

# 情報セキュリティプロジェクト

## 5月 定例報告会

# 1. 今月の活動

1. 全体

2. 高レイヤ班

3. 低レイヤ班

4. 新入部員

# 2. 来月の予定

# 3. 技術的トピックス

# 今月の活動 – 全体

# •SECCON Beginners CTF 2021に参加

- 総得点: 587 Pt
- 1095チーム中 271位

271

CLPWN

587

2021-05-23T13:10:30+09:00

参考:2021の結果

参考:2019の結果

参考:2020の結果

CLPWN

240th place

353 points



CLPWN

Japan

156th place

661 points



# •SECCON Beginners CTF 2021に参加

## •反省

- 前回よりも得点，順位が落ちてしまった
- 難易度 Beginner ~ Easyの問題でも解けない問題が結構あった
  - 基礎力の不足？
  - カリキュラムを見直し，もう一度基礎力の習得を目指したい
  - また復習もしっかり行い，「前回と似ていた問題だけど解けなかった！」ということをなくしたい


参考:2021の結果

271	CLPWN	587	2021-05-23T13:10:30+09:00
-----	-------	-----	---------------------------

参考:2019の結果

CLPWN
240th place
353 points
 

参考:2020の結果

CLPWN
 Japan
156th place
661 points
 

# •SECCON Beginners CTF 2021に参加

## •反省

- 参加者が3人と少なめ

- できるだけ早くから，部員に参加の呼びかけや予定の確認を呼びかけていきたい
- SECCON Beginners自体，開催決定～開催までが急だったというものもあるが…

参考:2021の結果

271	CLPWN	587	2021-05-23T13:10:30+09:00
-----	-------	-----	---------------------------

参考:2019の結果

CLPWN
240th place
353 points
 

参考:2020の結果

CLPWN
 Japan
156th place
661 points
 

# 今月の活動 – 低レイヤ

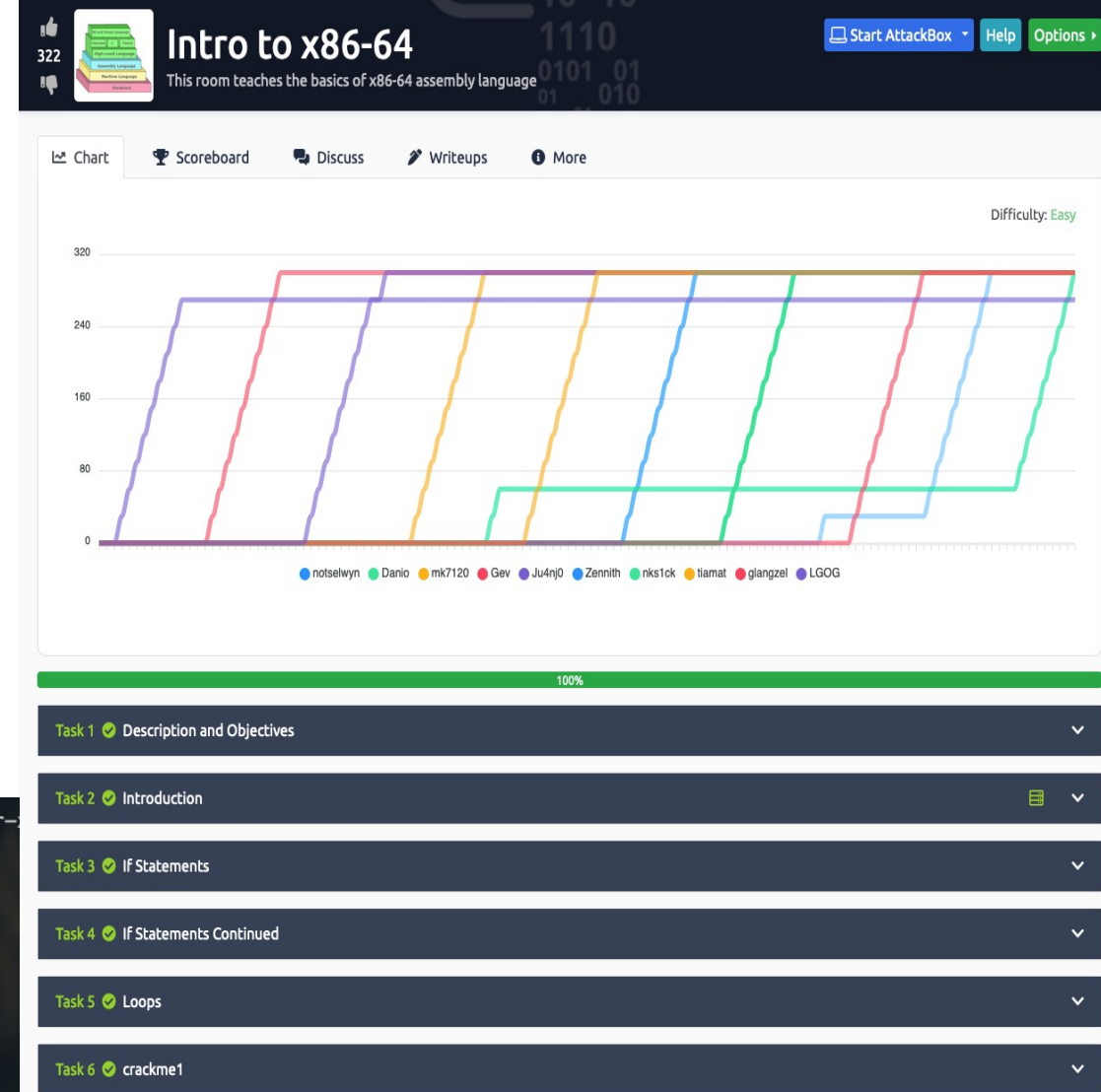
# • Reversing の復習

## • TryHackMe

(サイバーセキュリティ技術を学習するサイト)

の”Intro to x86-64”というコースでCPUの動作と，プログラムを解析する技術を復習した。

```
0x100003f04 ffc30d1 sub sp, sp, 0x30 ; [00] -r-
0x100003f08 fd7b02a9 stp x29, x30, [var_20h]
0x100003f0c fd830091 add x29, var_20h
0x100003f10 bfc31fb8 stur wzr, [var_0h] ; arg1
0x100003f14 bf831fb8 stur wzr, [var_8h] ; arg1
0x100003f18 bf831fb8 stur wzr, [var_8h] ; arg1
; CODE XREF from entry0 @ 0x100003f6c
0x100003f1c a8835fb8 ldur w8, [var_8h]
0x100003f20 09008052 movz w9, 0
0x100003f24 2801086b subs w8, w9, w8
0x100003f28 e9a79f1a cset w9, lt
0x100003f2c 29010012 and w9, w9, 1
0x100003f30 3f250071 cmp w9, 9
0x100003f34 ea010054 b.ge 0x100003f70
0x100003f38 a8835fb8 ldur w8, [var_8h]
0x100003f3c e00308aa mov x0, x8
0x100003f40 09000090 adrp x9, 0x100003000
```





# 今月の活動 – 高レイヤ

# • Web の脆弱性の復習

- TryHackMeの”OWASP Top 10”  
というコースで、実際に攻撃を行いながら  
Webの10大脆弱性を学習していった。
- OWASPはWebアプリケーションのセキュリティに  
関する研究などを行う団体。
- 『OWASP Top 10』は、  
悪用のしやすさ/弱点の蔓延度/弱点の検出のしやすさ/  
技術面への影響/ビジネスへの影響  
の観点から特に危険度の高い10種類の脆弱性が  
ピックアップされたものである。



# • Web の脆弱性の復習

- TryHackMeの”OWASP Top 10”  
というコースで、実際に攻撃を行いながら  
Webの10大脆弱性を学習していった。
- OWASP Top 10 の例:
  - インジェクション: 悪意のあるコードの挿入
  - 認証・アクセス制御の不備: 管理者権限の奪取
  - 機密データの露出: 暗号化,保護されていない個人  
人情報などの取得  
などが挙げられる。



# 今月の活動 – 新入部員

# •Linuxコマンドの学習

- 各種セキュリティツールの利用に不可欠なOS “Linux”の利用方法を学んだ。  
具体的には:
  - CD コマンド: フォルダ階層の移動
  - LS コマンド: フォルダの内容一覧の表示
  - CAT コマンド: ファイルの内容の表示
  - プログラムの実行方法: ./<FILE NAME>

## •C言語の学習

- 全体としては、  
変数の扱いと入出力系の関数  
まで学習を終えた。  
ポインタ・構造体まで習得した部員も

Linux Fundamentals Part 1

Get introduced to the Linux basics by learning how to use fundamentally important commands. Put this into practice by deploying and accessing your own remote (web-based) Linux machine.

Learn the Linux Fundamentals Part 1 | TryHackMe • Feb 7, 2021

Source: YouTube

10 10 1110 0101 0101 01 0101 01

TryHackMe

Learning Linux Part 1

見る YouTube

100%

Task 1 ☒ Intro

Task 2 ☒ Methodology

Task 3 ☒ [Section 2: Running Commands] - Basic Command Execution

Task 4 ☒ [Section 2: Running Commands] - Manual Pages and Flags

Task 5 ☒ [Section 3: Basic File Operations] - ls

Task 6 ☒ [Section 3: Basic File Operations] - cat

Task 7 ☒ [Section 3: Basic File Operations] - touch

Task 8 ☒ [Section 3: Basic File Operations] - Running A Binary

Task 9 ☒ Binary - Shiba1

Task 10 ☒ su

Task 11 ☒ Linux Fundamentals 2

来月の予定

## •全体

- SECCON Beginners CTF 2021のWrite-Up(回答)が公開されるので、それを元に復習を行う。

## •新入部員

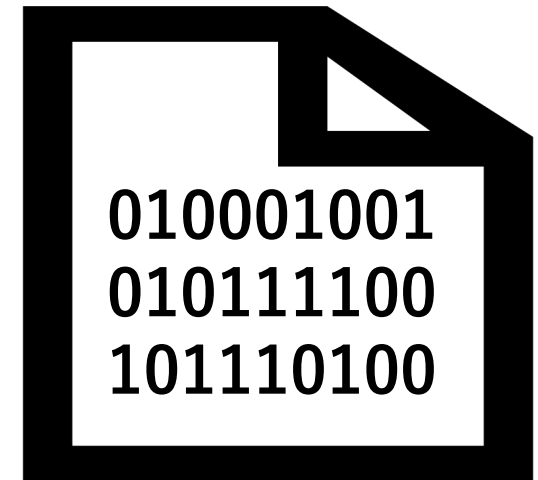
- 引き続きLinux コマンドの学習を行う。
  - 具体的には、SSHによるマシンの遠隔操作、ファイルの権限設定のコマンドの他、パイプなどを用いたコマンド同士の連携など、応用的な内容を学ぶ。

# 技術的トピックス: プログラムの解析



# •はじめに: プログラムの中身

- みなさんが普段PC, スマホで使っているプログラム(ソフトウェアやアプリ)は, 突き詰めれば“0 と 1”の集合体。
- CTFにおいて, 与えられたプログラムを”0と1”の状態から解析し, 処理を書き換えて**目的の値=フラグ**を提出させる種目が**Reversing 種目**である。



# • プログラムの過程

- 基本的には、以下の手順でソースコードからプログラムを生成します。

## ソースコード:

“プログラミング”で書くもの。  
人間がだいふ読みやすい形式。



## アセンブラコード:

機械語(0と1)と命令を対応\*させた言語。やや読みやすい。



## バイナリ:

“0と1”の集合体。人にはほぼ読めないが、PCで実行可能。

**これが”プログラム”**

**\*例:**

**ADD**(加算命令)は”**87**”と対応  
よって、バイナリで  
**87 03**(16進数表記)  
となっている部分は、  
“**3を加算**”という命令になる。

# •プログラムの過程

## ソースコード:

“プログラミング”で書くもの。  
人間がだいぶ読みやすい形式。

## アセンブラコード:

機械語(0と1)と命令を対応させた言語。やや読みやすい。

## バイナリ:

“0と1”の集合体。人にはほぼ読めないが、PCで実行可能。

```
1  #include<stdio.h>
2
3  int main(){
4      int i=0; //整数の変数 i=0
5
6      for(i=0;0<i<9;i++){
7          //iが9になるまでのループ
8
9              printf("%d",i);
10             //現在のiの値を出力
11         }
12     return 0;
13 }
```

```
ffc300d1    sub sp, sp, 0x30
fd7b02a9    stp x29, x30, [var_20h]
fd830091    add x29, var_20h
bfc31fb8    stur wzr, [var_0h]
bf831fb8    stur wzr, [var_8h]
bf831fb8    stur wzr, [var_8h]
entry0 @ 0x100003f6c
a8835fb8    ldur w8, [var_8h]
09008052    movz w9, 0
2801086b    subs w8, w9, w8
e9a79f1a    cset w9, lt
29010012    and w9, w9, 1
3f250071    cmp w9, 9
ea010054    b.ge 0x100003f70
a8835fb8    ldur w8, [var_8h]
e00308aa    mov x0, x8
09000090    adrp x9, 0x100003000
29d13e91    add x9, x9, 0xfb4
e00b00f9    str x0, [var_10h]
e00309aa    mov x0, x9
```

```
cffa edte 0c00 0001 0000 0000 020
1100 0000 6805 0000 8500 2000 000
1900 0000 4800 0000 5f5f 5041 474
524f 0000 0000 0000 0000 0000 000
0000 0000 0100 0000 0000 0000 000
0000 0000 0000 0000 0000 0000 000
0000 0000 0000 0000 1900 0000 d80
5f5f 5445 5854 0000 0000 0000 000
0000 0000 0100 0000 0040 0000 000
0000 0000 0000 0000 0040 0000 000
0500 0000 0500 0000 0500 0000 000
5f5f 7465 7874 0000 0000 0000 000
5f5f 5445 5854 0000 0000 0000 000
043f 0000 0100 0000 8000 0000 000
043f 0000 0200 0000 0000 0000 000
0004 0080 0000 0000 0000 0000 000
5f5f 7374 7562 7300 0000 0000 000
5f5f 5445 5854 0000 0000 0000 000
```

# • プログラムの解析 - Reversing

## バイナリ:

“0と1”の集合体。人にはほぼ読めないが、PCで実行可能。

```
cffa edfe 0c00 0001 0000 0000 0200
1100 0000 6805 0000 8500 2000 0000
1900 0000 4800 0000 5f5f 5041 4745
524f 0000 0000 0000 0000 0000 0000
0000 0000 0100 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 1900 0000 d801
5f5f 5445 5854 0000 0000 0000 0000
0000 0000 0100 0000 0040 0000 0000
0000 0000 0000 0000 0040 0000 0000
```

1. CTFでは、問題はこの形式(プログラムの形式)で与えられる。はじめに実際に動かしてみて、どのような挙動をするかを確かめる。

## アセンブラコード:

機械語(0と1)と命令を対応させた言語。やや読みやすい。

```
ffc300d1    sub sp, sp, 0x30
fd7b02a9    stp x29, x30, [var_20h]
fd830091    add x29, var_20h
bfc31fb8    stur wzr, [var_0h]
bf831fb8    stur wzr, [var_8h]
bf831fb8    stur wzr, [var_8h]
entry0 @ 0x100003f6c
a8835fb8    ldur w8, [var_8h]
09008052    movz w9, 0
7801085b    sub w8, w9, w8
```

2. ソフトウェアを使って、問題をこの形式に変形する。変形の正確性はほぼ100%  
なうえ、左よりだいぶ読みやすくなるので、この状態で問題を解くのが現実的。

## ソースコード:

“プログラミング”で書くもの。人間がだいぶ読みやすい形式。

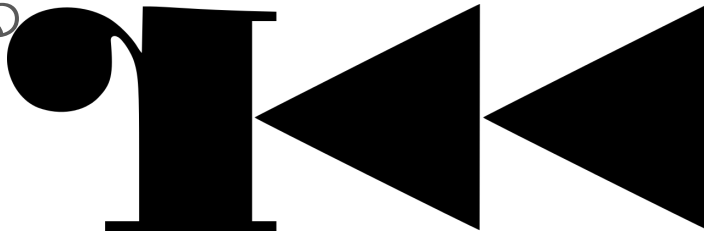
```
4      int i=0; //整数の変数 i=0
5
6      for(i=0;0<i<9;i++){
7          //iが9になるまでのループ
8
9          printf("%d",i);
10         //現在のiの値を出力
```

(おまけ)ソースコードに戻せるソフトウェアもあるが、無料のものはほぼ不正確、性能の良いものは100万円単位するので非現実的。



# • プログラムの解析 – Reversing

- 解析には **Radare2** というソフトウェアを使用する。
- 単純にプログラムをアセンブラコードに変形してくれるだけでなく、自動で注釈もつけてくれる。
- その上、プログラムを1行ずつ実行しながら、気になるところで止めて命令などを書き換えたり…と問題を解く上で便利な機能が数多く付属している。



```
r2 test
-- r2-goverity: found corruption - please eliminate!
[0x100003f04]> aaaaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[x] Finding xrefs in noncode section with anal.in=io.maps
[x] Analyze value pointers (aav)
[x] Value from 0x100000000 to 0x100004000 (aav)

[0x100003f04]> pdf@main
; UNKNOWN XREF from aav.0x100000020 @ +0xb0
;-- main:
;-- section.0.__TEXT.__text:
;-- _main:
;-- func.100003f04:
;-- pc:
128: entry0 (int64_t arg1, int64_t arg_30h);
; var int64_t var_8h @ x29-0x8
; var int64_t var_0h_2 @ sp+0x0
; var int64_t var_10h @ sp+0x10
; var int64_t var_0h @ sp+0x1c
; var int64_t var_20h @ sp+0x20
; var int64_t var_20h_2 @ sp+0x28
; arg int64_t arg_30h @ sp+0x60
; arg int64_t arg1 @ x0
0x100003f04 ffc300d1 sub sp, sp, 0x30 ; [00] -r-x section size 128 nam
0x100003f08 fd7b02a9 stp x29, x30, [var_20h]
0x100003f0c fd830091 add x29, var_20h
0x100003f10 bfc31fb8 stur wzr, [var_0h] ; arg1
0x100003f14 bf831fb8 stur wzr, [var_8h] ; arg1
0x100003f18 bf831fb8 stur wzr, [var_8h] ; arg1
; CODE XREF from entry0 @ 0x100003f6c
0x100003f1c a8835fb8 ldur w8, [var_8h]
0x100003f20 09008052 movz w9, 0
0x100003f24 2801086b subs w8, w9, w8
0x100003f28 e9a79f1a cset w9, lt
0x100003f2c 29010012 and w9, w9, 1
0x100003f30 3f250071 cmp w9, 9
0x100003f34 ea010054 b.ge 0x100003f70
0x100003f38 a8835fb8 ldur w8, [var_8h]
0x100003f3c e00308aa mov x0, x8
0x100003f40 09000090 adrp x9, 0x100003000
0x100003f44 29d13e91 add x9, x9, 0xfb4 ; 0x100003fb4 ; "%d"
0x100003f48 e00b00f9 str x0, [var_10h]
0x100003f4c e00309aa mov x0, x9 ; const char *format
0x100003f50 e9030091 mov x9, sp
0x100003f54 ea0b40f9 ldr x10, [var_10h] ; 0x4 ; 4
0x100003f58 2a0100f9 str x10, [x9]
0x100003f5c 0a000094 bl sym.imp.printf ; int printf(const char *format)
0x100003f60 a8835fb8 ldur w8, [var_8h]
0x100003f64 08050011 add w8, w8, 1
0x100003f68 a8831fb8 stur w8, [var_8h]
0x100003f6c ecffff17 b 0x100003f1c
; CODE XREF from entry0 @ 0x100003f34
0x100003f70 08008052 movz w8, 0
0x100003f74 e00308aa mov x0, x8
0x100003f78 fd7b42a9 ldp x29, x30, [var_20h]
0x100003f7c ffc30091 add sp, arg_30h ; 0x178000
0x100003f80 c0035fd6 ret
```

ご静聴ありがとうございました。