



# Sigurnost računala i sustava - labovi

## Lab 1: Man-in-the-middle attack (ARP spoofing)

- ARP protokol - koristi se za otkrivanje mac adresa i mapiranje sa pripadajućom Ip adresom u LAN mreži.

Korsnik na **Station-1** ako želi komunicirati sa korisnikom na **Station-2** odašilje poruku svima u mrežu "Ti, na IP adresi 172.24.04, koja je tvoja MAC adresa?". **Evil-station** iskorištava ranjivost sustava tako da posebno kreiranim ARP porukama prebacuje poruke koje **Station-1** odašilje preko switcha da umjesto rootera idu na računalo Evil.station na način da u ARP cache memoriju korisnika station-1 upiše svoju MAC adresu. To je moguće zbog toga što je već na **Stationu-1** zapamćena Ip adresa od **Station-2** sa svojom MAC adresom, koju korisnik na **Evil-station** mijena u svoju. Na taj način su mogući **DoS** i **Ros** napadi.

[https://excalidraw.com/#json=4981702338281472,W-i-XeDnIN\\_kHI71FNXV5w](https://excalidraw.com/#json=4981702338281472,W-i-XeDnIN_kHI71FNXV5w)

## Korištene linije koda

- git clone <https://github.com/mcagali/SRP-2021-22>
- cd SRP-2021-22/arp-spoofing/
- \$ docker exec -it station-1 bash
- ping - send ICMP ECHO\_REQUEST to network hosts
- tcpdump - dump traffic on a network
- arpspoof - intercept packets on a switched LAN
- netcat - arbitrary TCP and UDP connections and listens

U ovoj vježbi smo pokazali kako se može narušiti integritet računala povezanim LAN mrežom ,tj pokazali ranjivosti ARP protokola.

## Lab 2: Passive sniffing and analyzing network traffic using Wireshark

- **Wireshark** - alat koristan za networking i analizu podataka mrežnog prometa, također ga mogu koristiti hakeri za "passive sniffing". U vježbi 2 pokazali smo ranjivosti i opasnosti mreža koje koriste primitivne vrste kriptiranja podataka koji se prenose wi-fi mrežom.

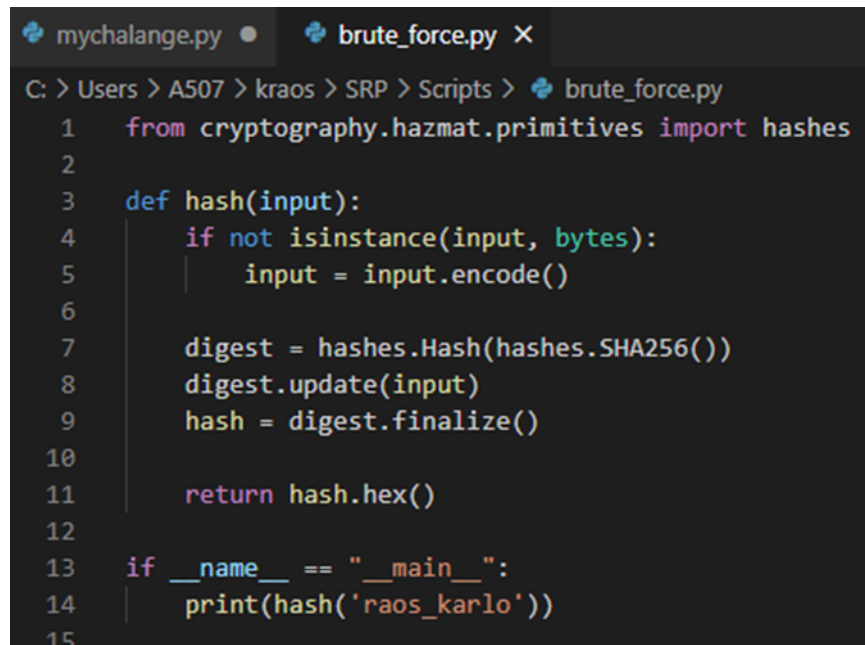
Spojili smo se na interni server našeg laboratorija te smo u direktoriju **SRP** kreirali virtualno okruženje te smo aktivirali direktorij **Scripts**. Zatim u **Scripts** instaliramo cryptography biblioteku iz jezika python.

```
Command Prompt - python3 x Command Prompt x + -
C:\Users\A507>cd kraos
C:\Users\A507\kraos>python3 --version
Python 3.9.7
C:\Users\A507\kraos>python3 venv -m SRP
python3: can't open file 'C:\Users\A507\kraos\venv': [Errno 2] No such file or directory
C:\Users\A507\kraos>python3 -m venv SRP
C:\Users\A507\kraos>cd SRP
C:\Users\A507\kraos\SRP>cd Scripts
C:\Users\A507\kraos\SRP\Scripts>activate
(SRP) C:\Users\A507\kraos\SRP\Scripts>pip install cryptography
Collecting cryptography
  Using cached cryptography-35.0.0-cp36-abi3-win_amd64.whl (2.1 MB)
Collecting cffi>=1.12
  Using cached cffi-1.15.0-cp39-cp39-win_amd64.whl (180 kB)
Collecting pycparser
  Using cached pycparser-2.20-py2.py3-none-any.whl (112 kB)
Installing collected packages: pycparser, cffi, cryptography
Successfully installed cffi-1.15.0 cryptography-35.0.0 pycparser-2.20
WARNING: You are using pip version 21.2.3; however, version 21.3.1 is available.
You should consider upgrading via the 'C:\Users\A507\kraos\SRP\Scripts\python.exe -m pip install --upgrade pip' command.
(SRP) C:\Users\A507\kraos\SRP\Scripts>
```

Sljedeći korak bio je pokazati kako se generira enkripcijski ključ za naše poruke.

```
>>> Fernet.generate_key()
b'LiTLBC_G9P8DhKZlSEKmm8THZK8ZN4sUP316zCZ8fRE='
>>> Fernet.generate_key()
b'_OgZ3X4v9M3-lAts29f9lu-Ia7tTQuLyQWvHS25Kz-c='
>>> key = Fernet.generate_key()
>>> key
b'9VdPzRlVybta0UmHmentxkvuBq3pZfvnuHtUbdWxXQk='
>>>
```

kreiralo smo **plaintext** "hello world" te smo taj tekst enkriptirali sa nasim ekripcijskim ključem i pretvorili ga na taj način u **ciphertext**. U pythonu smo zatim nas kreirali **hash** verziju naseg ciphertexta ,hash funkcija pretvara bilo koji tekst u tekst fiksne velicine koji sada nazvima **hash value**.



```

mychallenge.py ●  brute_force.py ✕
C: > Users > A507 > kraos > SRP > Scripts > brute_force.py
1  from cryptography.hazmat.primitives import hashes
2
3  def hash(input):
4      if not isinstance(input, bytes):
5          input = input.encode()
6
7      digest = hashes.Hash(hashes.SHA256())
8      digest.update(input)
9      hash = digest.finalize()
10
11     return hash.hex()
12
13 if __name__ == "__main__":
14     print(hash('raos_karlo'))
15

```

Zatim cemo našu datoteku u nasoj datoteci **brute\_force.py** kreirati funkciju brute force koja ce nasu enkriptiranu poruku dekreptirati na način da će to pokušati sa svakim ključem te u prompt-u ispisivati svaku 10 000 iteraciju dok ne dođe do onog odgovarajućeg ključa. U koliko je ključ nađen rezultat ciphertexta će se otvoriti kao datoteka slika koju smo dali ime **BINGO.png** (.png jer vec znamo da se radi o nekakvoj slici).

```

def brute_force():
    filename = "7e51a305a57172fd2cf25f9435d00435f1c87b358665f1f93aded3eeec2b9ba0.encrypted"
    with open(filename, "rb") as file:
        ciphertext = file.read()
    ctr = 0
    while True:
        key_bytes = ctr.to_bytes(32, "big")
        key = base64.urlsafe_b64encode(key_bytes)
        if not (ctr + 1) % 1000:
            print(f"[*] Keys tested: {ctr + 1:},", end="\n")
            # Now initialize the Fernet system with the given key
            # and try to decrypt your challenge.
            # Think, how do you know that the key tested is the correct key
            # (i.e., how do you break out of this infinite loop)?
            try:
                plaintext = Fernet(key).decrypt(ciphertext)
                header = plaintext[:32]

                if test_png(header):
                    print(f"[+] KEY FOUND: {KEY}")

                    with open("BIGNO.PNG", "wb") as file:
                        file.write(plaintext)
                    break
            except Exception:
                pass

        ctr += 1

if __name__ == "__main__":
    brute_force()

```

Rezultat dekripcije je bila slika sa našim imenom i prigodna čestitka za uspješno otkrivanje poruke.