;
15        local_c = local_c + 1;
16    }
17    local_8 = 0;
18    while (local_8 < 0x100) {
19        local_10 = *(byte *) (param_2 + local_8) + local_10 + (int)param_1[local_8 % (int)sVar1] &
20            0x800000ff;
21        if ((int)local_10 < 0) {
22            local_10 = (local_10 - 1 | 0xffffffff00) + 1;
23        }
24        thunk_FUN_00d074b0((undefined *) (param_2 + local_8), (undefined *) (param_2 + local_10));
25        local_8 = local_8 + 1;
26    }
27    return 0;
28 }
```

256 文字の配列の初期化、配列内の swap を行っている。この後の処理をみても XOR をしている部分も見つかることから RC4 のアルゴリズムである。

Base64 をデコードした文字が、Rc4 で暗号化された文字列、「n96~~」の文字列がカギとなる。



「HarekazeCTF{rc4_1s_c0mmonly_us3d_1n_m@lwar3}」

余談：難易度低下案

解説作って思ったのが、初心者に対しても難易度高めな気がしました。以下のいずれかを行って下方修正してもいいかなと思いました。

- UPX をツールでもアンパックできるようにする
- RC4 の部分を取り除いて Base64 の文字列にする