

ASSIGNMENT 3



Ryan Hanif Dwihandoyo
Bootcamp CS Batch 3

Cryptographic Failures (Kegagalan Kriptografi) adalah salah satu dari sepuluh risiko keamanan aplikasi web paling kritis menurut OWASP Top 10. Kegagalan ini terjadi ketika data sensitif tidak dilindungi dengan baik, baik saat disimpan (at rest) maupun saat dikirim (in transit). Praktikum ini bertujuan untuk mengidentifikasi, menganalisis, dan memberikan rekomendasi perbaikan terhadap beberapa contoh umum kegagalan kriptografi, seperti penyimpanan

password dalam bentuk plaintext, penggunaan algoritma hash yang lemah, dan transmisi data melalui protokol HTTP yang tidak aman.

1. Penyimpanan Password: Plaintext vs. Hash

Tujuan dari bagian ini adalah untuk membandingkan secara langsung risiko keamanan antara menyimpan

password sebagai teks biasa (plaintext) dengan menyimpannya dalam bentuk hash yang aman di dalam database.

Proses Praktikum

Membuat Database dan Tabel:

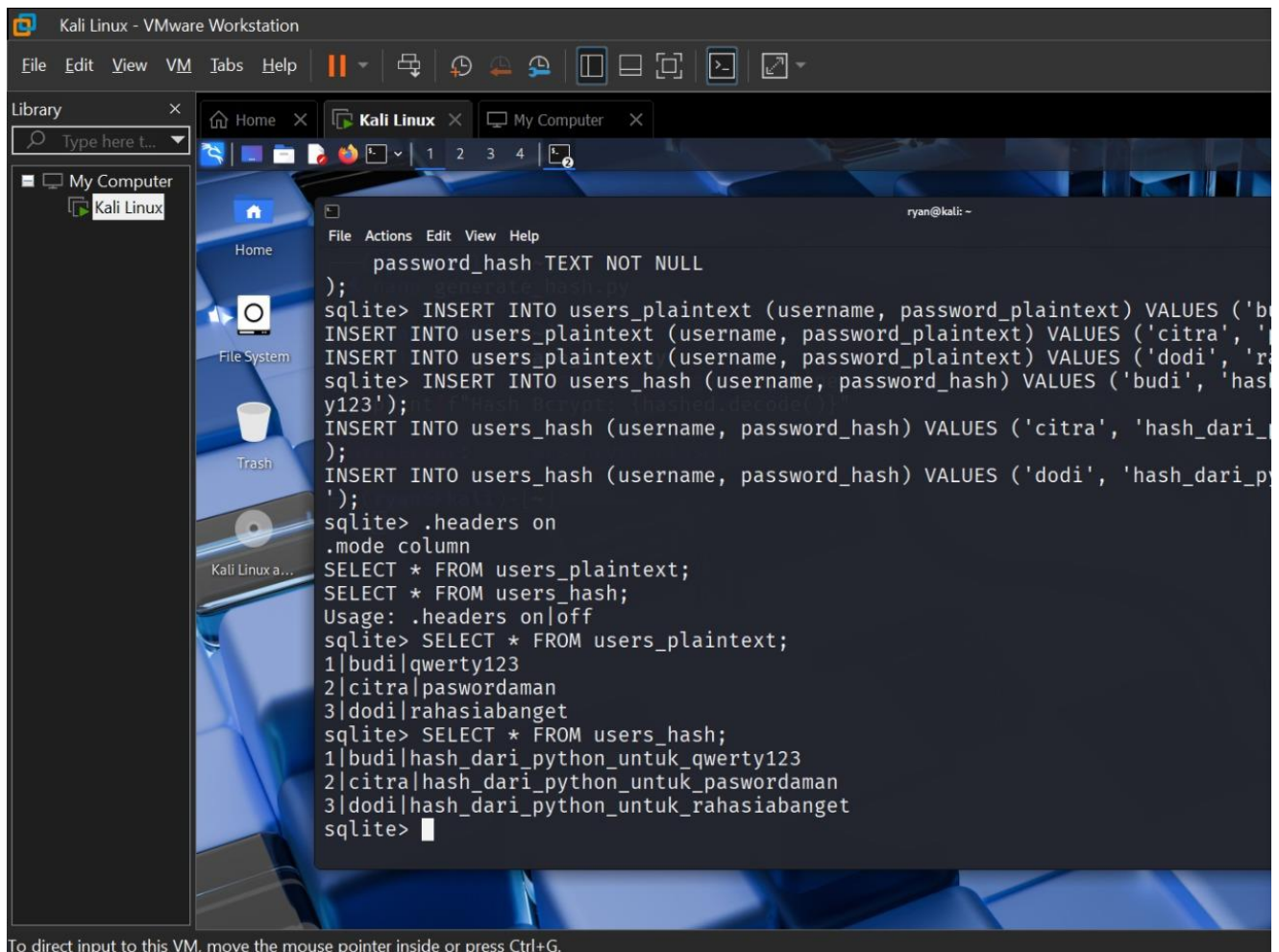
Kita akan menggunakan SQLite karena sederhana dan tidak memerlukan instalasi server. Buat dua tabel:

users_plaintext untuk menyimpan password apa adanya dan users_hash untuk menyimpan password yang sudah di-hash menggunakan bcrypt.

Menambahkan Data:

Kita akan menambahkan tiga akun ke dalam masing-masing tabel. Untuk

users_hash, kita perlu membuat hash dari password terlebih dahulu menggunakan Python.



Analisis Risiko

Jika terjadi kebocoran

database (database breach):

Tabel `users_plaintext`: Risiko sangat tinggi. Penyerang bisa langsung melihat dan menggunakan kombinasi username dan password semua pengguna. Ini membuka pintu untuk pengambilalihan akun, pencurian identitas, dan penyerang dapat mencoba kredensial yang sama di platform lain (credential stuffing).

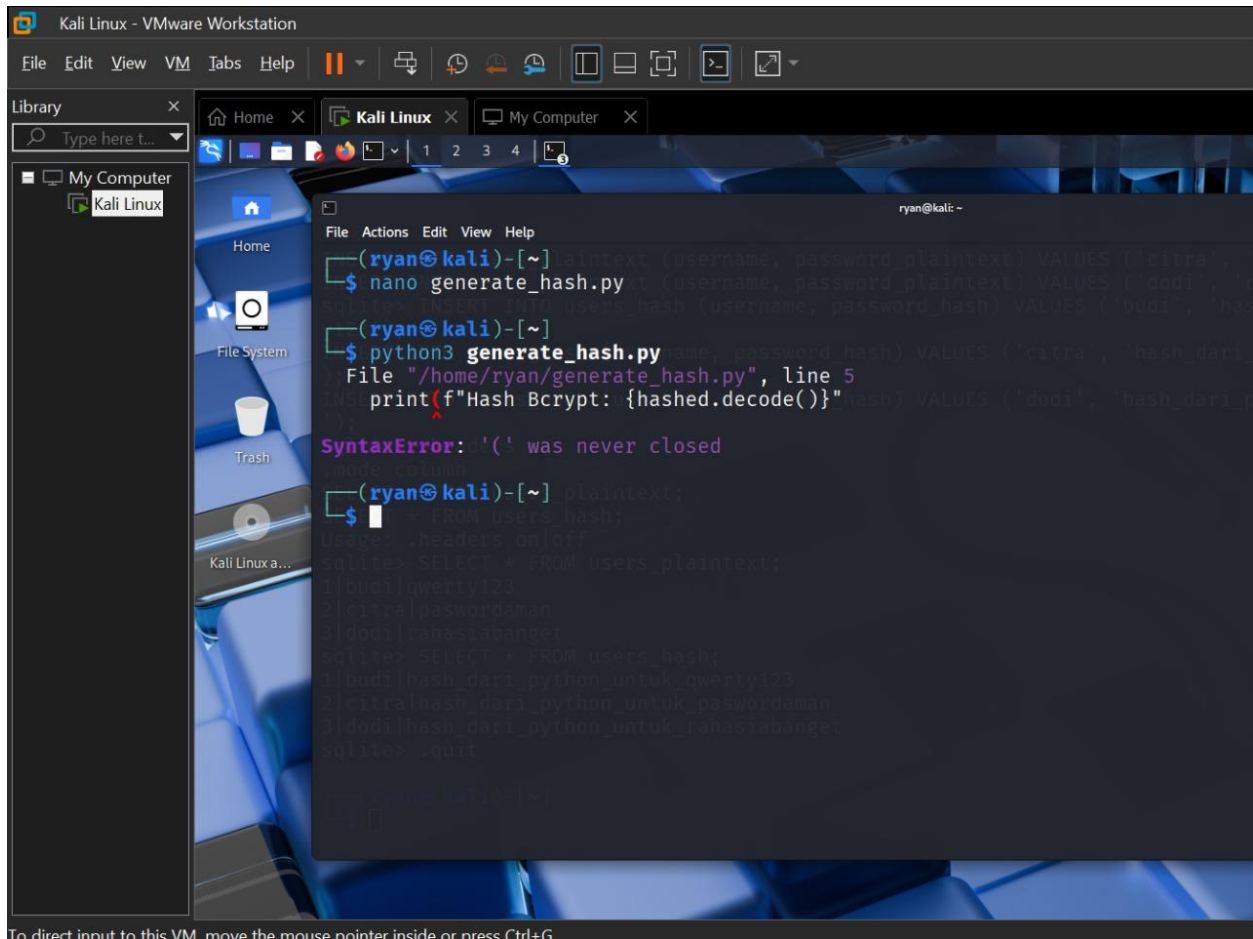
Tabel `users_hash`: Risiko jauh lebih rendah. Penyerang hanya mendapatkan rentetan karakter acak (hash). Karena kita menggunakan bcrypt, yang mengandung salt unik untuk setiap password, penyerang tidak bisa menggunakan rainbow table untuk memecahkannya dengan cepat. Mereka harus melakukan serangan brute-force pada setiap hash satu per satu, yang secara komputasi sangat mahal dan lambat.

2. Perbandingan Hashing: MD5 vs. bcrypt

Bagian ini bertujuan untuk mendemonstrasikan kelemahan algoritma hash lama (MD5) dibandingkan dengan algoritma modern (bcrypt).

Proses Praktikum

Generate Hash: Gunakan Python untuk membuat hash dari sebuah password (misal: ilovecybersecurity) menggunakan MD5 dan bcrypt.



```
File Actions Edit View Help
(ryan@kali)-[~]
$ nano generate_hash.py
(ryan@kali)-[~]
$ python3 generate_hash.py
File "/home/ryan/generate_hash.py", line 5
    print(f"Hash Bcrypt: {hashed.decode()}")
SyntaxError: '(' was never closed
(ryan@kali)-[~]
$ cat generate_hash.py
#!/usr/bin/env python3
import hashlib
import bcrypt
import sys

def main():
    username = input("Username: ")
    password = input("Password: ")

    # Generate MD5 hash
    md5_hash = hashlib.md5(password.encode()).hexdigest()

    # Generate bcrypt hash
    hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt())

    print(f"MD5 Hash: {md5_hash}")
    print(f"Hash Bcrypt: {hashed.decode()}")

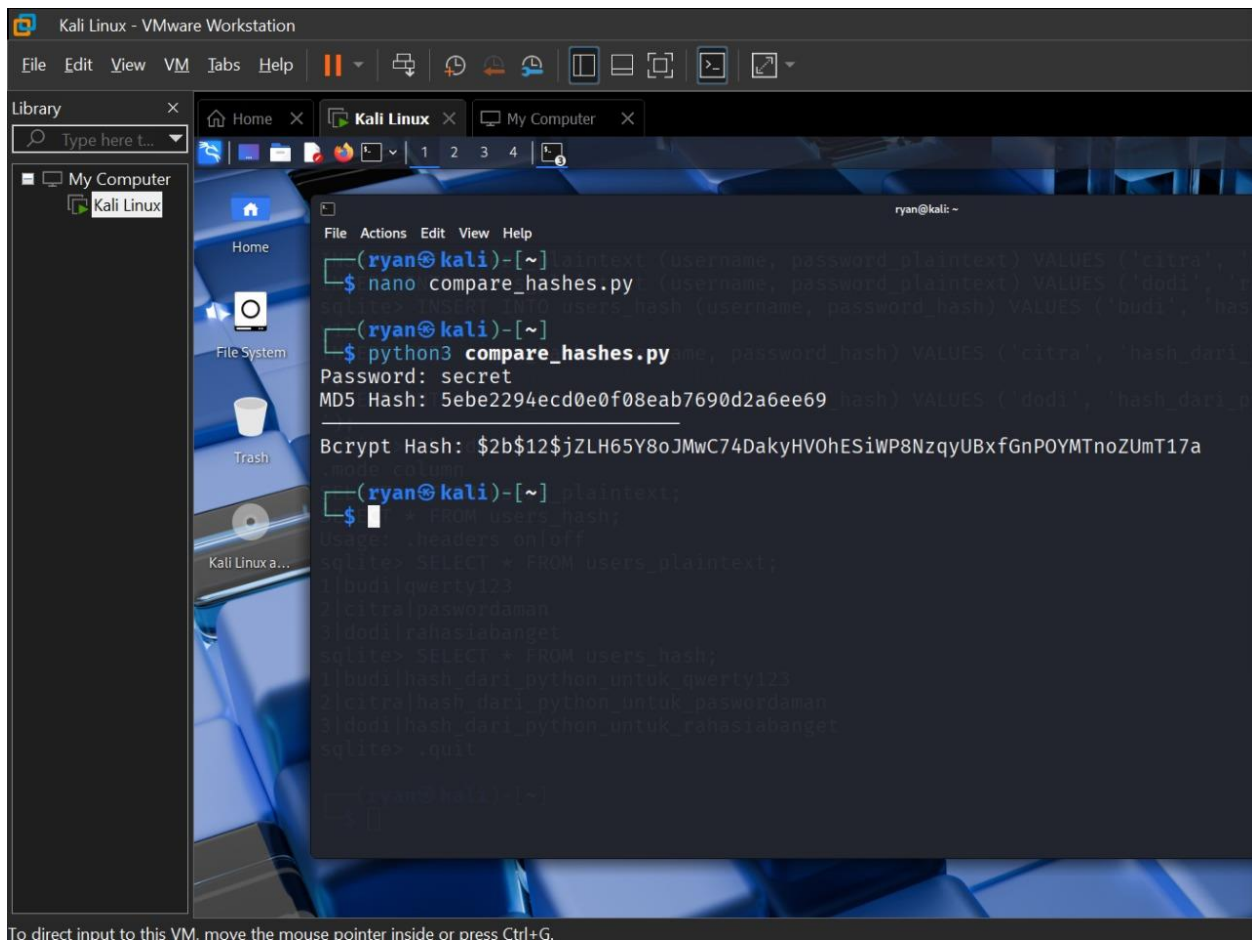
if __name__ == '__main__':
    main()
```

Analisis Peran Salt dan Work Factor pada bcrypt

Salt: *Salt* adalah data acak yang ditambahkan pada password sebelum proses hashing. bcrypt secara otomatis menghasilkan salt yang unik untuk setiap password. Fungsi utamanya adalah untuk menggagalkan serangan rainbow table. Karena setiap password memiliki salt yang berbeda, bahkan untuk password yang sama (misal, dua pengguna memakai password '123456'), hasil hash-nya akan benar-benar berbeda. Penyerang tidak bisa membuat satu tabel hash untuk semua kemungkinan password dan menggunakannya kembali.

Work Factor (Cost Factor): *Work factor* adalah parameter yang menentukan seberapa lambat dan intensif secara komputasi proses hashing bcrypt. Semakin tinggi angkanya (misalnya, 12 atau 14), semakin lama waktu yang dibutuhkan untuk menghasilkan satu hash. Ini adalah pertahanan yang sangat efektif melawan serangan brute-force. Jika satu percobaan hash membutuhkan waktu 100 milidetik, maka mencoba miliaran kombinasi akan memakan waktu bertahun-tahun, membuat serangan menjadi tidak praktis.

MD5 tidak memiliki kedua mekanisme ini, membuatnya cepat dan rentan terhadap rainbow table, sehingga sangat tidak aman untuk penyimpanan password.



The screenshot shows a Kali Linux virtual machine running in VMware Workstation. The terminal window displays the following commands and output:

```
(ryan@kali)-[~]
$ nano compare_hashes.py
(ryan@kali)-[~]
$ python3 compare_hashes.py
Password: secret
MD5 Hash: 5ebe2294ecd0e0f08eab7690d2a6ee69

Bcrypt Hash: $2b$12$jZLH65Y8oJMwC74DakyHVOhESiWP8NzqyUBxfGnPOYMTnoZUmT17a

(ryan@kali)-[~]
$ cat compare_hashes.py
#!/usr/bin/env python3
import sys
import sqlite3
import hashlib

def main():
    plaintext = sys.argv[1]
    password = sys.argv[2]
    hash = sys.argv[3]

    # Connect to the database
    conn = sqlite3.connect('users.db')
    cursor = conn.cursor()

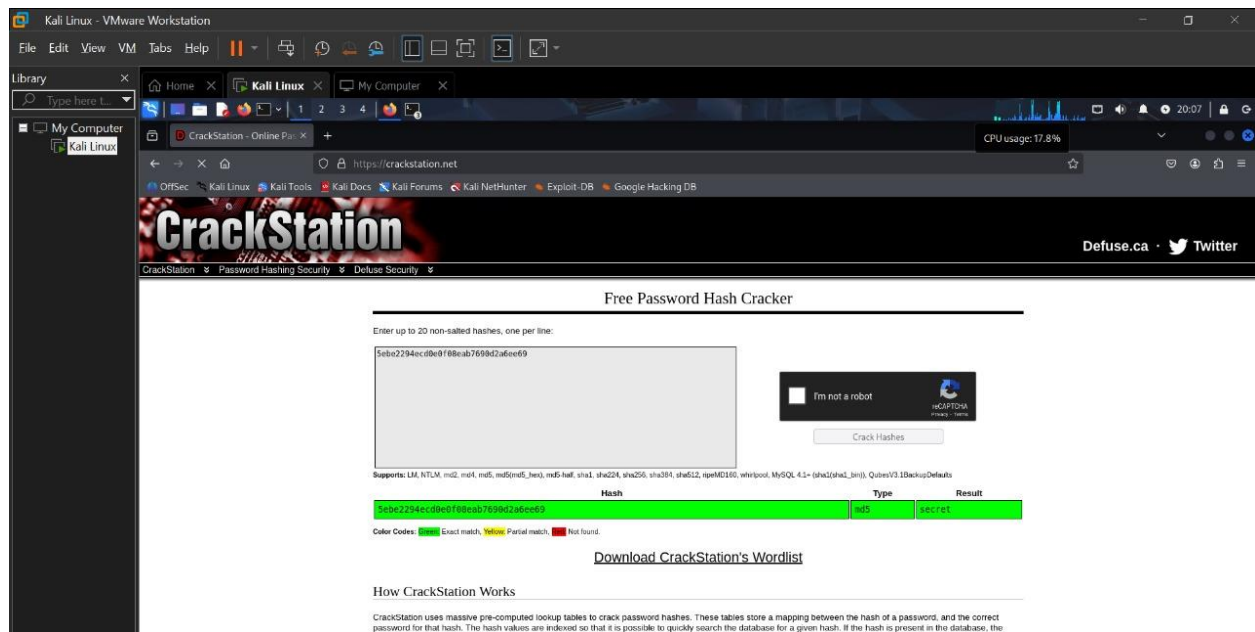
    # Query the database for the user's password and hash
    cursor.execute('SELECT * FROM users;')
    results = cursor.fetchall()

    # Find the user's password and hash
    for row in results:
        username, password, hash = row
        if password == plaintext:
            print(f'Password: {password}')
            print(f'Hash: {hash}')
            break

    # Close the database connection
    conn.close()

if __name__ == '__main__':
    main()
```

The terminal output shows the MD5 hash of the password 'secret' and the bcrypt hash of the password 'secret'. The script also queries the 'users' database for the user's password and hash.



3. Analisis Trafik Login: HTTP vs. HTTPS

Tujuan bagian ini adalah untuk menunjukkan secara praktis bagaimana data yang dikirim melalui HTTP dapat "diintip" (sniffing) dan bagaimana HTTPS melindunginya.

Proses Praktikum

Siapkan Tools: Instal Wireshark.

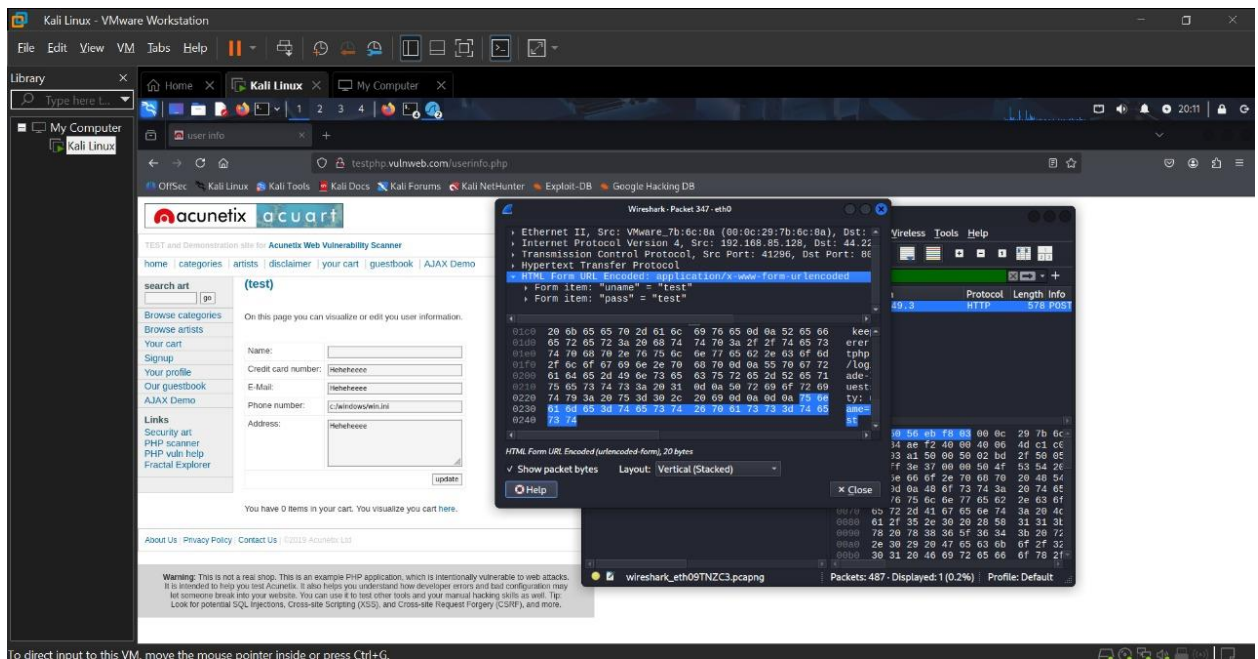
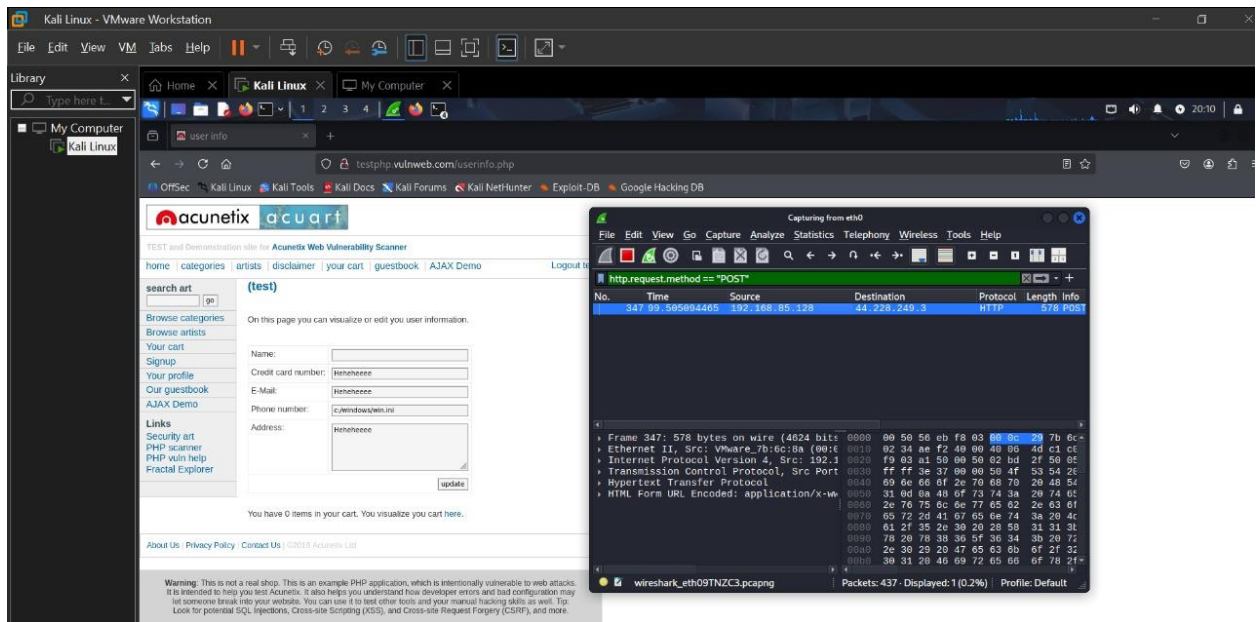
Cari Target: Temukan sebuah situs web yang masih menggunakan form login HTTP. (Catatan: Ini semakin sulit ditemukan. Anda bisa menggunakan situs uji coba seperti <http://testphp.vulnweb.com/login.php>).

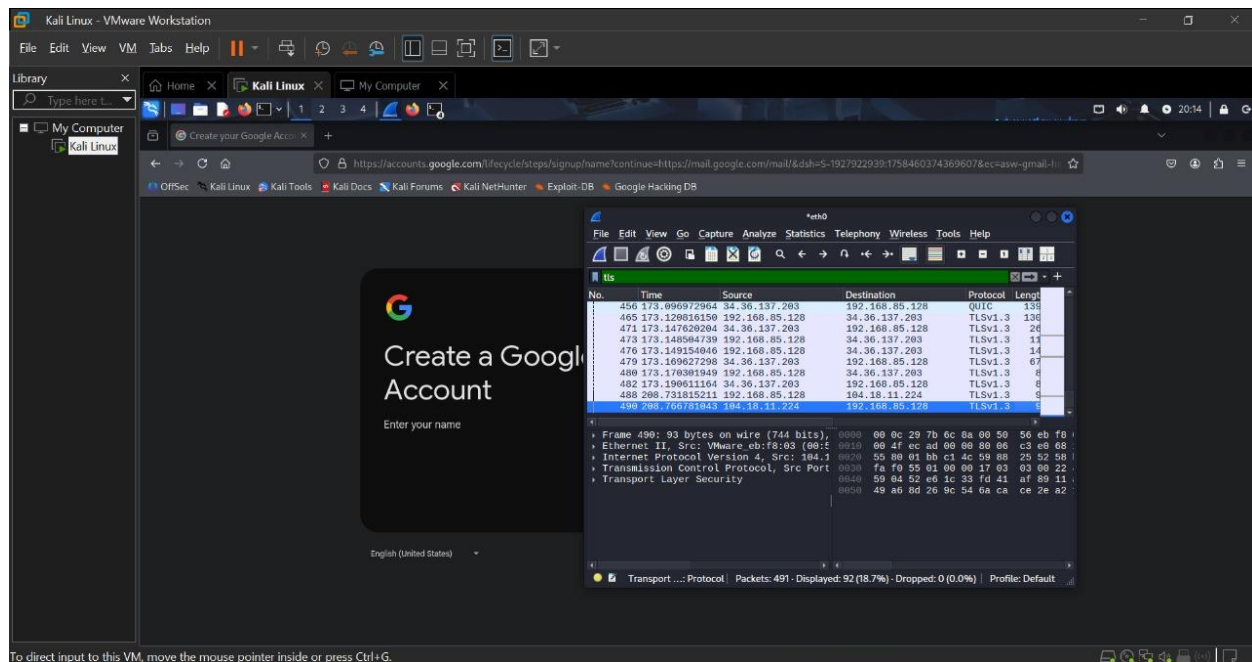
Buka Wireshark dan mulai capturing pada interface jaringan yang Anda gunakan (misal, Wi-Fi atau Ethernet). Buka browser dan akses situs login HTTP.

Di Wireshark, gunakan filter `http.request.method == "POST"`.

Masukkan username dan password palsu (misal, testuser / testpass), lalu klik login.

Hentikan capture di Wireshark. Temukan paket POST, klik kanan, pilih "Follow" -> "TCP Stream". Anda akan melihat kredensial dalam bentuk plaintext.





Rekomendasi Perbaikan

Berdasarkan temuan dari ketiga praktikum di atas, berikut adalah rekomendasi perbaikan yang spesifik dan praktis.

Temuan 1 (Password Plaintext):

Solusi: Jangan pernah menyimpan password dalam bentuk plaintext. Semua password pengguna wajib di-hash sebelum disimpan ke database.

Praktis: Gunakan algoritma hashing modern yang adaptif dan memiliki salt, seperti bcrypt atau Argon2 (pemenang kompetisi Password Hashing).

Temuan 2 (Hashing Lemah - MD5):

Solusi: Segera migrasikan semua hash password yang dibuat dengan algoritma lama (MD5, SHA1) ke algoritma yang kuat.

Praktis: Implementasikan strategi migrasi hash. Ketika pengguna login dengan password lama mereka, verifikasi menggunakan hash MD5. Jika berhasil, segera buat hash baru menggunakan bcrypt dan simpan di database, lalu hapus hash MD5 yang lama.

Temuan 3 (Transmisi via HTTP):

Solusi: Wajibkan penggunaan HTTPS di seluruh bagian situs web, bukan hanya pada halaman login.

Praktis

- :Dapatkan dan instal sertifikat SSL/TLS pada server web.
- Konfigurasi server untuk mengalihkan (redirect) semua trafik HTTP ke HTTPS secara otomatis (301 redirect).
- Implementasikan header HTTP Strict Transport Security (HSTS) untuk memaksa browser hanya berkomunikasi melalui HTTPS, mencegah serangan SSL stripping.

Kesimpulan

Praktikum ini telah menunjukkan dengan jelas dampak signifikan dari Cryptographic Failures terhadap keamanan data sensitif.

Ringkasan Hasil:

- Menyimpan password sebagai plaintext sama saja dengan menyerahkannya langsung kepada penyerang saat terjadi kebocoran data.
- Algoritma hash lama seperti MD5 sangat rapuh dan dapat dipecahkan dalam hitungan detik menggunakan teknik modern.
- Mengirimkan data melalui HTTP memungkinkan siapa saja di jaringan untuk mengintip dan mencuri informasi dengan mudah.
- Dampak: Kegagalan dalam menerapkan kriptografi yang benar dapat menyebabkan kompromi akun massal, pencurian data finansial, pelanggaran privasi, dan kerusakan reputasi yang parah bagi organisasi. Oleh karena itu, penerapan enkripsi yang kuat baik untuk data at rest (disimpan) maupun data in transit (dikirim) adalah pilar fundamental dalam keamanan siber.

