

Reversing入門

...

自己紹介

- ・田口 仁大
- ・セキュリティエンジニア
- ・フリーランスとして活動する中、Omiserica Security Doctor株式会社のCTOとして就任。脆弱性診断業務やコンサル業務に従事している。

Reversingとは

CTFにおけるreversingとは未知のファイルが渡されるので解析するジャンル。
パスワードを当てたり、ゲームの得点を改ざんしたりしてFlag文字列を取得する。

今回、2021年8月4日から開催されたsetodaNoteCTFの問題を題材としてさわりを説明する。

setodaNoteCTFとは

2021.8.21 PM 9:00 ～ 2021.9.4 PM 9:00 (JST)で開催されたオンラインCTF。

入門者向けの問題も多く、得点を競うよりすべて解くことを目標とされたCTF。

※現在はアクセスできたが、いずれ閉じるとのこと。

<https://ctf.setodanote.net/>

今回取り扱う問題

- ELF
- Passcode
- Passcode2

ELF

Challenge

187 Solves



ELF

80

監獄というより研究室のような施設だった。見る角度が大切なんだ。ガラスで隔てられた部屋を白衣の男が歩いている。すべてを疑ってみることから始める。そばにある端末の電源が入る。手を動かして検証するというのは実に大事なことだ。

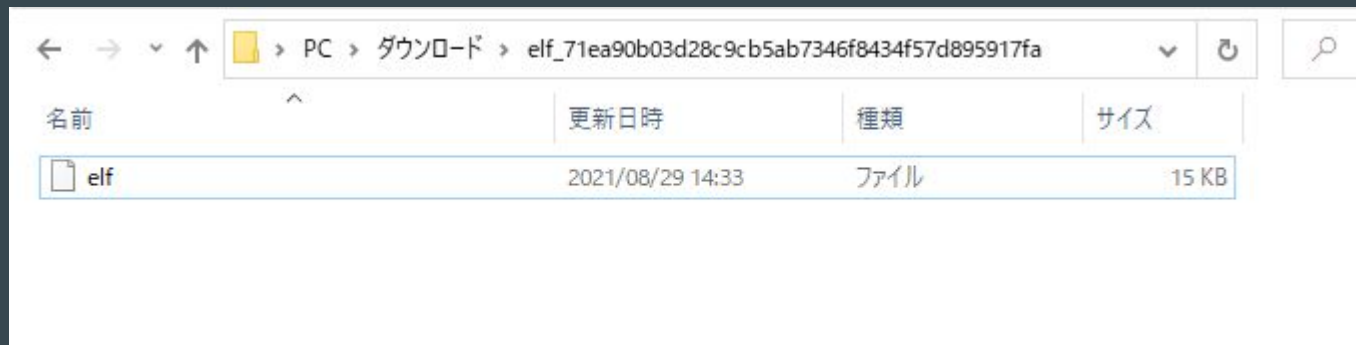
添付されたファイルを解析してフラグを入手してください。

 elf_71ea90b03d28c9cb5ab...

Flag

Submit

問題ファイルをDLし、zipを展開すると、「elf」というファイルが出てくる。



VMを起動し、Fileコマンドを確認すると、data fileとなる。

※ELFファイルなら、「ELF 64-bit LSB executable」といった情報が出てくる。

実行もできない。

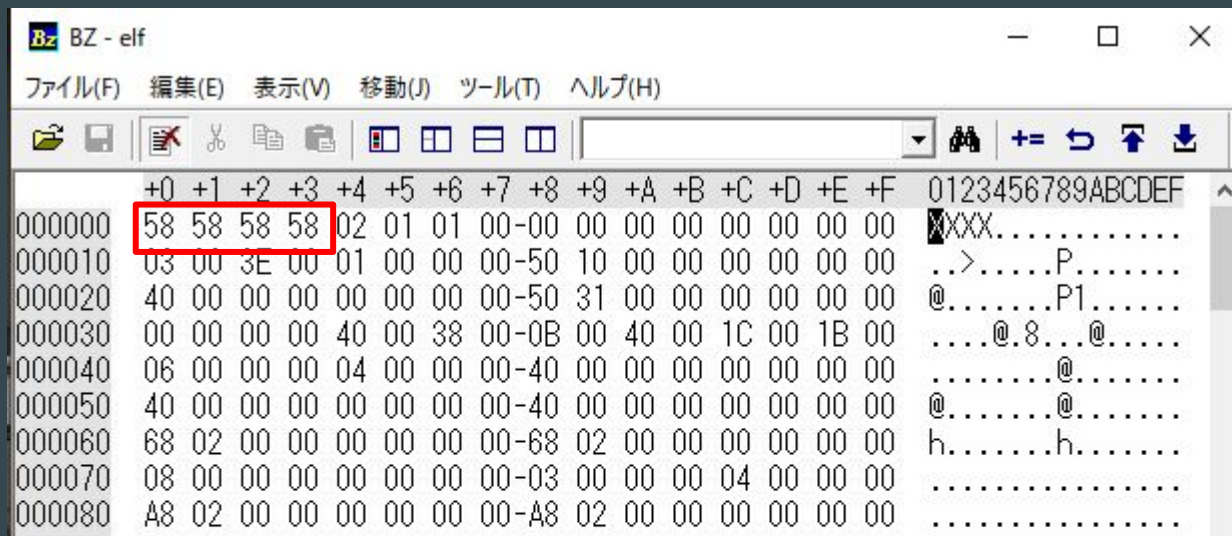
```
ctf@ctf-VirtualBox:~/temp$ file elf
elf: data
ctf@ctf-VirtualBox:~/temp$ ./elf
bash: ./elf: バイナリファイルを実行できません: 実行形式エラー
ctf@ctf-VirtualBox:~/temp$
```


方針

- 1.ELFファイルが壊れていると仮定し、バイナリエディタで確認する。
- 2.ELFファイルが壊れている場合、修復して実行できないか確認する。

バイナリエディタで確認するとマジックナンバーが「XXXX」になっている。

問題文が「ELF」なので、elfファイルのマジックナンバー「7F 45 4C 46」に書き換える。



linuxマシンで実行するとフラグが取れる。

```
ctf@ctf-VirtualBox:~/binary$ ./elf  
flag{run_makiba}  
ctf@ctf-VirtualBox:~/binary$
```

Passcode

Challenge

163 Solves

×

Passcode

120

その部屋はまぶしいほどの明かりで照らされていた。ここからが本番だ。白衣の人物が書類に目を落としながらつぶやくように話している。結果がすべてという訳ではないが。そばにある端末が起動する。いい結果を期待している。

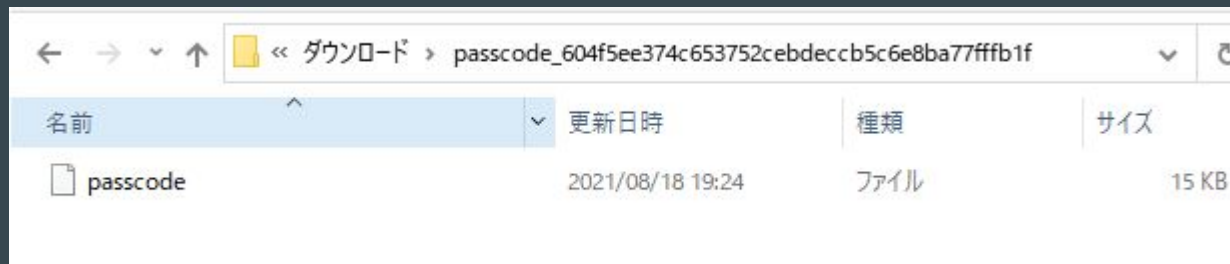
添付されたファイルを解析してフラグを得てください。

passcode_604f5ee374c653...

Flag

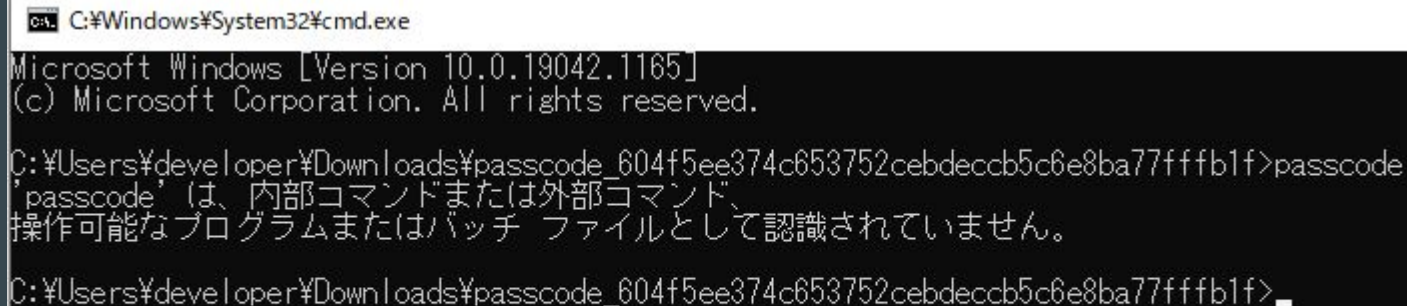
Submit

問題ファイルをDLし、zipを展開すると、「passcode」というファイルが出てくる。



Windows上で実行してみても実行できないことがわかる。

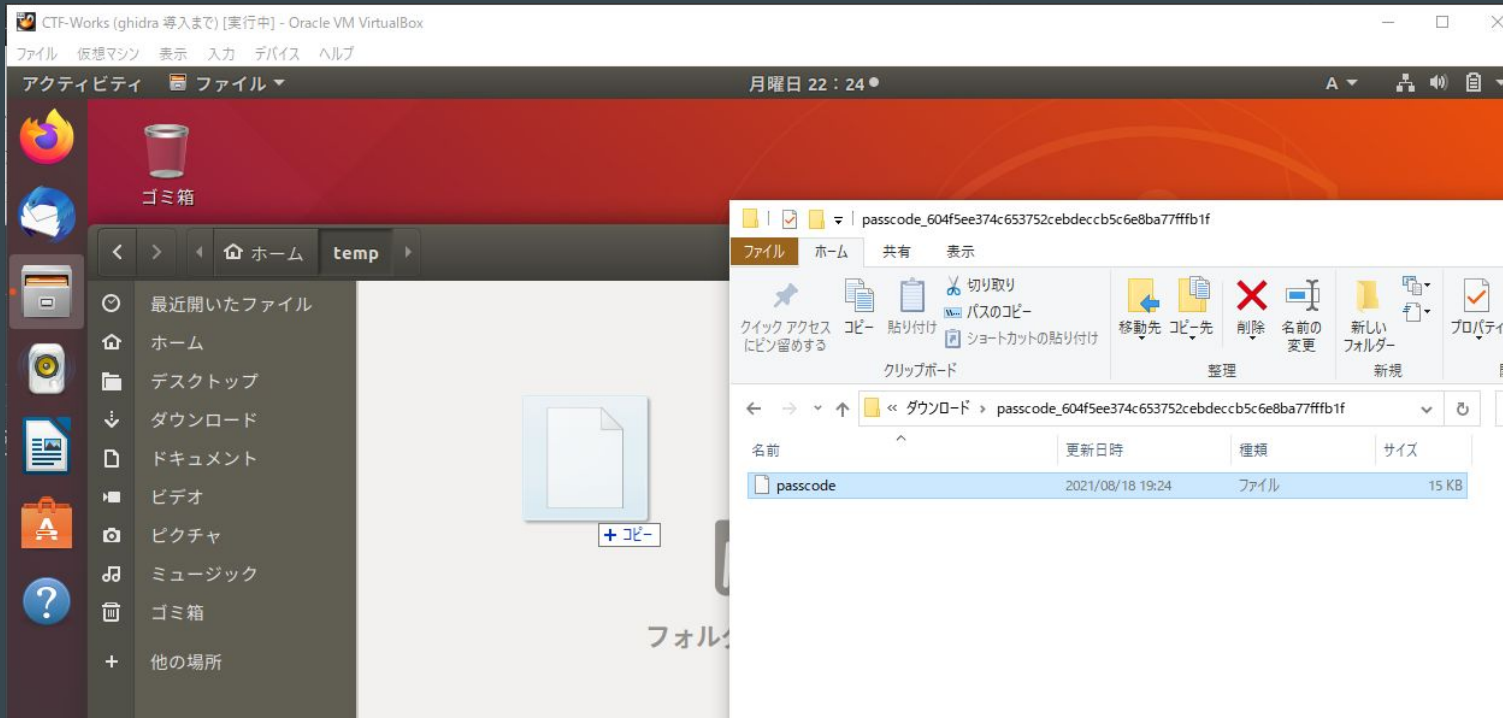
Linux上で実行を試みる。



```
CA C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19042.1165]
(c) Microsoft Corporation. All rights reserved.

C:\Users\developer\Downloads\passcode_604f5ee374c653752cebdeccb5c6e8ba77fffb1f>passcode
'passcode' は、内部コマンドまたは外部コマンド、
操作可能なプログラムまたはバッチ ファイルとして認識されていません。

C:\Users\developer\Downloads\passcode_604f5ee374c653752cebdeccb5c6e8ba77fffb1f>_
```



Passcodeを実行できるよう準備

```
ctf@ctf-VirtualBox:~/temp$ ls
passcode
ctf@ctf-VirtualBox:~/temp$ chmod 744 ./passcode
ctf@ctf-VirtualBox:~/temp$ ls
passcode
ctf@ctf-VirtualBox:~/temp$ ./passcode
Enter the passcode: █
```

どのような反応が変わるか確認

```
ctf@ctf-VirtualBox:~/temp$ ./passcode
Enter the passcode: hogebuga
Invalid passcode. Nice try.
ctf@ctf-VirtualBox:~/temp$ ./passcode
Enter the passcode: 1234567
Invalid passcode. Too short.
ctf@ctf-VirtualBox:~/temp$ ./passcode
Enter the passcode: 123456789
Invalid passcode. Too long.
ctf@ctf-VirtualBox:~/temp$
```

ここまでのまとめ

- ・問題対象のファイルは、Linuxの実行ファイル(ELFファイル)だとわかった。
- ・実行してみたところ、パスワードを求められた。
- ・7文字だと「Too Short」、9文字だと「Too Long」となったことから、文字列長は8文字であることがわかった。

ここから先の方針

- ・8文字のパスワードはなにか当てる問題
- ・ブルートフォース攻撃しても良いが時間がもったいないので、今回は実施しない。
- ・パスワードの一致比較関数を探す。
- ・ディスアセンブラであるGhidraを使ってパスワードを探す。

ghidraとは

NSAが開発したリバースエンジニアリングツールのこと。











自動で解析してアセンブラやファンクションツリーや、疑似C言語にしてくれる機能もある。(今回、この機能を活用する。)

デバッガ機能は(現状)ないので、動的解析することはできない。

Ghidraの起動

解析はWindowsマシンで行う。

ghidraRun.batで起動する。

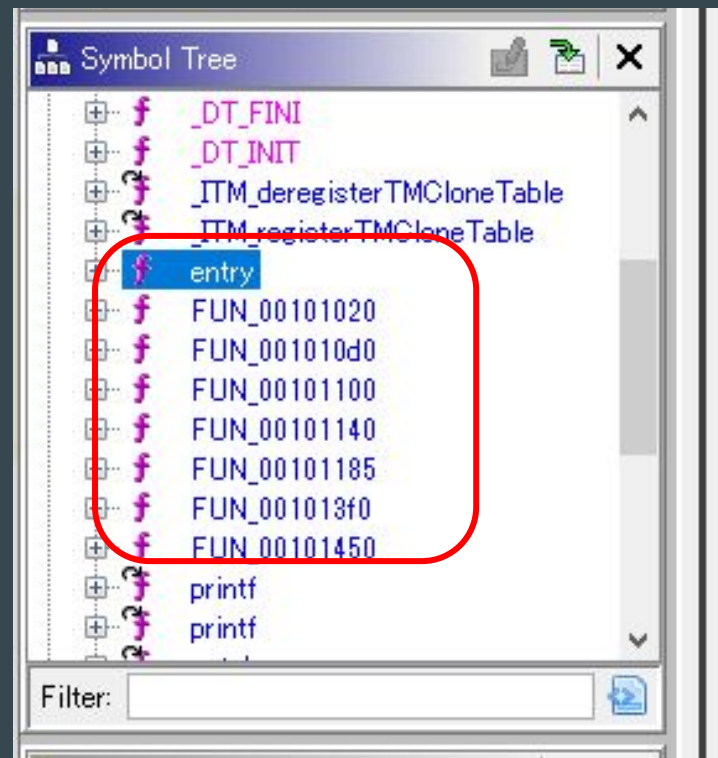
	docs	2020/12/15 14:30	ファイル フォルダー	
	Extensions	2020/12/15 14:30	ファイル フォルダー	
	Ghidra	2020/12/15 14:30	ファイル フォルダー	
	GPL	2020/12/15 14:30	ファイル フォルダー	
	licenses	2020/12/15 14:30	ファイル フォルダー	
	server	2020/12/15 14:30	ファイル フォルダー	
	support	2020/12/15 14:30	ファイル フォルダー	
	ghidraRun	2020/12/15 14:30	ファイル	1 KB
	ghidraRun.bat	2020/12/15 14:30	Windows バッチ ファ...	1 KB
	LICENSE	2020/12/15 14:30	ファイル	12 KB

ghidra画面左のSymbol Treeを見る。

FUN_nnnnnnnnnという関数がいくつもある。

この中にパスワード一致関数があると推測し

詳細を見ていく。



fun_00101185を見ると変数宣言をしているので、更に詳細に見ていく

```
*****
*                                     *
*                                     FUNCTION                                     *
*                                     *
*****

undefined FUN_00101185()

undefined      AL:1      <RETURN>
undefined8     Stack[-0x10]:8 local_10      XREF
undefined8     Stack[-0x18]:8 local_18      XREF
undefined8     Stack[-0x20]:8 local_20      XREF
undefined8     Stack[-0x28]:8 local_28      XREF
undefined8     Stack[-0x30]:8 local_30      XREF
undefined8     Stack[-0x38]:8 local_38      XREF
undefined8     Stack[-0x40]:8 local_40      XREF
undefined8     Stack[-0x48]:8 local_48      XREF
undefined8     Stack[-0x50]:8 local_50      XREF
undefined8     Stack[-0x58]:8 local_58      XREF
undefined8     Stack[-0x60]:8 local_60      XREF
undefined8     Stack[-0x68]:8 local_68      XREF
```



```
Listing: passcode
001012a0 48 c7 45 MOV     qword ptr [RBP + local_20],0x0
0000 00 00
001012a8 48 c7 45 MOV     qword ptr [RBP + local_20],0x0
0000 00 00
001012b0 48 c7 45 MOV     qword ptr [RBP + local_18],0x0
0000 00 00
001012b8 48 c7 45 MOV     qword ptr [RBP + local_10],0x0
0000 00 00
001012c0 48 8d 3d LEA     RDI,[s_Enter_the_passcode:_00102008]
0000 41 0d 00 00
001012c7 b8 00 00 MOV     EAX,0x0
0000 00 00
001012cc e8 8f fd CALL    printf
0000 ff ff
001012d1 48 8d 85 LEA     RAX=>local_108,[RBP + -0x100]
0000 00 ff ff ff
001012d8 48 89 c6 MOV     RSI,RAX
001012db 48 8d 3d LEA     RDI,[s_%255[^\n]*[^\n]_0010201d]
0000 3b 0d 00 00
001012e2 b8 00 00 MOV     EAX,0x0
0000 00 00
001012e7 e8 94 fd CALL    __isoc99_scanf
0000 ff ff
001012ec 83 f8 ff CMP     EAX,-0x1
001012ef 75 0a JNZ     LAB_001012fb
001012f1 b8 01 00 MOV     EAX,0x1
0000 00 00
001012f6 e9 e8 00 JMP     LAB_001013e3
```

アセンブラ表示

```
Decompile: FUN_00101185 - (passcode)
71
72 local_20 = 0;
73 local_18 = 0;
74 local_10 = 0;
75 printf("Enter the passcode: ");
76 iVar1 = __isoc99_scanf("%255[^\n]*[^\n]",&local_108);
77 if (iVar1 == -1) {
78     uVar2 = 1;
79 }
80 else {
81     __isoc99_scanf(&DAT_0010202c);
82     if ((char)local_108 == '\0') {
83         printf("Invalid passcode.");
84     }
85     else {
86         sVar3 = strlen((char *)&local_108);
87         if (sVar3 < 8) {
88             printf("Invalid passcode. Too short.");
89         }
90     }
91     else {
92         sVar3 = strlen((char *)&local_108);
93         if (sVar3 < 9) {
94             iVar1 = strcmp((char *)&local_108,"20150109");
95             if (iVar1 == 0) {
96                 puts("The passcode has been verified.\n");
97                 printf("Flag is : flag{%s}",&local_108);
98             }
99             else {
100                 printf("Invalid passcode. Nice try.");
101             }
102         }
103     }
104     else {
105         printf("Invalid passcode. Too long.");
106     }
107 }
```

デコンパイラ表示
※疑似C言語で表示

```
if (sVar3 < 9) {  
    iVar1 = strcmp((char *)&local_108, "20150109");  
    if (iVar1 == 0) {  
        puts("The passcode has been verified.\n");  
        printf("Flag is : flag{%s}", &local_108);  
    }  
    else {  
        printf("Invalid passcode. Nice try.");  
    }  
}
```

怪しい文字列をpasscodeとして入力してみると、フラグが取得できる。

```
ctf@ctf-VirtualBox:~/binary$ ./passcode
Enter the passcode: 20150109
The passcode has been verified.

Flag is : flag{20150109}
```

Passcode2

Challenge

80 Solves

×

Passcode2

150

予想以上の結果だった。今日もガラス越しに対象が目覚めます。ここまでうまくいったことはかつてない。端末に今日のデータを送信する。今度こそうまくいくかもしれない。

添付されたファイルを解析してフラグを得てください。

 passcode2_63485b6dbdde...

Flag

Submit

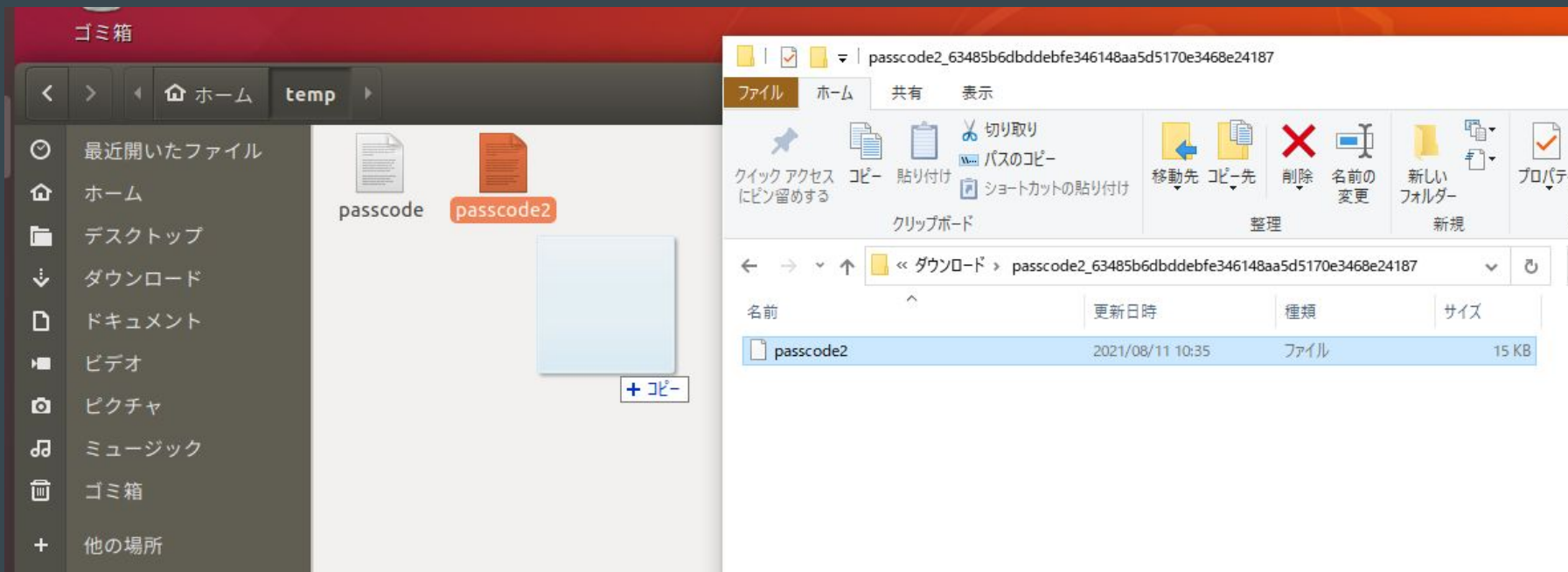
問題ファイルをDLし、zipを展開すると、「passcode2」というファイルが出てくる。

← → ▾ ↑					📁 << ダウンロード >> passcode2_63485b6dbddebfe346148aa5d5170e3468e24187		▾	🔄
名前		更新日時		種類	サイズ			
📄 passcode2		2021/08/11 10:35		ファイル	15 KB			

Windowsで実行できないことを確認。

```
C:¥Users¥developer¥Downloads¥passcode2_63485b6dbddebfe346148aa5d5170e3468e24187>passcode2
'passcode2' は、内部コマンドまたは外部コマンド、
操作可能なプログラムまたはバッチ ファイルとして認識されていません。

C:¥Users¥developer¥Downloads¥passcode2_63485b6dbddebfe346148aa5d5170e3468e24187>
```



fileコマンドで確認すると、ELF実行ファイルということがわかる。

```
ctf@ctf-VirtualBox:~/binary$ file passcode2
passcode2: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically l
inked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=a396332a87a60f8e35
3e93a001a1a9521673f19d, for GNU/Linux 3.2.0, stripped
ctf@ctf-VirtualBox:~/binary$
```


Passcode2を実行する準備

```
ctf@ctf-VirtualBox:~/temp$ ls
passcode  passcode2
ctf@ctf-VirtualBox:~/temp$ chmod 744 ./passcode2
ctf@ctf-VirtualBox:~/temp$ ls
passcode  passcode2
ctf@ctf-VirtualBox:~/temp$ ./passcode2
Enter the passcode:
```

Passcode2の挙動を確認

```
ctf@ctf-VirtualBox:~/temp$ ./passcode2
Enter the passcode: 1234567890
Invalid passcode. Too short.
ctf@ctf-VirtualBox:~/temp$ ./passcode2
Enter the passcode: 12345678901
Invalid passcode. Nice try.
ctf@ctf-VirtualBox:~/temp$ ./passcode2
Enter the passcode: 123456789012
Invalid passcode. Too long.
ctf@ctf-VirtualBox:~/temp$
```

ここまでのまとめ

- ・問題対象のファイルは、Linuxの実行ファイル(ELFファイル)だとわかった。
- ・実行してみたところ、パスワードを求められた。
- ・10文字だと「Too Short」、12文字だと「Too Long」となったことから、文字列長は11文字であることがわかった。

ここから先の方針

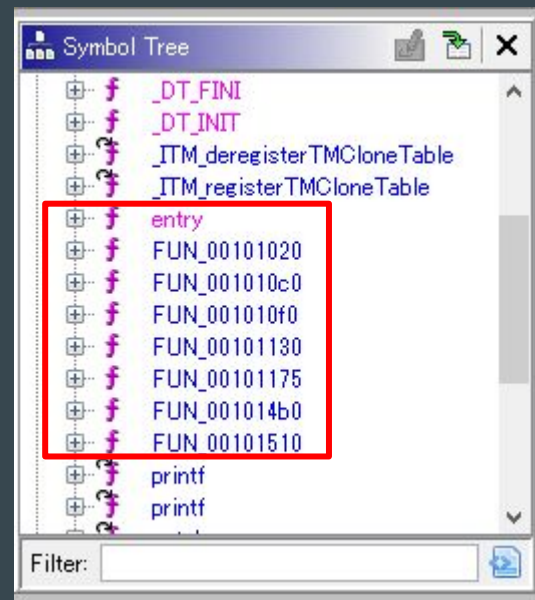
- ・11文字のパスワードはなにか当てる問題
- ・ブルートフォース攻撃しても良いが時間がもったいないので、今回は実施しない。
- ・パスワードの一致比較関数を探す。
- ・ディスアセンブラであるGhidraを使ってパスワードを探す。

ghidra画面左のSymbol Treeを見る。

FUN_nnnnnnnnnという関数がいくつもある。

この中にパスワード一致関数があると推測し

詳細を見ていく。



FUN_00101175を見ると、変数宣言されているので、詳細に見ていく

```
*****
undefined FUN_00101175()
undefined      AL:1      <RETURN>
undefined8     Stack[-0x10]:8 local_10      XREF

undefined8     Stack[-0x20]:8 local_20      XREF
undefined8     Stack[-0x28]:8 local_28      XREF
undefined8     Stack[-0x30]:8 local_30      XREF
undefined8     Stack[-0x38]:8 local_38      XREF
undefined8     Stack[-0x40]:8 local_40      XREF
undefined8     Stack[-0x48]:8 local_48      XREF
undefined8     Stack[-0x50]:8 local_50      XREF
undefined8     Stack[-0x58]:8 local_58      XREF
```

```

else {
    __isoc99_scanf(&DAT_0010202c);
    if ((char)local_118 == '\0') {
        printf("Invalid passcode.");
    }
    else {
        sVar3 = strlen((char *)&local_118);
        if (sVar3 < 0xb) {
            printf("Invalid passcode. Too short.");
        }
        else {
            sVar3 = strlen((char *)&local_118);
            if (sVar3 < 0xc) {
                sVar3 = strlen((char *)&local_118);
                if (sVar3 == 0xb) {
                    local_10 = 0;
                    while ((sVar3 = strlen((char *)local_124), local_10 < sVar3 &&
                        (*(byte *)((long)&local_118 + local_10) == (local_124[local_10] ^ 0x2a)))) {
                        local_10 = local_10 + 1;
                    }
                    sVar3 = strlen((char *)local_124);
                    if (local_10 == sVar3) {
                        puts("The passcode has been verified.\n");
                        printf("Flag is : flag(%s)", &local_118);
                    }
                    else {
                        printf("Invalid passcode. Nice try.");
                    }
                }
            }
        }
    }
}

```

デコンパイラ画面を見ると、**赤枠**部分にパスコードが正しい時の処理。
青枠部分がパスコードを検証している条件が記載されている。

パスコードを検証している箇所を読み解いて、パスコードを推測する。

主なパスコード検証箇所は以下の箇所

```
local_10 = 0;
while ((sVar3 = strlen((char *)local_124), local_10 < sVar3 &&
      (*(byte *)((long)&local_118 + local_10) == (local_124[local_10] ^ 0x2a)))) {
    local_10 = local_10 + 1;
}
sVar3 = strlen((char *)local_124);
if (local_10 == sVar3) {
    puts("The passcode has been verified.\n");
    printf("Flag is : flag{%s}", &local_118);
}
else {
    printf("Invalid passcode. Nice try.");
}
```


ポイントとなるのは赤枠の処理

読み解くと、

①local_124から11文字分格納されている値を「2a」でxorし比較

②合致していたらフラグを表示

```
local_10 = 0;
while ((sVar3 = strlen((char *)local_124), local_10 < sVar3 &&
    (* (byte *) ((long)&local_118 + local_10) == (local_124[local_10] ^ 0x2a)))) {
    local_10 = local_10 + 1;
}
sVar3 = strlen((char *)local_124);
if (local_10 == sVar3) {
    puts("The passcode has been verified.\n");
    printf("Flag is : flag{%s}", &local_118);
}
else {
    printf("Invalid passcode. Nice try.");
}
```

local_124から11文字分は、ghidraの
デコンパイラ画面を追っていくことで
確認できる。

```
local_124[0] = 0x18;  
local_124[1] = 0x1f;  
local_124[2] = 4;  
local_124[3] = 0x79;  
local_120 = 0x4f;  
local_11f = 0x5a;  
local_11e = 4;  
local_11d = 0x18;  
local_11c = 0x1a;  
local_11b = 0x1b;  
local_11a = 0x1e;  
local_119 = 0;
```

11文字分

0x2aでxorする。

```
local_124[0] = 0x18;  
local_124[1] = 0x1f;  
local_124[2] = 4;  
local_124[3] = 0x79;  
local_120 = 0x4f;  
local_11f = 0x5a;  
local_11e = 4;  
local_11d = 0x18;  
local_11c = 0x1a;  
local_11b = 0x1b;  
local_11a = 0x1e;  
local_119 = 0;
```

0x18 xor 0x2a = 0x32

0x1f xor 0x2a = 0x35

0x4 xor 0x2a = 0x2E

0x79 xor 0x2a = 0x53

0x4f xor 0x2a = 0x65

0x5a xor 0x2a = 0x70

0x4 xor 0x2a = 0x2E

0x18 xor 0x2a = 0x32

0x1a xor 0x2a = 0x30

0x1b xor 0x2a = 0x31

0x1e xor 0x2a = 0x34

xorした結果をASCII変換

```
local_124[0] = 0x18;  
local_124[1] = 0x1f;  
local_124[2] = 4;  
local_124[3] = 0x79;  
local_120 = 0x4f;  
local_11f = 0x5a;  
local_11e = 4;  
local_11d = 0x18;  
local_11c = 0x1a;  
local_11b = 0x1b;  
local_11a = 0x1e;  
local_119 = 0;
```

0x18 xor 0x2a = 0x32 => 2

0x1f xor 0x2a = 0x35 => 5

0x4 xor 0x2a = 0x2E => .

0x79 xor 0x2a = 0x53 => S

0x4f xor 0x2a = 0x65 => e

0x5a xor 0x2a = 0x70 => p

0x4 xor 0x2a = 0x2E => .

0x18 xor 0x2a = 0x32 => 2

0x1a xor 0x2a = 0x30 => 0

0x1b xor 0x2a = 0x31 => 1

0x1e xor 0x2a = 0x34 => 4

文字列を入力すると、フラグを取得できる。

```
ctf@ctf-VirtualBox:~/temp$ ./passcode2
Enter the passcode: 25.Sep.2014
The passcode has been verified.

Flag is : flag{25.Sep.2014}
ctf@ctf-VirtualBox:~/temp$
```

まとめ

- ・なんか書く