



Capture_7he_F1ag

Legal and fun way to Cybersecurity...

Basics of CyberSecurity:

- **A shield** : Cybersecurity protects our digital world—devices, networks, and data—from cyber threats like hackers and malware, much like a multi-layered fortress.
- **What is done to protect?** Various fields such as **network security** (guarding data in transit), **application security** (securing software), **endpoint security** (protecting individual devices), and **incident response** (swiftly countering breaches) comes under the field of **CyberSec**.
- **Ever-Evolving Battle:** Cybersecurity is a dynamic field where constant innovation meets creative defense strategies as new threats emerge and defenses adapt in real time.

What is Hacking?

Hacking is the practice of exploring computer systems to understand their inner workings and uncover vulnerabilities.

It is about pushing technological boundaries and continuously evolving in the digital landscape.

Threats :

● Injection Attacks:

SQL and XSS injection to manipulate databases or web apps.

In 2009, attackers exploited a SQL injection vulnerability at Heartland Payment Systems, gaining access to millions of credit card records by manipulating input fields.

● Malware & Ransomware:

Viruses or Ransomware can cripple systems by corrupting data or locking files until a ransom is paid.

WannaCry (2017), Exploited a Windows vulnerability to lock up computers worldwide, notably disrupting the UK's NHS and costing billions in losses.

● DDoS Attacks:

Overloading networks with excessive traffic can shut down services, resulting in lost revenue and tarnished reputations.

GitHub DDoS Attack (2018),

In one of the largest DDoS events, GitHub was hit by an amplified attack peaking at 1.35 Tbps that led to a brief but critical service disruption, stalling code deployments and impacting developers worldwide.

Coming to the point now ...

Capture the Flag!

- **Interactive Learning Environment:** CTFs are hands-on competitions where participants solve **cybersecurity puzzles** by finding hidden "flags" within challenges.
- **Diverse Competition Formats:** They include **Jeopardy-style contests** (solving independent puzzles in areas like web, crypto, and binary) and attack-defense rounds (simulating real-world system breaches and defenses).
- **Skill Integration:** These events test a broad range of talents—from vulnerability discovery and exploit creation to toolkit development and operational security—merging theoretical concepts with practical application.
- **Real- World Relevance:** Success in CTFs mirrors professional cybersecurity skills, providing valuable experience and acting as an indicator for potential job performance in the field.

How to get into Capturing the Flags?

Learn basics

Just an introduction is enough!

- C/C++
- Python
- Linux terminal

Get started

- Practice sessions
- picoCTF, HTB
- TryHackMe, crackmes.one

Get a hang of it

- Play wargames
- Get used to terminal and tools
- Discuss problems in groups

Discord servers have your back!

Dive into Battle!

- Get into any of the live CTF events
- Several platforms host CTF events everyday!
- Keep track of them at ctftime.org

Almost all the CTF challenges have the same structure:

PW Crack 5

Medium General Skills Beginner picoMini 2022 password cracking hashing

AUTHOR: LT 'SYREAL' JONES

Description

Can you crack the password to get the flag?
Download the password checker [here](#) and you'll need the encrypted flag and the hash in the same directory too. Here's a [dictionary](#) with all possible passwords based on the password conventions we've seen so far.

Hints 1

1 2 3

Opening a file in Python is crucial to using the provided dictionary.

30,622 users solved 91% Liked

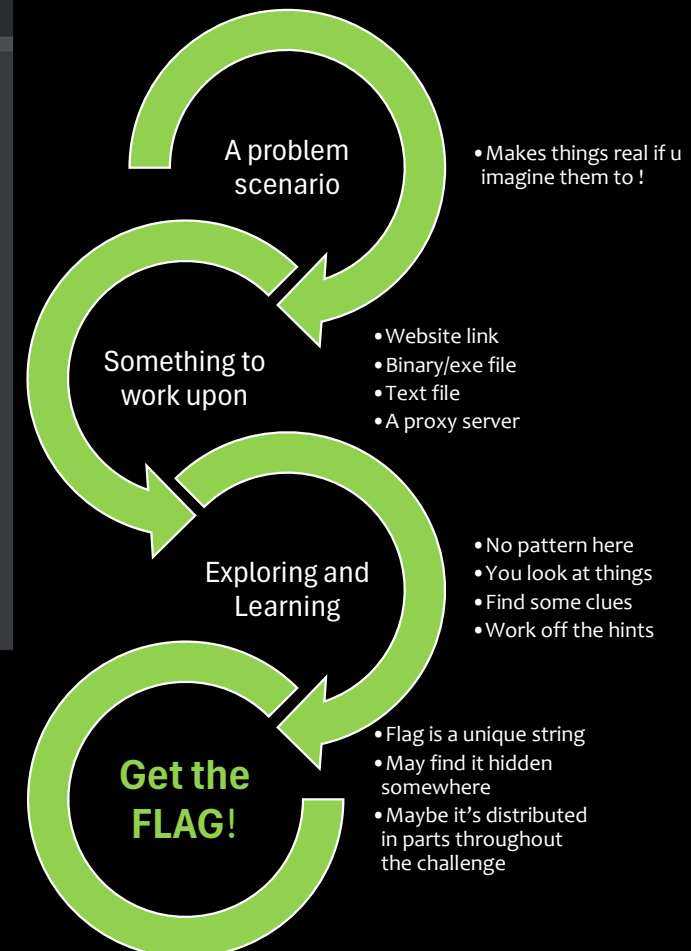
picoCTF{FLAG} Submit Flag

```
/bin/python3.13 /home/namikaze/pw3/level5.py
zsh: corrupt history file /home/namikaze/.zsh_history
(namikaze@namikaze) - [~/pw3]
$ /bin/python3.13 /home/namikaze/pw3/level5.py
Welcome back... your flag, user:
picoCTF{h45h_sl1ng1ng_40f26f81}
(namikaze@namikaze) - [~/pw3]
$
```

```
level3.py dic.py level5.py
level5.py ...
1 import hashlib
2
3 ## THIS FUNCTION WILL NOT HELP YOU FIND THE FLAG --LT #####
4 def str_xor(secret, key):
5     extend key to secret length
6     new_key = key
7     i = 0
8     while len(new_key) < len(secret):
9         new_key = new_key + key[i]
10        i = (i + 1) % len(key)
11    return ''.join([chr(ord(secret_c) ^ ord(new_key_c)) for (secret_c, new_key_c) in zip(secret, new_key)])
12 #####
13
14 flag_enc = open('level5.flag.txt.enc', 'rb').read()
15 correct_pw_hash = open('level5.hash.bin', 'rb').read()
16
17 dictionary = open('dictionary.txt', 'r').read()
18 dic_list = dictionary.split('\n')
19
20 def hash_pw(pw_str):
21     pw_bytes = bytearray()
22     pw_bytes.extend(pw_str.encode())
23     m = hashlib.md5()
24     m.update(pw_bytes)
25     return m.digest()
26
27 def level_5_pw_check(dic_list):
28     for i in range(1, 65537):
29         user_pw = dic_list[i]
30         user_pw_hash = hash_pw(user_pw.strip())
31
32         if( user_pw_hash == correct_pw_hash ):
33             print("Welcome back... your flag, user:")
34             decryption = str_xor(flag_enc.decode(), user_pw)
35             print(decryption)
36             return
37
38 level_5_pw_check(dic_list)
```

picoCTF{h45h_sl1ng1ng_40f26f81} Submit Flag

Hurray! You solved this challenge. X



Challenges we face

```
Almost there!! Crack this hash: 916e8c4f79b25028c9e467f1eb8eee6d6bbdff965f9928310ad30a8d88697745
Enter the password for the identified hash: qwerty098
Correct! You've cracked the SHA-256 hash with a secret found.
The flag is: picoCTF{UseStrOnG_h@shEs_&PaSswDs!_869e658e}

namikaze2022-picoctf@webshell:~$
```

Hashes to crack and get the correct passwords...

```
004022f8 51      PUSH    ECX
004022f9 8b 55 fc MOV     EDI,dword ptr [EBP + local_8]
004022fc 52      PUSH    EDI
004022fd 6a 00    PUSH    0x0
004022ff ff 15 0c CALL     dword ptr [USER32.DLL::LoadStringA]
                                = 0000312a
00402305 8d 85 60 LEA     EAX,[local_4a4,[EBP + 0xfffffb60]]
0040230b 50      PUSH    EAX
0040230c 8d 8d 68 LEA     ECX,[local_9c,[EBP + 0xfffffb68]]
00402312 e8 19 ff CALL     MD5::digestString
                                char * digestString(MD5 * this, ...
00402317 89 85 64 MOV     dword ptr [EBP + local_a0],EAX
0040231d 6a 30    PUSH    0x30
0040231f 68 30 30 PUSH    s_We've_been_compromised!_00403030
                                = "We've been compromised!"
00402325 40 00
```

```
2 void entry(void)
3 {
4 {
5     CHAR local_4a4;
6     undefined4 local_4a3 [1027];
7     char *local_a0;
8     MD5 local_9c [144];
9     HRSRC local_c;
10    undefined4 local_8;
11
12    MD5::MD5(local_9c);
13    local_4a4 = "\0";
14    memset(local_4a2,0,0x3ff);
15    local_8 = 0;
16    local_c = FindResourceA((HMODULE)0x0,"rc.rc", (LPCSTR)0x6);
17    local_8 = 0x110;
18    LoadStringA((HINSTANCE)0x0,0x110,&local_4a4,0x3ff);
19    local_a0 = MD5::digestString(local_9c,&local_4a4);
20    MessageBoxA((HWND)0x0,local_a0,"We've been compromised!",0x30);
21    /* WARNING: Subroutine does not return */
22    ExitProcess(0);
23 }
```

Analyzing **disassembled code** from binaries/executables

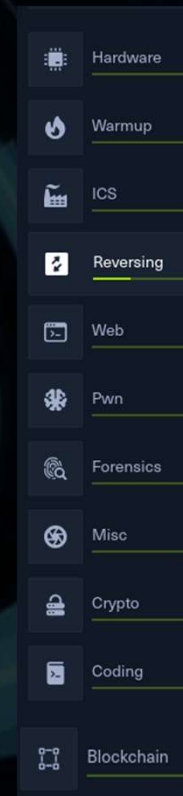
Understanding **decompiled code** to reverse engineer the algorithm

#Add one more screenshot from some new challenge type.
Maybe **crypto** or, **Web**?

Question types

The problems in CTF events are broadly divided into the following categories:

- Hardware
- ICS
- Reversing
- Web
- Python Wrangling
- Forensics
- Crypto
- Coding
- Blockchain
- Miscellaneous



Reverse Engineering (RE): A useful skill

One of the more interesting and difficult type of challenge in RE :

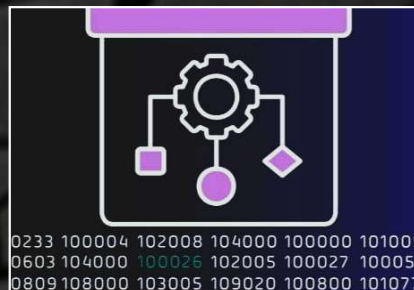
- Provided a piece of code or any binary/exe file, the task would be to find out how it works modify or reverse the process to accomplish certain things (like obtaining a flag).



Analysing binaries

- Participants are provided with a compiled program (e.g., .exe, .bin, or .elf files).
- The goal is to understand the program's functionality without access to its source code.

- Challenges may require reverse-engineering algorithms used in the program.
- This could involve deciphering encryption methods, compression techniques, or custom logic.



Understanding Algorithms



Bypassing Protection

- Some challenges include anti-debugging mechanisms or obfuscation techniques.
- We may be required to bypass these protections to analyze the program effectively.

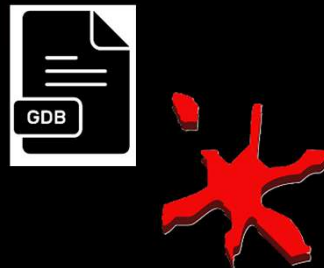
Common tools used for RE:

Disassembler



Tools like **IDA Pro**, **Ghidra** or **Radare2** are used to convert binaries into assembly code.

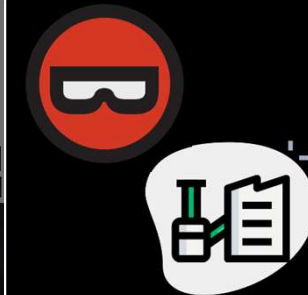
Debuggers



Tools like **GDB**, **OLLY DBG** help in through the program's execution.

- Executes programs step-by-step in real time.
- Sets breakpoints and inspects memory and registers.
- Helps bypass anti-debugging techniques.

Decompiler



Tools like **Binary Ninja** or **Hopper** attempt to reconstruct source code from binaries.

- Converts binaries into high-level pseudocode.
- Highlights loops, conditionals and structures.
- Accelerates overall code comprehension.

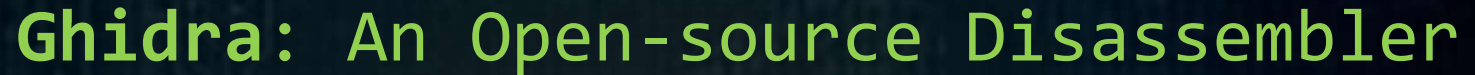
Hex Editors



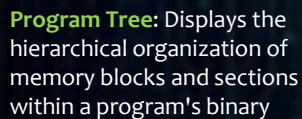
Tools like **HxD** allow us to view and edit the raw binary data.

- Displays file data in hex and ASCII.
- Permits direct byte-level modifications.
- Detects Patterns and embedded signature.

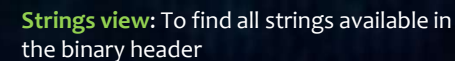
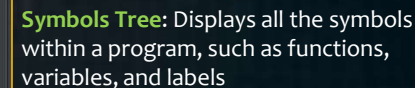
- Converts binary code into assembly language.
- Visualizes control flow and functions.
- Allows annotations for easier analysis.



Decompiler: To view a pseudo source code in C



Disassembly: To view Assembly code form the binary/exe files



- ❑ Ghidra is a powerful tool for analyzing binaries and understanding code execution flow through static analysis.

- ❑ By thoroughly examining a binary with Ghidra, we can uncover hidden functions that may not be apparent during runtime.
- ❑ Additionally, in CTF challenges, Ghidra's string analysis can quickly reveal function call references or even the flag itself in the Strings view, significantly simplifying the reversing process.

Techniques used for RE:

Static Analysis (Without Executing the Binary)



Disassembly: Using tools like Ghidra, IDA to analyze assembly code.

Decompilation: Converting binary code back into a high-level language (C, Python).

String Analysis: Extracting human-readable strings to find function calls and flag data.

Symbol Table Analysis: Checking for Function names, global variables, and debug symbols.

Control Flow Analysis: Understanding function calls and loops through CFG(Control Flow Graphs).

Dynamic Analysis (Executing and Observing Behaviour)



Debugger Usage: Using GDB, WinDBG or OllyDBG to set breakpoints and inspect registers/memory.

Tracing Execution: Using tools like strace (Linux) or API Monitor (Windows) to track system calls.

Memory Inspection: Dumping and analyzing process memory with tools like Volatility or Cheat Engine.

Hooking & Patching: Injecting custom code to modify behaviour at runtime.


During the past few days:


■ Basic Malware RE	Medium	Reverse	● Solved			FLAG{CAN-MAKE-IT-ANYMORE}	March 23, 2025	March 24, 2025
■ EncryptedScroll [CyberApocalypse]	Very Easy	Reverse	● Solved	900	HTB{s1mpl3_fl4g_4r1thm3t1c}		March 22, 2025	March 22, 2025
■ SealedRune	Very Easy	Reverse	● Solved	850	HTB{run3_m4g1c_r3v34l3d}		March 22, 2025	March 22, 2025
■ Endlesscycle	Easy	Reverse	● Need Help	950			March 22, 2025	
■ Flag Casino [Try-Outs]	Very Easy	Reverse	● Solved	975	HTB{needle_in_the_stack} some		March 17, 2025	March 20, 2025

and many more ...

Flag Casino (HTB Try-outs): RE challenge


CTF Try Out


 Challenges

 Scoreboard

Overview


Solves

 Challenge Pwned

 20 Mar, 2025 16:13

CHALLENGE NAME


FlagCasino



The team stumbles into a long-abandoned casino. As you enter, the lights and music whirl to life, and a staff of robots begin moving around and offering games, while skeletons of prewar patrons are slumped at slot machines. A robotic dealer waves you over and promises great wealth if you can win - can you beat the house and gather funds for the mission?

Submit flag & press enter



 Download Files

Files to assist you in finding the flag

975
POINTS

very easy
DIFFICULTY



```
Provided a zip file
containing a single binary
named casino.
```

```

namikaze@namikaze: ~/Desktop/stuff/revengg/rev_flagcasino
zsh: corrupt history file /home/namikaze/.zsh_history
[namikaze@namikaze]~$ cd ~/Desktop/stuff/revengg/rev_flagcasino
[namikaze@namikaze]~/Desktop/stuff/revengg/rev_flagcasino$ ./casino
[ ** WELCOME TO ROBO CASINO ** ]

      /\
     (oo)
      (o)
      ||
  [ ] /-----\ [ ] \
    / \         \ \
   /   \         \
  (     \         \

-----
[*** PLEASE PLACE YOUR BETS ***]
> A
[ * INCORRECT * ]
[ *** ACTIVATING SECURITY SYSTEM - PLEASE VACATE *** ]

[namikaze@namikaze]~/Desktop/stuff/revengg/rev_flagcasino$

```

A problem from CTF Try-out
on **HackTheBox** platform.
Reversing category;
Difficulty: **Very Easy**

On running the binary, I was prompted for an input used a random char 'A', to checkout how the program goes.
No clue found from this output yet.

Flag Casino: Continued...

```

1
2 undefined8 main(void)
3
4 {
5     int buffer;
6     char input;
7     uint num;
8
9     puts("[ ** WELCOME TO ROBO CASINO **]");
10    puts(
11        "
12            , .\n      (\n_____) \n      (_oo_) \n      (O) \n      _||_   \\ \n\n      {} / ____\\ \\ \n\n      / ____ / \\ \\ \n / ___ \\ \\ \n / ____ \\ \\ \n-----"
13    );
14    puts("[*** PLEASE PLACE YOUR BETS ***]");
15    num = 0;
16    while( true ) {
17        if (28 < num) {
18            puts("[ ** HOUSE BALANCE $0 - PLEASE COME BACK LATER ** ]");
19            return 0;
20        }
21        printf("> ");
22        buffer = __isoc99_scanf(&DAT_001020fc,&input);
23        if (buffer != 1) break;
24        srand((int)input);
25        buffer = rand();
26        if (buffer != *(int *)(&check + (long)(int)num * 4)) {
27            puts("[ * INCORRECT * ]");
28            puts("[ *** ACTIVATING SECURITY SYSTEM - PLEASE VACATE *** ]");
29            /* WARNING: Subroutine does not return */
30            exit(-2);
31        }
32        puts("[ * CORRECT * ]");
33        num = num + 1;
34    }
35    /* WARNING: Subroutine does not return */
36    exit(-1);
37}

```

Looking at the decompiled pseudo-code in `ghidra`, the following points can be noted:

- We are supposed to enter a single character for each loop
- The loop is supposed to run for 29 times (maybe the length of flag?)
- We see that the input char is converted to int and used to seed the `rand()` function.
- Then the number from `rand()` is compared to the int value stored in resource check

```
namikaze@namikaze: ~/Desktop/stuff/revenegg/rev_flagcasino... 🔍 ⓘ ⌵
```

```
zsh: corrupt history file /home/namikaze/.zsh_history  
[namikaze@ namikaze] ~  
$ cd ~/Desktop/stuff/revenegg/rev_endlesscycle  
  
[namikaze@ namikaze] ~/Desktop/stuff/revenegg/rev_endlesscycle  
$ cd ~/Desktop/stuff/revenegg/rev_flagcasino  
  
[namikaze@ namikaze] ~/Desktop/stuff/revenegg/rev_flagcasino  
$ ./casino  
[ ** WELCOME TO ROBO CASINO **]  
  
      _ _ _  
     (oo)  
    (o)  
   _||_  
  [ ] [ ] [ ] [ ]  
 /   \   \   \  
/_ _ _/_ _ _\
```

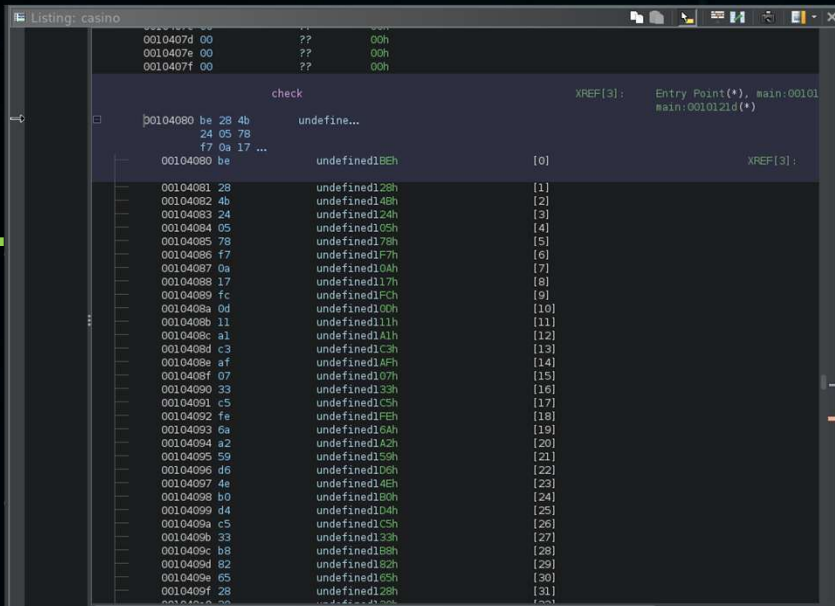
```
-----  
[*** PLEASE PLACE YOUR BETS ***]  
  
> H  
[ * CORRECT *]  
> T  
[ * CORRECT *]  
> B  
[ * CORRECT *]  
> █
```

Knowing that all HTB flags have the structure `HTB{flag}`. I guessed maybe, it's supposed to check flag as the input. So, I tried the characters H, T, B ... Clue spotted!

Flag Casino: Continued...

Analyzing the while loop further to understand how the input is checked:

- we know that the `rand()` gives a fixed value for a constant seeding.
- the `rand()` value is stored in `buffer` which is then compared to `check`
- notice that for each successful iteration, the next comparison is made by moving 4 steps ahead in `check`.



```
while( true ) {
    if (28 < num) {
        puts("[ ** HOUSE BALANCE $0 - PLEASE COME BACK LATER ** ]");
        return 0;
    }
    printf("> ");
    buffer = __isoc99_scanf(&DAT_001020fc,&input);
    if (buffer != 1) break;
    srand((int)input);
    buffer = rand();
    if (buffer != *(int*)(check + (long)(int)num * 4)) {
        puts("[ * INCORRECT * ]");
        puts("[ *** ACTIVATING SECURITY SYSTEM - PLEASE VACATE *** ]");
        /* WARNING: Subroutine does not return */
        exit(-2);
    }
    puts("[ * CORRECT * ]");
    num = num + 1;
}
/* WARNING: Subroutine does not return */
exit(-1);
}
```

The `check` resource contains a series of **hex values** of undefined data type (this is typecasted to `(int *)`) in the code. So, in each loop the comparison value is moved by 4 bytes in `check`.

```

Listing: casino
001011f0 e8 7b fe    CALL    <EXTERNAL>::exit          void exit(int __status)
ff ff
-- Flow Override: CALL_RETURN (CALL_TERMINATOR)
LAB_001011f5
001011f5 0f b6 45 fb    MOVZX   buffer,byte ptr [RBP + input]  XREF[1]: 001011e9(j)
001011f9 0f be c0      MOVZX   buffer,buffer
001011fc 89 c7        MOV     EDI,buffer
001011fe e8 ad fe     CALL    <EXTERNAL>::srand          void srand(uint __seed)
ff ff
00101203 e8 7b fe     CALL    <EXTERNAL>::rand          int rand(void)
ff ff
00101208 8b 55 fc     MOV     EDX,dword ptr [RBP + num]
0010120b 48 63 d2     MOVSDX  RDX,EDX
0010120e 48 bd 0c     LEA     RCX,[RCX*0x4]
95 00 00
00 00
00101216 48 bd 15     LEA     RDX,[check]
63 24 00 00
0010121d 8b 14 11     MOV     EDX=>check,dword ptr [RCX + RDX*0x1]
00101220 39 60      CMP     buffer,EDX
00101222 75 0e      JNZ     LAB_00101232
00101224 48 bd 3d     LEA     RDI,[s[_*_CORRECT_*]_00102100]  = "I * CORRECT *I"
d5 0e 00 00
0010122b e8 00 fe     CALL    <EXTERNAL>::puts          int puts(char * __s)
ff ff
00101230 eb 22      JMP     LAB_00101254
LAB_00101232
00101232 48 bd 3d     LEA     RDI,[s[_*_INCORRECT_*]_0010210f] XREF[1]: 00101222(j)
d5 0e 00 00
00101239 e8 f2 fd     CALL    <EXTERNAL>::puts          int puts(char * __s)
ff ff
0010123e 48 bd 3d     LEA     RDI,[s[_*** ACTIVATING_SECURITY_SYSTEM_001021... = "I *** ACTIVATING SEC
e3 0e 00 00
00101245 e8 e6 fd     CALL    <EXTERNAL>::puts          int puts(char * __s)
ff ff
0010124a b7 fe ff     MOV     EDI,0xffffffff
ff ff
0010124f e8 1c fe     CALL    <EXTERNAL>::exit          void exit(int __status)
ff ff

```

The one thing that comes to mind is to change the source code in such a way that we obtain the values in check, but for that I'd have to change assembly code...which is a rather difficult endeavour .

So, can it be done in any other way?

- ✓ The approach is to make a map of all characters as per their respective `rand()` index obtained by using the character itself as `seed`.
- ✓ Then, all I need to do is get the integer value that `check` is supposed to compare `buffer` with, because the character in the map at that index will lead to the flag string (by combining all such characters).



```

test.sh  level5.py  lvl5.txt  decrypt.py  enc.py
rev_flagcasino > script.py > ...
1  from pwn import *
2  import ctypes as c
3
4  libc = c.CDLL('libc.so.6')
5  map={}
6  for seed in range(255):
7      libc.srand(seed)
8      map[libc.rand()]=chr(seed)
9
10 flag = ""
11 casino = ELF("./rev_flagcasino/casino", checksec=False)
12
13 for num in range(29):
14     key = casino.u32(casino.sym["check"] + num*4)
15     flag += map[key]
16
17 print(flag)

```

This python script does that task. Using the `pwn` library, the data check can be extracted from the binary file. Moreover, the `ctypes` library enables the use of C functions to emulate the features of obfuscation of string using `rand()` function.

Running the script, the flag itself is directly revealed!

Flag: `HTB{r4nd_1s_v3ry_pr3d1ct4bl3}`

```

PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

File "/home/namikaze/Desktop/stuff/rev_flagcasino/script.py", line 15, in <module>
    flag += map[key]
           ~~~~
KeyError: 0

(nami/bin/python3.13 /home/namikaze/Desktop/stuff/rev_flagcasino/script.py
Traceback (most recent call last):
  File "/home/namikaze/Desktop/stuff/rev_flagcasino/script.py", line 15, in <module>
    flag += map[key]
           ~~~~
KeyError: 0

(nami/bin/python3.13 /home/namikaze/Desktop/stuff/rev_flagcasino/script.py
HTB{r4nd_1s_v3ry_pr3d1ct4bl3}

(namikaze@namikaze) - [~/Desktop/stuff]
$

```


Key take-aways from this problem:

- ❑ `rand()` gives same output for the same seed.
- ❑ use of `pwn` library to extract specific data from a binary
- ❑ `brute-forcing` such a problem is not a wise approach.

Resources used:

- ❑ **Ghidra** for disassembly, Decompilation
- ❑ **HackTheBox** for the challenge.
- ❑ **bingqer.com** and **medium.com** for problem writeups.