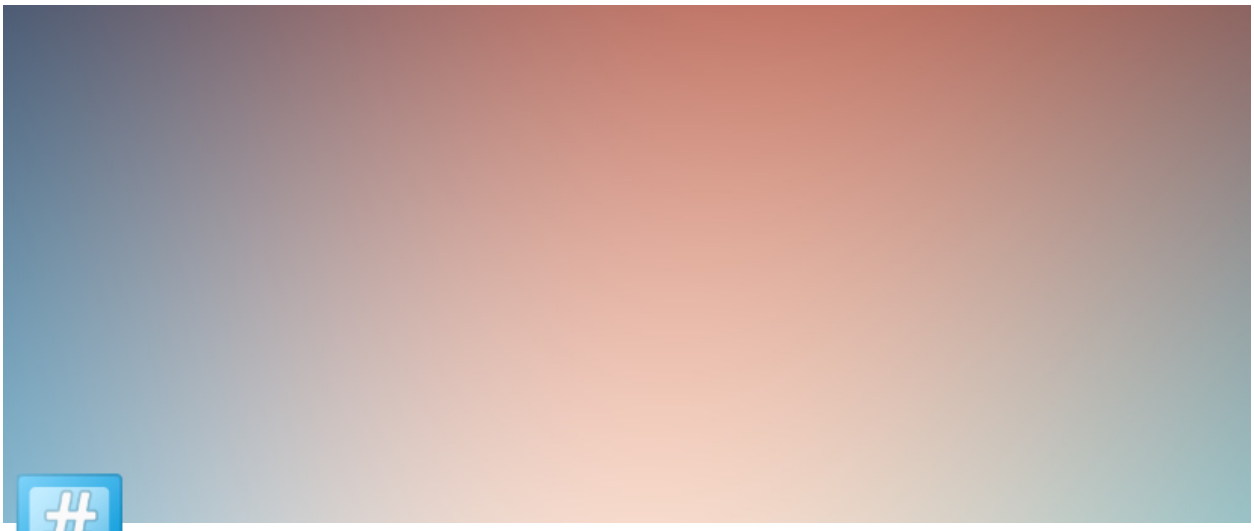

Izvještaj laboratorijske vježbe

4. Password-hashing

(iterative hashing, salt,memory-hard functions)

Zadatak

Upoznali smo se sa osnovnim konceptima sigurne pohrane lozinki. Usporedili smo klasične (brze) i specijalizirane spore (spore i memorijski zahtjevne) kriptografske hash funkcije za sigurnu pohranu lozinki i izvođenje derivacijskog ključa.

- Kao i do sad, dani kod smo otvorili u python datoteci i instalirali potrebne pakete odnosno module.

```
(lab4) C:\Users\MARIO\Desktop\SRP\lab4>pip install prettytable

(lab4) C:\Users\MARIO\Desktop\SRP\lab4>pip install passlib
```

- prvo smo se upoznali sa **"time_it"** funkcijom, koja za argument prima neku od hash funkcija i računa njeno vrijeme izvršavanja.

```
def time_it(function):
    def wrapper(*args, **kwargs):
        start_time = time()
        result = function(*args, **kwargs)
        end_time = time()
        measure = kwargs.get("measure")
        if measure:
            execution_time = end_time - start_time
            return result, execution_time
        return result
    return wrapper
```

- Kada smo pokrenuli program u virtualnom python okruženju kao rezultat smo dobili tablicu koja sadrži ime funckije i njeno prosječno vrijeme izvršavanja na 100 iteracija. Vidimo kako je AES dosta sporiji od hash funkcija MD5 i SHA256.

```
(lab4) C:\Users\MARIO\Desktop\SRP\lab4>python password_hash.py
+----------+----------------------+
| Function | Avg. Time (100 runs) |
+----------+----------------------+
| AES      |       0.00483        |
+----------+----------------------+


+----------+----------------------+
| Function | Avg. Time (100 runs) |
+----------+----------------------+
| HASH_MD5 |       3.4e-05        |
| AES      |       0.00483        |
+----------+----------------------+


+-------------+----------------------+
| Function    | Avg. Time (100 runs) |
+-------------+----------------------+
| HASH_MD5    |       3.4e-05        |
| HASH_SHA256 |       4.2e-05        |
| AES         |       0.00483        |
+-------------+----------------------+
```

- Zatim smo pratili vrijeme izvršavanja za funkciju **"linux_hash"** za 5000 rundi i 1000000(milijun) rundi. Ova funkcija koristi mehanizme **"iterative hashing"** i **"password salting"** koristeći funkciju **crypt (**u našem slučaju sha512_crypt()**).**

```
TESTS = [
        {
            "name": "AES",
            "service": lambda: aes(measure=True)
        },
        {
            "name": "HASH_MD5",
            "service": lambda: sha512(password, measure=True)
        },
        {
            "name": "HASH_SHA256",
            "service": lambda: sha512(password, measure=True)
        },
        {
            "name": "Linux CRYPTO 5k",
            "service": lambda: linux_hash(password, measure=True)
        },
        {
            "name": "Linux CRYPTO 1M",
            "service": lambda: linux_hash(password, rounds=10**6, measure=True)
        }

    ]
```

- Sada smo na terminalu dobili sljedeći sadržaj tablice. Vidimo kako je vrijeme izvršavanja **"Linux_hash"** funkcije po iteraciji za milijun rundi osjetno duže.

```
(lab4) C:\Users\MARIO\Desktop\SRP\lab4>python password_hash.py
+----------+----------------------+
| Function | Avg. Time (100 runs) |
+----------+----------------------+
| AES      |       0.000526       |
+----------+----------------------+


+----------+----------------------+
| Function | Avg. Time (100 runs) |
+----------+----------------------+
| HASH_MD5 |       4.3e-05        |
```

```
| AES      |       0.000526       |
+----------+----------------------+


+-------------+----------------------+
| Function    | Avg. Time (100 runs) |
+-------------+----------------------+
| HASH_SHA256 |       3.3e-05        |
| HASH_MD5    |       4.3e-05        |
| AES         |       0.000526       |
+-------------+----------------------+


+-----------------+----------------------+
| Function        | Avg. Time (100 runs) |
+-----------------+----------------------+
| HASH_SHA256     |       3.3e-05        |
| HASH_MD5        |       4.3e-05        |
| AES             |       0.000526       |
| Linux CRYPTO 5k |       0.007947       |
+-----------------+----------------------+


+-----------------+----------------------+
| Function        | Avg. Time (100 runs) |
+-----------------+----------------------+
| HASH_SHA256     |       3.3e-05        |
| HASH_MD5        |       4.3e-05        |
| AES             |       0.000526       |
| Linux CRYPTO 5k |       0.007947       |
| Linux CRYPTO 1M |       1.842686       |
+-----------------+----------------------+
```

Zaključak

Najčešća metoda za pohranu lozinki je **"password hashing"**. Jedan od **"offline guessing"** napada na ovakav sustav je **"pre-computed dictionary attack".** Radi po principu da se za listu kandidata za lozinku izračuna hash vrijednost i takvi parovi pohrane u tablicu. Cilj napadača je nekako naučiti hash vrijdenost žrtvine lozinke i takvu istu pokušati pronaći u svojoj tablici (pre-computed dictionary). Ukoliko mu to uspije, našao je i traženu lozinku. Mehanizmi koji demotiviraju napadača su gore spomenuti **"iterative hashing"** i **"password salting"**. Prvi mehanizam hash-ira lozinku ne jednom već n puta, čime se bitno usporava napadač i ograničava na robusnost njegovog hardvera. Drugi mehanizam zajedno sa lozinkom hash-ira i k-bitnu vrijednost **"salt"**.

Tako se napadač također usporava, ali se i onemogućava pojava dupliciranih lozinki. Ovi mehanizmi vode do povećanja sigurnosti našeg sustava, ali s druge strane smanjuju kvalitetu istog (npr. legitimni server verificira n-puta hash-irane lozinke i uzrokuje **"denial of service"**). Dakle, bitno je odrediti prioritete zahtjeva sustava i na temelju njih implementirati određenu zaštitu.