



OliCyber.IT - Training Camp 2

Writeup simulazioni



Misc 1 - Sanity check

Leggi (cit.)



Misc 2 - Gab-chan.png

- La challenge consiste in un file png in cui è stata nascosta la flag tramite *steganografia*
- Caricando la foto su un sito come aperisolve si possono trovare 3 dati nascosti:
 - La password *VjyQ[6M8WFX[sLCT*
 - La presenza di un file chiamato flag.txt salvato nella foto tramite *steganografia lsb (least significant bit)*
 - Il commento !flag: https://bit.ly/s3cr3t_m3s5ag3 il cui link redirecta a un Rickroll
- Utilizzando un tool come *stego-lsb* si può estrarre il file nascosto
- Il file è uno zip protetto dalla password *VjyQ[6M8WFX[sLCT* che contiene il file flag.txt con la flag



Misc 3 - SSH login

- La challenge è una shell SSH fittizia
- Viene allegato un pcap contenente traffico di un login
- Nel pcap si legge facilmente l'username e la password di login
- Si può quindi collegarsi, analizzare il file system e trovare la flag come se fosse una vera shell.

```
*** System restart required ***
gabibbo@olicyber.it:/home/gabibbo# cd Desktop
gabibbo@olicyber.it:/home/gabibbo/Desktop# cd olicyber
gabibbo@olicyber.it:/home/gabibbo/Desktop/olicyber# cd 2022
gabibbo@olicyber.it:/home/gabibbo/Desktop/olicyber/2022# cd Olicyber_git_repository_2022
gabibbo@olicyber.it:/home/gabibbo/Desktop/olicyber/2022/Olicyber_git_repository_2022# cd network_brutta
gabibbo@olicyber.it:/home/gabibbo/Desktop/olicyber/2022/Olicyber_git_repository_2022/network_brutta# cat flag.txt
flag{b3lla_st4_sh3ll_ssh...}
gabibbo@olicyber.it:/home/gabibbo/Desktop/olicyber/2022/Olicyber_git_repository_2022/network_brutta# |
```



Web 1 - MEME SHOP

Abbiamo uno shop di immagini in cui tutto costa 1, tranne la flag che costa 100. Partiamo con un credito di 10 e possiamo comporre il nostro carrello aggiungendo i singoli elementi.

Infine una view per tutti gli acquisti correttamente eseguiti ed una per eseguire il vero e proprio checkout del carrello.

Bilancio: 10 € - Shop - Carrello - Lista acquisti		
flag	100	Aggiungi al carrello
Videogame	1	Aggiungi al carrello
AI	1	Aggiungi al carrello
Malevisione	1	Aggiungi al carrello
Seal of approval	1	Aggiungi al carrello
Gaspere	1	Aggiungi al carrello



Web 1 - MEME SHOP (cont)

Nel sorgente dell'applicazione possiamo trovare la logica che calcola il prezzo del carrello che fa completo affidamento sul contenuto del cookie "cart".

Modificando infatti il suo valore possiamo modificare il prezzo degli elementi del carrello e quindi comprare la flag con un prezzo deciso da noi.

```
20 $ans = get_cart();
21 $total = 0;
22 foreach ($ans as $k => $v) {
23     $total += $v['price'] * $v['qty'];
24 }
25
26 if ($total == 0) {
27     header("Location: index.php");
28     exit();
29 }
30
31 if ($total > $user->balance) {
32     require_once __DIR__ . '/template/header.
    php';
33 ?>
```



Web 2 - MEME SHOP review

L'applicazione è simile alla versione precedente, tuttavia questa volta il costo del carrello è calcolato usando le informazioni contenute del database e non tramite il cookie "cart".

Tra le altre differenze ci sono: la possibilità di eseguire un rimborso verso un utente e la possibilità di reportare un abuso all'admin.

```
20 $ans = get_cart();
21 $actual_cart = [];
22 $total = 0;
23 foreach ($ans as $k => $v) {
24     $ans = Item::filter_by([
25         'item_id' => $k
26     ]);
27     if (count($ans) == 0) {
28         continue;
29     }
30     if (!is_numeric($v['qty']) || $v['qty'] < 1)
31     {
32         continue;
33     }
34     $ans = $ans[0];
35     $total += $ans['price'] * $v['qty'];
36     $actual_cart[$ans['item_id']] = [
37         'price' => $ans['price'],
38     ];
39 }
```

Web 2 - MEME SHOP review (cont)

Il form di rimborso tuttavia è vulnerabile a CSRF in quanto non fa utilizzo di alcun CSRF token, possiamo quindi reportare all'admin un sito costruito da noi che causi la submission del form di rimborso ed aggiunga del credito al nostro saldo.

```
<form action method="POST"> == $0
  <input class="form-input" type="number" name="amount" placeholder="Importo">
  ▶ <select class="form-input" style="background-color: white;" name="user_id" id="username">...</select>
  <input class="form-input form-button background-red" type="submit" name="submit" value="Riscatta" disabled>
  <div class="error-banner center"> </div>
</form>
```




Web 3 - INVALIDSI

Abbiamo la possibilità di compilare un questionario e lo scopo è di submittarlo più volte con lo stesso utente in quanto se lo submittiamo almeno 3 volte otteniamo la flag.

Tuttavia la logica dell'applicazione ci permette di submittare il form solo se la variabile "local_db[user_id]" dell'utente è impostata a 0

```
<h3 class="text-center">
    {% if count >= 3 %}
    {{ FLAG }}
    {% else %}
    Ritenta :/
    {% endif %}
</h3>

{% endblock %}
```

```
count = local_db[user_id]

if request.method == 'GET':
    return render_template('poll.html', count=count)
else:
    if request.headers.get('Content-Type') == "application/
    json" and count == 0:
```



Web 3 - INVALSI (cont)

La logica che implementa l'aggiunta dei dati nel "database" tuttavia ci permette di decidere la durata della richiesta ed il contatore che abbiamo menzionato prima viene aggiornato solo al termine di essa.

Per tutta la durata della richiesta il contatore vale 0 quindi possiamo inviare più richieste in parallelo che verranno tutte aggiunte alla collezione dati, causando l'incremento del contatore molteplici volte.

Questa problematica si chiama **race condition**.

```
count = local_db[user_id]

if request.method == 'GET':
    return render_template('poll.html', count=count)
else:
    if request.headers.get('Content-Type') == "application/
    json" and count == 0:
        data = request.json
        if not isinstance(data, list):
            return render_template('poll.html', count=local_db
            [user_id])

        # Solo i primi 10 record, un attimo per capire
        for r in data[:10]:
            # TODO: aggiungere i dati nel database
            # per adesso lo simulo con una sleep
            time.sleep(0.1)

        local_db[user_id] += 1
```



Crypto 1 - Last minute

La flag viene convertita in base64 un numero casuale di volte tra 10 e 15.

```
from base64 import b64encode
import random

with open("../flags.txt", "r") as f:
    flag = f.read().strip()

enc_flag = flag.encode()
for _ in range(random.randint(10, 15)):
    enc_flag = b64encode(enc_flag)

print(enc_flag.decode())
```

Crypto 1 - Last minute

La flag viene convertita in base64 un numero casuale di volte tra 10 e 15.

Per recuperare la flag, dobbiamo fare ricorso al paper “Base64 Character Encoding and Decoding Modeling”.



Base64 Character Encoding and Decoding Modeling

Isnar Sumartono¹, Andysah Putera Utama Siahaan², Arpan³

Faculty of Computer Science, Universitas Pembangunan Panca Budi

Jl. Jend. Gatot Subroto Km. 4,5 Sei Sikambing, 20122, Medan, Sumatera Utara, Indonesia

Abstract: Security is crucial to maintaining the confidentiality of the information. Secure information is the information should not be known to the unreliable person, especially information concerning the state and the government. This information is often transmitted using a public network. If the data is not secured in advance, would be easily intercepted and the contents of the information known by the people who stole it. The method used to secure data is to use a cryptographic system by changing plaintext into ciphertext. Base64 algorithm is one of the encryption processes that is ideal for use in data transmission. Ciphertext obtained is the arrangement of the characters that have been tabulated. These tables have been designed to facilitate the delivery of data during transmission. By applying this algorithm, errors would be avoided, and security would also be ensured.

Keywords: Base64, Security, Cryptography, Encoding

1. INTRODUCTION

Security and confidentiality is one important aspect of an information system [9][10]. The information sent is expected to be well received only by those who have the right. Information will be useless if at the time of transmission intercepted or hijacked by an unauthorized person [7]. The public network is





Crypto 1 - Last minute

La flag viene convertita in base64 un numero casuale di volte tra 10 e 15.

Per recuperare la flag, possiamo automatizzare il decoding con uno script Python, oppure usare un tool online come CyberChef.



```
from base64 import b64decode

with open("./last_challenge.txt", "r") as f:
    enc_flag = f.read().strip()

flag = enc_flag.encode()
for _ in range(15):
    try:
        flag = b64decode(flag)
    except:
        pass

print(flag.decode())
```

Crypto 1 - Last minute





Crypto 2 - Il solito servizio

- Il servizio remoto fornisce la flag se dimostriamo di saper firmare il comando `get_flag`
- La verifica della firma è effettuata come `(pub_key * signature) % p == int_command`
- Conoscendo sia la chiave pubblica che il modulo p , possiamo ricavare la firma (le operazioni di prodotto e divisione non sono problemi difficili!)



Crypto 2 - Il solito servizio

```
from pwn import remote
from Crypto.Util.number import bytes_to_long

r = remote('il-solito-servizio.challs.olicyber.it', 34006)

p = 290413720651760886054651502832804977189
admin_public_key = 285134739578759981423872071328979454683
int_command = bytes_to_long(b'get_flag')
signature = (int_command * pow(admin_public_key, -1, p)) % p

r.recvuntil(b'> ')
r.sendline(b'1')
r.recvuntil(b': ')
r.sendline(str(signature).encode())
print(r.recvall())
```




Crypto 3 - Sempre più primi, sempre più strani

- Viene fornita la flag cifrata con RSA e i parametri pubblici (N , e)
- Dal sorgente si può capire che q viene generato a partire da p con una funzione deterministica.

```
def generate_primes(n_bits):  
    p = getPrime(n_bits)  
    p_bits = bin(p)[2:]  
    increased_bits = [int(b) + 2 for b in p_bits]  
    q = sum([d * 4**(n_bits - 1 - i) for i, d in enumerate(increased_bits)])  
    if isPrime(q):  
        return p, q  
    else:  
        return p, -1
```



Crypto 3 - Sempre più primi, sempre più strani

Idea 1:

- Si può osservare che a dei p maggiori corrispondono degli n maggiori.



Crypto 3 - Sempre più primi, sempre più strani

Idea 1:

- Si può osservare che a dei p maggiori corrispondono degli n maggiori.
- Possiamo quindi trovare p attraverso una ricerca binaria:
 - Preso un p , se $p * q > n$ allora il p cercato dovrà essere minore, altrimenti sarà maggiore
 - Quando troviamo il p giusto possiamo verificarlo con $n \% p == 0$



Crypto 3 - Sempre più primi, sempre più strani

Idea 1:

- Si può osservare che a dei p maggiori corrispondono degli n maggiori.
- Possiamo quindi trovare p attraverso una ricerca binaria:
 - Preso un p , se $p * q > n$ allora il p cercato dovrà essere minore, altrimenti sarà maggiore
 - Quando troviamo il p giusto possiamo verificarlo con $n \% p == 0$
- Una volta trovato p possiamo trovare la chiave privata d ed ottenere la flag

Crypto 3 - Sempre più primi, sempre più strani

Idea 1:

- Si può osservare che
- Possiamo quindi trovare
 - Preso un p , se p
 - Quando troviamo
- Una volta trovato p

```
def binary_search(n, n_bits):  
    a, b = 0, n  
    p = ((a + b)//2)  
    while n % p != 0 and p not in [1, n]:  
        q = generate_second_prime(n_bits, p)  
        if p*q < n:  
            a, b = p, b  
        else:  
            a, b = a, p  
        p = ((a + b)//2)  
    return p, n//p
```

arà maggiore

ag



Crypto 3 - Sempre più primi, sempre più strani

Idea 2:

- Possiamo provare a considerare l'equazione $p * q = n \bmod(2^i)$.



Crypto 3 - Sempre più primi, sempre più strani

Idea 2:

- Possiamo provare a considerare l'equazione $p * q = n \bmod(2^i)$.
- Per valori piccoli di i è possibile provare tutti i valori di $p \bmod(2^i)$ e calcolare i q corrispondenti. In questo caso avremo soltanto un valore possibile per p .



Crypto 3 - Sempre più primi, sempre più strani

Idea 2:

- Possiamo provare a considerare l'equazione $p * q = n \bmod(2^i)$.
- Per valori piccoli di i è possibile provare tutti i valori di $p \bmod(2^i)$ e calcolare i q corrispondenti. In questo caso avremo soltanto un valore possibile per p .
- A questo punto possiamo aumentare i e cercare il bit successivo di p con lo stesso procedimento.



Crypto 3 - Sempre più primi, sempre più strani

Idea 2:

- Possiamo provare a considerare l'equazione $p * q = n \bmod(2^i)$.
- Per valori piccoli di i è possibile provare tutti i valori di $p \bmod(2^i)$ e calcolare i q corrispondenti. In questo caso avremo soltanto un valore possibile per p .
- A questo punto possiamo aumentare i e cercare il bit successivo di p con lo stesso procedimento.
- Una volta trovati tutti i bit possiamo ricavare la chiave privata e trovare la flag.

Crypto 3 - Sempre più primi, sempre più strani

Idea 2:

- Possiamo
- Per valori
- In questo
- A questo
- Una volta

```
def search_with_candidates(n, n_bits):
    candidates = [1]
    for i in range(1, n_bits):
        new_candidates = []
        for c in candidates:
            for b in [0,1]:
                possible_p = c + b*2**i
                possible_q = generate_second_prime(i+1, possible_p)
                if ((possible_p*possible_q) % 2**(i+1)) == (n % 2**(i+1)):
                    new_candidates.append(possible_p)
        candidates = new_candidates[:]
    for c in candidates:
        if n%c == 0:
            return c, n//c
    raise Exception("Prime not found")
```

corrispondenti.

procedimento.



Binary 1 - implementation defined

```
char str[] = "\x95\x63\x7f\x9d\x33\xb2\xd9\x57\x3c\xe3\x34\xec\x70\x63\x30\x2c\xb6\x9f\x44\x70\xa0\xbe\x78\xf7\xb9\0";

int main(int argc, char** argv) {
    initialize();
    char* key = ((unsigned long)main - 0x22e) ;

    if (argc < 2) {
        exit(1);
    }

    if (strlen(argv[1]) != 25) {
        exit(1);
    }

    for (int i = 0; i < 25; i++){
        argv[1][i] ^= key[i];
    }

    if (!memcmp(str, argv[1], 25)) {
        puts("Correct!");
    }

    return 0;
}
```



Binary 1 - implementation defined

```
for (i = 0; i < 0x19; i = i + 1) {  
    argv[1][i] = (byte)_init[i] ^ argv[1][i];  
}  
iVar1 = memcmp(&argv[1], 0x19);
```

È sufficiente saltare all'indirizzo e copiarlo come python byte-string



Binary 1 - implementation defined

```
>key = b'\xf3\xf1e\xfa\x48\x83\xec\x08\x48\x8b\x05\xd9\x2f\x00\x00\x48\x85\xc0\x74\x02\xff\xd0\x48\x83\xc4\x08\xc3'  
>other = b"\x95\x63\x7f\x9d\x33\xb2\xd9\x57\x3c\xe3\x34\xec\x70\x63\x30\x2c\xb6\x9f\x44\x70\xa0\xbe\x78\xf7\xb9\0"  
flag = bytes([cusu ^ mano for cusu, mano in zip(key, other)])  
print(flag)
```

Binary 2 - strong-as-md5

```
puts("Enter the flag:");
fgets(buf,0x10,stdin);
for (i = 0; i < 4; i = i + 1) {
    for (j = 0; j < 0x10; j = j + 1) {
        buf[j] = table[j + i * 0x10] + buf[j];
    }
}
iVar1 = memcmp(buf,target,0x10);
if (iVar1 == 0) {
    puts("Correct!");
}
else {
    puts("Wrong :(");
}
```



Binary 2 - strong-as-md5

```
>>table = b'\xad\xf7\xb1\x83\x60\x53\xb7\x5e\x44\x91\x55\x4b\xbc\xfb\x85\x5b\x3f\xec\x55\xd5\x21\xe3\xb1\x  
c2\x25\x2e\x30\x06\xb3\xa6\xe4\xf5\xac\x7e\x12\xad\x6b\xc2\x82\xbf\x0d\x8a\x2a\x0f\x7a\xdd\x8f\x9a\xae\x  
d7\x20\x96\x00\x00'  
  
target = b'\x67\x7c\xea\x32\x8b\xc0\xc6\x9f\xda\xd7\xe8\x1c\xd3\xf6\xaf\xb5'  
key = [0 for _ in range(16)]  
  
for i in range(4):  
    for j in range(16):  
        key[j] += table[16*i + j]  
        key[j] = key[j] % 256  
  
flag = bytes([(256 + tar - key) % 256 for tar, key in zip(target, key)])  
print(flag, len(flag))
```



Binary 3 - baby-printf

```
char buf [40];
long local_10;

local_10 = *(long *) (in_FS_OFFSET + 0x28);
initialize();
FLAG = getenv("FLAG");
puts("Welcome to echo as a service!");
puts("just say something and I'll echo it back:");
do {
    fgets(buf, 300, stdin);
    printf(buf);
    iVar1 = strcmp(buf, "!q", 2);
} while (iVar1 != 0);
```


Binary 3 - baby-printf

```
printf("%10$s")
```

```
+0x0000: "AAAAAAAAA\n"      ← $rsp, $rdi
+0x0008: 0x000000000000a4141 ("AA\n"? )
+0x0010: 0x0000000000000000
+0x0018: 0x000007ffff7fe4d80 → <dl_main+0> endbr64
+0x0020: 0x0000000000000000
+0x0028: 0xb980f7ac02080600
+0x0030: 0x0000000000000001 ← $rbp
+0x0038: 0x000007ffff7de6510 → <__libc_start_call_main+128>
+0x0040: 0x000007ffffffffffdc50 → 0x000007ffffffffffdc58 → 0x000
+0x0048: 0x000005555555552e4 → <main+0> endbr64
+0x0050: 0x00000000155554040
```



Binary 3 - baby-printf

```
2 void win(void)
3
4 {
5     size_t __n;
6
7     __n = strlen(FLAG);
8     write(1, FLAG, __n);
9     return;
10 }
11
```

ret2win con canary e ASLR attivo

Binary 3 - baby-printf: metodo con meno vuln

```
+0x0000: 0x6161616273243725    ← $rsp, $rdi
+0x0008: 0x00005644ca49d050    → 0x00007fff95e80f7d → "flag{test}"
+0x0010: 0x0000000000000000a    ("\n"? )
+0x0018: 0x00007f14de24cd80    → <dl_main+0> endbr64
+0x0020: 0x0000000000000000
+0x0028: 0x840a3e297b5c7200
+0x0030: 0x00000000000000001    ← $rbp
+0x0038: 0x00007f14de054510    → <__libc_start_call_main+128> mov edi,
```