LAPORAN PROYEK AKHIR

WEB PENETRATION TESTING & BUG BOUNTY

Disusun oleh:

Malvin Wijaya

Batch 22

Diselenggarakan oleh:

JadiHacker 2025

DAFTAR ISI

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Seiring dengan pesatnya perkembangan teknologi digital, keamanan aplikasi web menjadi salah satu aspek krusial yang tidak dapat diabaikan. Banyaknya data sensitif yang dikelola oleh aplikasi web membuatnya menjadi target utama bagi para pelaku kejahatan siber. Oleh karena itu, kebutuhan akan praktisi keamanan siber yang kompeten untuk mengidentifikasi dan menanggulangi kerentanan semakin meningkat.

Menjawab kebutuhan tersebut, program Bootcamp Web Penetration Testing & Bug Bounty yang diselenggarakan oleh JadiHacker dirancang untuk mencetak talenta-talenta baru di bidang keamanan siber. Laporan ini disusun sebagai syarat kelulusan dan merupakan dokumentasi resmi dari pengerjaan Tugas Akhir (*Final Exam*) program tersebut.

Dokumen ini menyajikan secara rinci seluruh proses, metodologi, dan hasil dari pengerjaan serangkaian laboratorium (lab) keamanan web yang telah diberikan. Pengerjaan tugas ini bertujuan untuk menguji dan memvalidasi pemahaman serta keterampilan praktis peserta dalam mengidentifikasi, mengeksploitasi, dan mendokumentasikan berbagai jenis kerentanan secara profesional.

1.2 Tujuan dan Sasaran

Adapun tujuan dan sasaran dari penyusunan laporan tugas akhir ini adalah sebagai berikut:

1. Memenuhi salah satu syarat utama kelulusan dari program Bootcamp Web Penetration Testing & Bug Bounty yang diselenggarakan oleh JadiHacker.
2. Mendokumentasikan setiap langkah (*step-by-step*) yang dilakukan dalam menyelesaikan semua lab pada tugas akhir, lengkap dengan bukti berupa gambar pendukung.
3. Menunjukkan pemahaman mendalam mengenai konsep, dampak, dan cara mitigasi dari setiap kerentanan yang berhasil dieksploitasi.
4. Menjadi bukti kompetensi dalam menggunakan berbagai *tools* penetration testing yang relevan, seperti Burp Suite, SQLMap, dan Dirsearch, dalam skenario yang realistis.

1.3 Ruang Linkup Pengerjaan

Ruang lingkup pengerjaan dalam laporan ini terbatas pada penyelesaian dan dokumentasi lab-lab yang telah ditetapkan dalam ujian akhir. Lab-lab tersebut terbagi menjadi dua kategori, yaitu tugas wajib dan tugas opsional, dengan rincian sebagai berikut:

A. Tugas Wajib Pengerjaan dan pelaporan dari lab-lab berikut:

- Eksploitasi kerentanan SQL Injection untuk melakukan *bypass* login dan enumerasi database.
- Eksploitasi kerentanan SQL Injection untuk menemukan password dari database.

- Analisis dan eksploitasi seluruh kerentanan yang ada pada lab http://IP_LOCAL/vuln/3.
- Penyelesaian lab PortSwigger mengenai enumerasi konten database pada platform Oracle dan Non-Oracle.
- Penyelesaian seluruh lab PortSwigger pada kategori kerentanan Path Traversal.

B. Tugas Opsional Pengerjaan dan pelaporan dari salah satu kategori lab berikut:

- .GIT Vuln xsslabs (Dari Dirsearch hingga Extract)

BAB 2

PEMBAHASAN TUGAS WAJIB

2.1 SQL Injection: Bypass Login & Database Collection (Manual & SQLMap)

Aplikasi web ini memiliki kerentanan SQL Injection pada fungsionalitas login administrator. Kerentanan ini terjadi karena input pengguna pada field "Email Address" tidak disanitasi dengan benar sebelum digabungkan ke dalam *query* SQL. Akibatnya, seorang penyerang dapat memanipulasi *query* untuk melewati otentikasi dan mendapatkan akses tidak sah. Setelah kerentanan terkonfirmasi, celah yang sama dapat dieksploitasi lebih lanjut untuk membaca seluruh isi database menggunakan *tools* otomatis.

Url Affected/Endpoint	http://127.0.0.1:2222/lab/sql-injection/post-login/index.php
Severity	**Critical**
CWE	CWE-89: *Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*
OWASP	A03:2021-Injection
CVSS Score	9.8
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

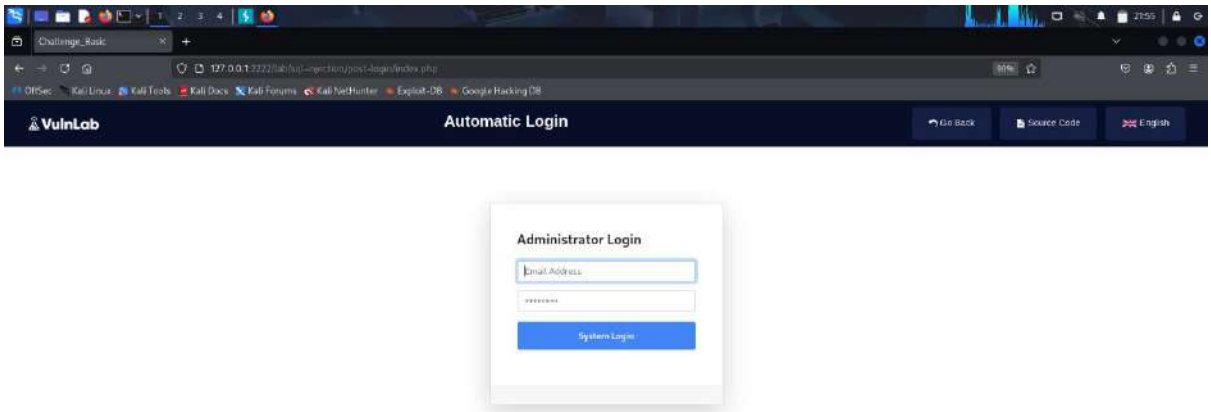
Jika dieksploitasi, dampak dari kerentanan ini sangat serius, antara lain:

1. Akses Administratif Tidak Sah: Penyerang dapat login sebagai administrator, memberikan mereka kontrol penuh atas aplikasi.
2. Kompromi Database Penuh: Penyerang dapat melihat, mengubah, atau menghapus seluruh data sensitif yang tersimpan di dalam database, termasuk kredensial pengguna, informasi pribadi, dan data bisnis lainnya.
3. Pengambilalihan Sistem: Bergantung pada konfigurasi database, kerentanan ini bisa mengarah pada eksekusi perintah pada sistem operasi (*Remote Code Execution*).

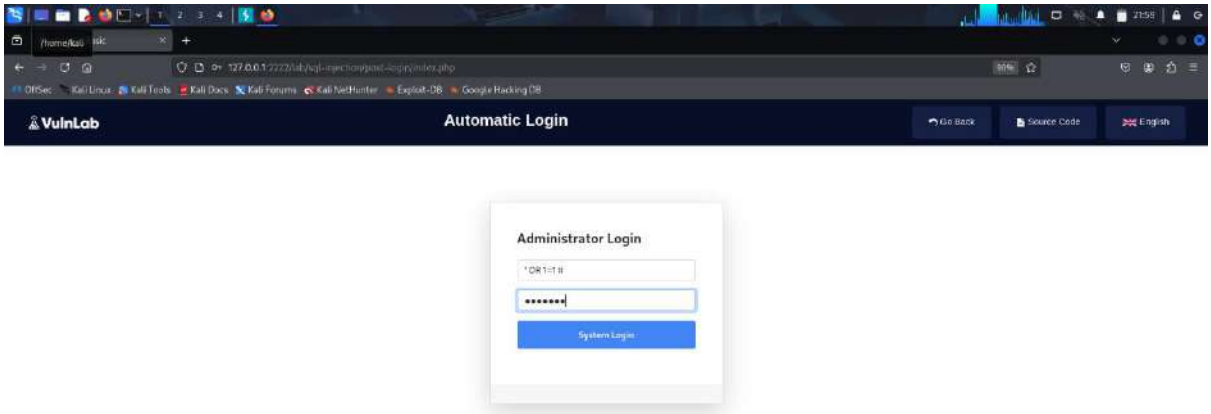
Proses eksploitasi dilakukan dalam dua tahap utama: bypass otentikasi secara manual untuk membuktikan adanya kerentanan, dilanjutkan dengan enumerasi database menggunakan SQLMap untuk menunjukkan dampak penuhnya.

Tahap 1 - Bypass Otentikasi Manual

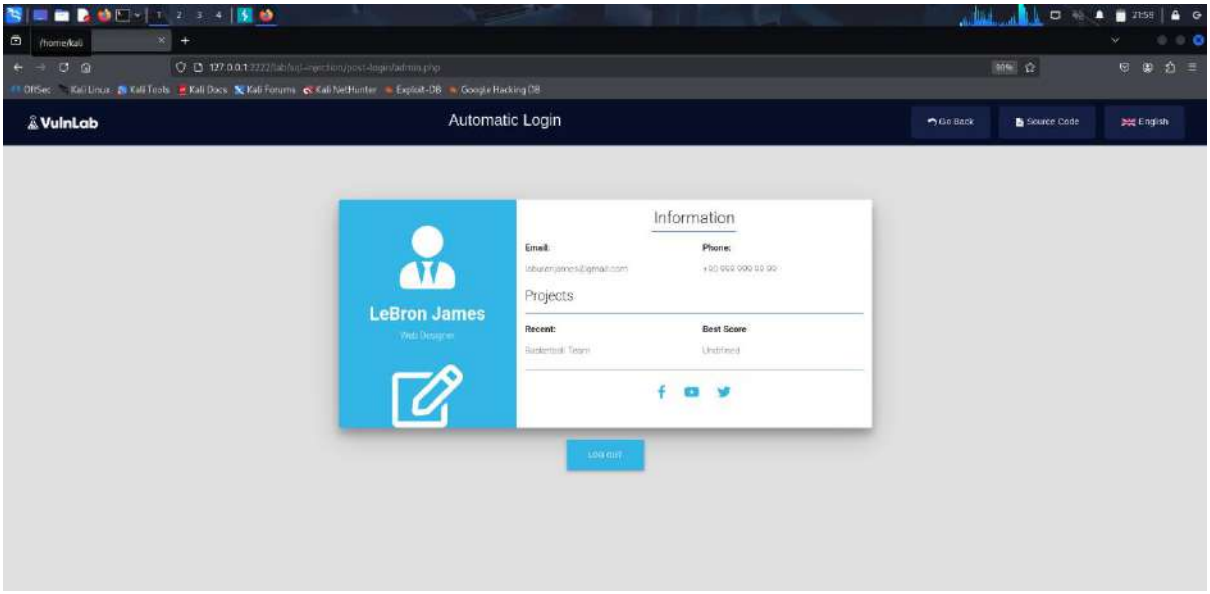
1. Buka halaman login administrator pada URL yang terdampak.



2. Pada field "Email Address", masukkan *payload* berikut: ' OR 1=1 #

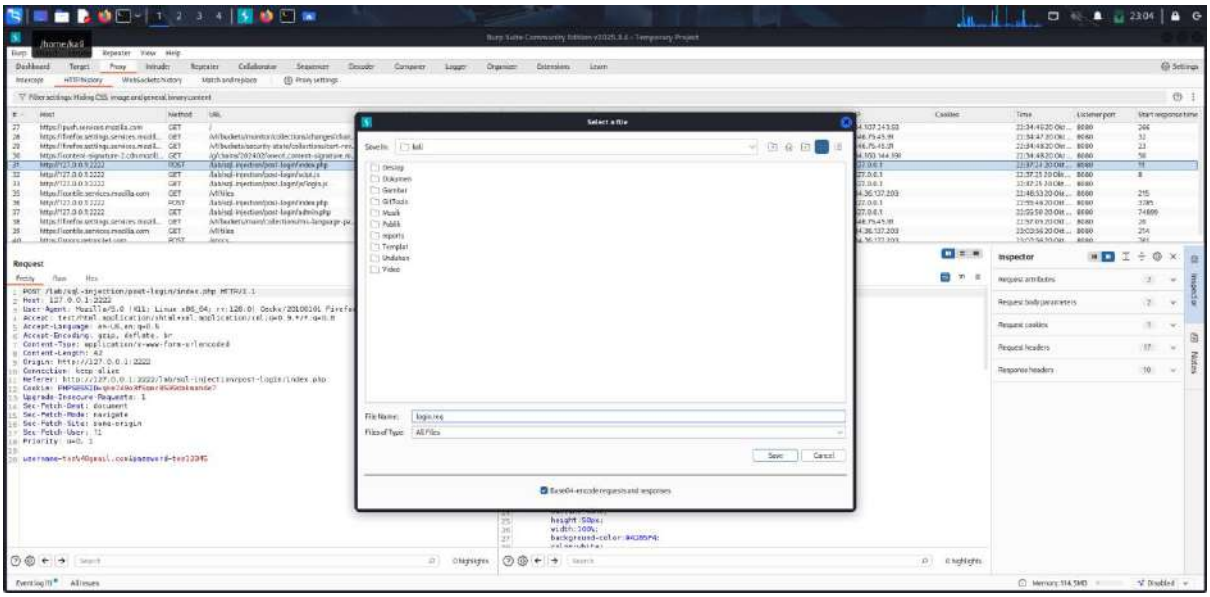


3. Pada field "Password", masukkan string acak (contoh: password).
4. Klik "System Login". Anda akan berhasil masuk ke dasbor admin, membuktikan otentikasi berhasil dilewati.

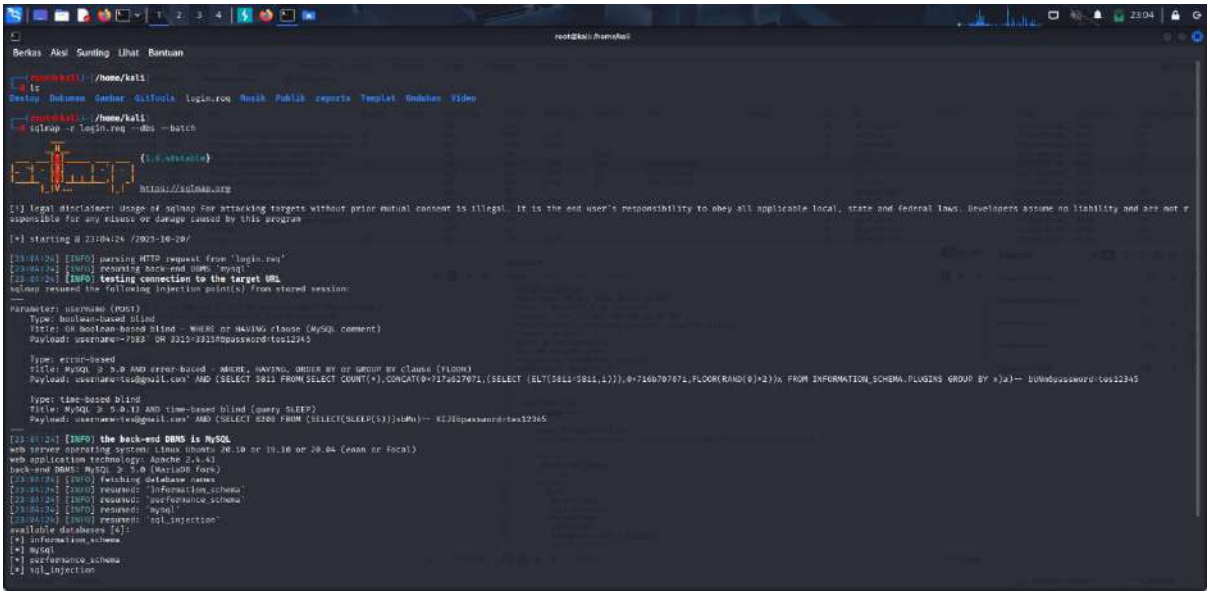


Tahap 2 - Enumerasi Database dengan SQLMap

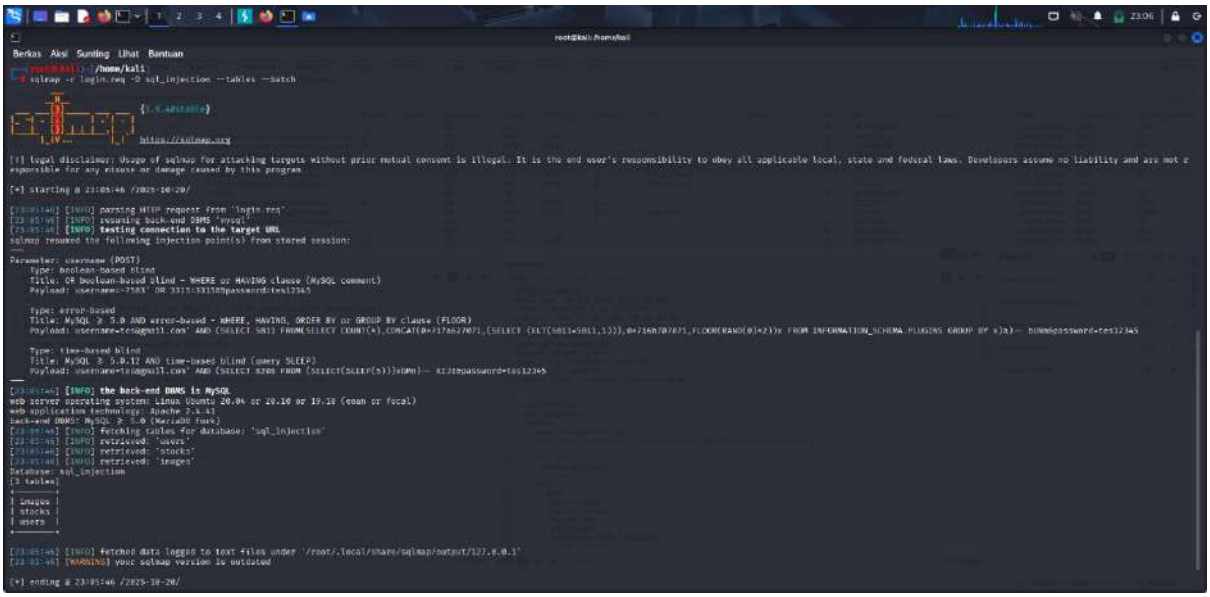
1. Tangkap *request* HTTP POST saat mencoba login menggunakan Burp Suite, lalu simpan ke dalam file bernama login.req.



2. Gunakan SQLMap untuk mendaftar semua database yang ada dengan perintah berikut di terminal: sqlmap -r login.req --dbs –batch

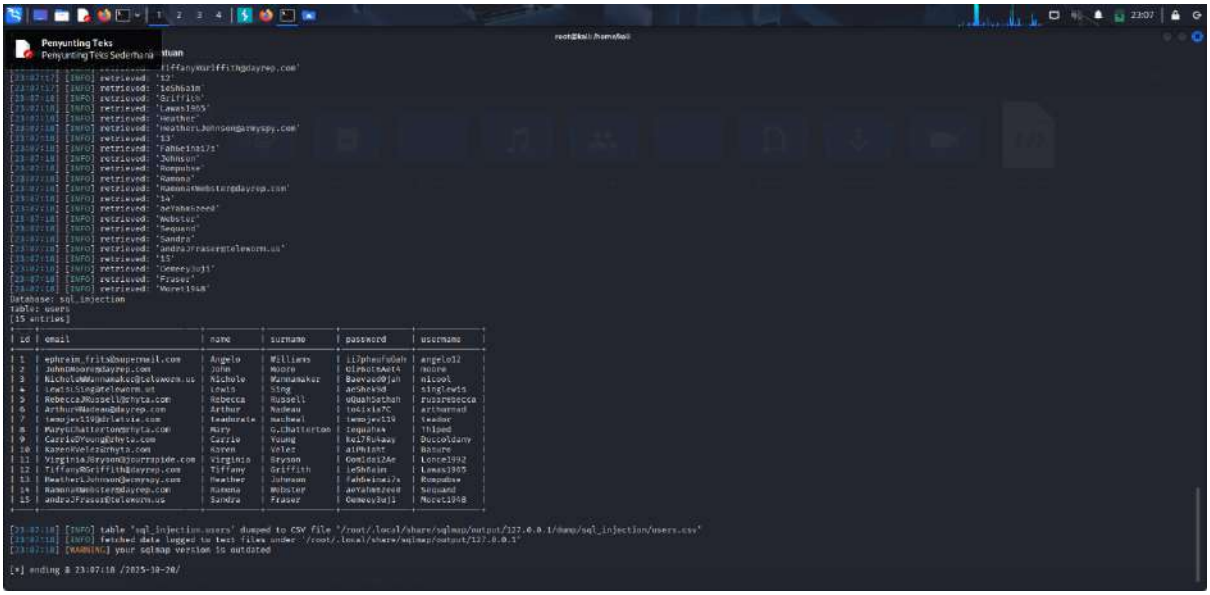


3. Setelah database target ditemukan, lanjutkan dengan mendaftar semua tabel di dalamnya: sqlmap -r login.req -D [NAMA_DATABASE] --tables –batch



4. Terakhir, ambil (dump) data dari tabel yang relevan (misalnya, tabel users):

   sqlmap -r login.req -D [NAMA_DATABASE] -T users --dump –batch



Untuk memperbaiki kerentanan ini secara efektif, direkomendasikan untuk:

1. Gunakan Parameterized Queries (Prepared Statements): Ini adalah metode pertahanan paling fundamental yang memisahkan logika SQL dari data, sehingga input pengguna tidak akan pernah dieksekusi sebagai perintah.
2. Terapkan Validasi Input: Lakukan validasi sisi server yang ketat pada semua input. Untuk email, pastikan formatnya valid.
3. Prinsip Hak Akses Terkecil (Principle of Least Privilege): Pastikan akun database yang digunakan oleh aplikasi web hanya memiliki izin minimum yang diperlukan.

2.2 SQL Injection: Menemukan Password (Manual & SQLMap)

Aplikasi web memiliki kerentanan SQL Injection pada fungsionalitas pencarian (*search*). Parameter search pada URL tidak melakukan sanitasi atau validasi terhadap input pengguna secara memadai. Hal ini memungkinkan penyerang untuk menyisipkan *payload* SQL, khususnya menggunakan teknik UNION, untuk menggabungkan hasil *query* berbahaya dengan hasil *query* yang sah. Eksploitasi berhasil memungkinkan penyerang untuk membaca data sensitif dari tabel lain di dalam database, termasuk *username* dan *password* semua pengguna.

Url Affected/Endpoint	http://127.0.0.1:2222/lab/sql-injection/find-password/?search=...
Severity	**Critical**
CWE	CWE-89: *Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*
OWASP	A03:2021-Injection
CVSS Score	9.8

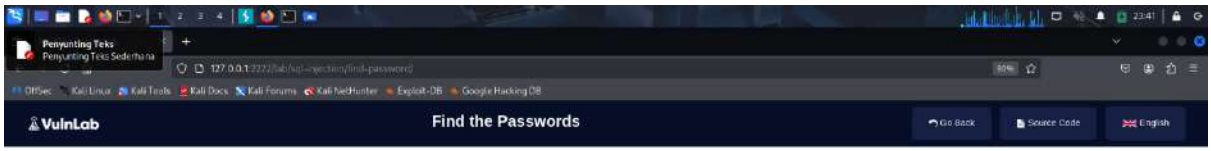
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

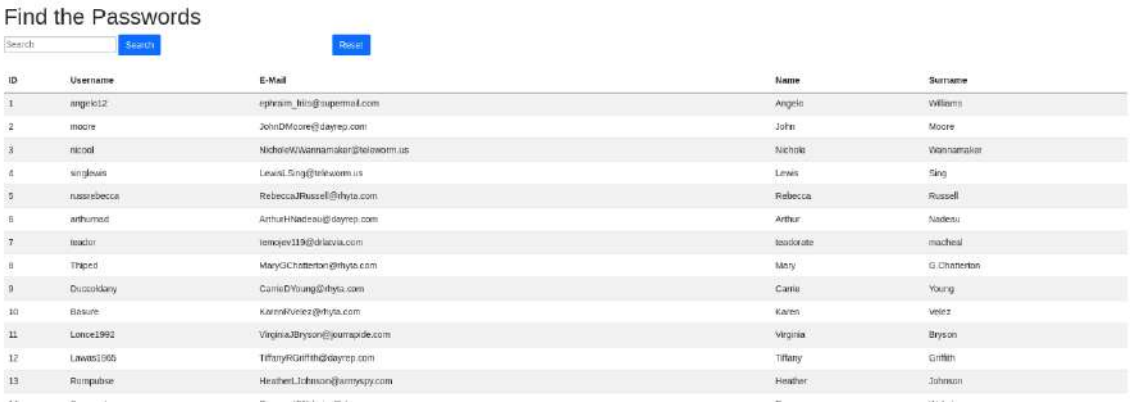
Dampak dari kerentanan ini sangat signifikan dan dapat menyebabkan:

1. Kebocoran Data Kredensial: Penyerang dapat memperoleh seluruh daftar *username* dan *password* pengguna, yang dapat digunakan untuk pengambilalihan akun.
2. Pelanggaran Privasi Pengguna: Informasi pribadi pengguna lainnya seperti email, nama, dan detail lainnya dapat terekspos.
3. Kompromi Database Penuh: Penyerang dapat memperluas serangan untuk memetakan seluruh struktur database dan mengekstrak data dari tabel lainnya.

Proses eksploitasi dimulai dari observasi manual untuk memahami fungsionalitas dan mengonfirmasi kerentanan, kemudian dilanjutkan dengan eksploitasi otomatis menggunakan SQLMap.

1. Observasi Awal: Halaman lab menampilkan fitur pencarian yang menampilkan data pengguna. Fungsionalitas normal memungkinkan pencarian berdasarkan nama, seperti "angelo".

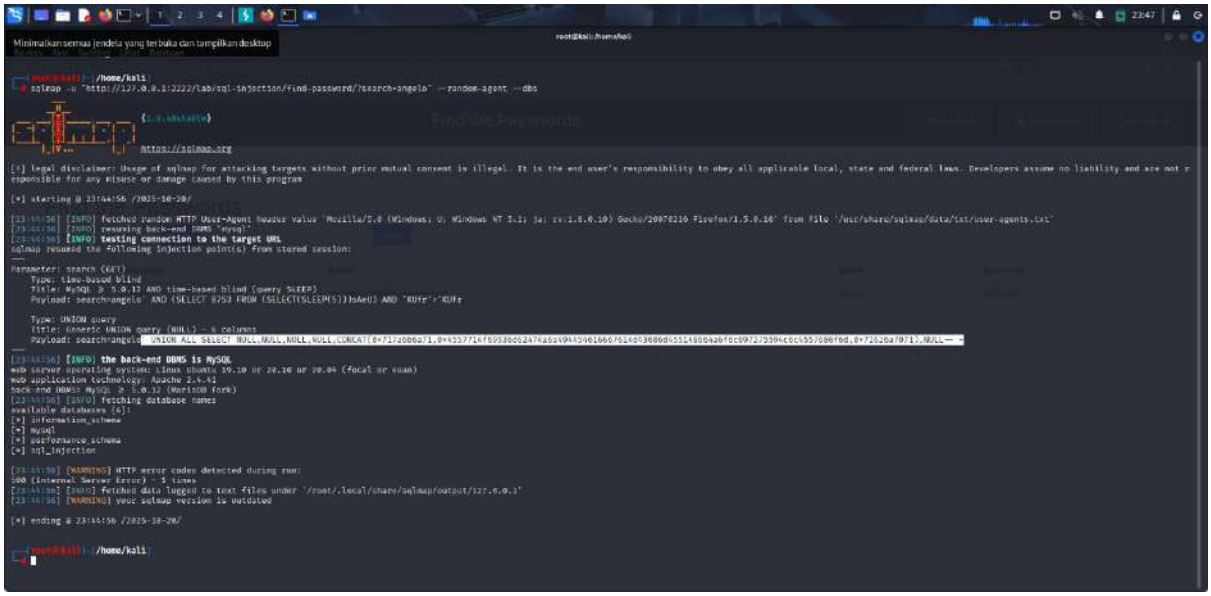


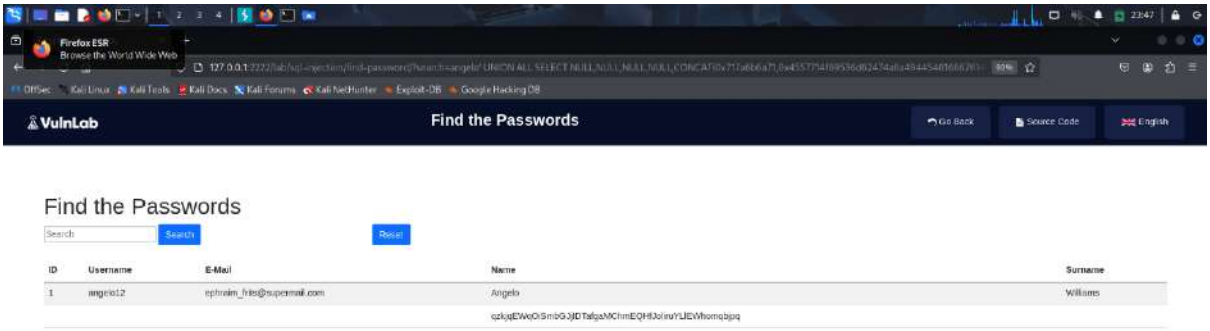


2. Konfirmasi Kerentanan Manual

3. Enumerasi Database (SQLMap): Proses eksploitasi otomatis dimulai dengan SQLMap untuk mendaftar semua database yang ada. Database target, sql_injection, berhasil diidentifikasi.

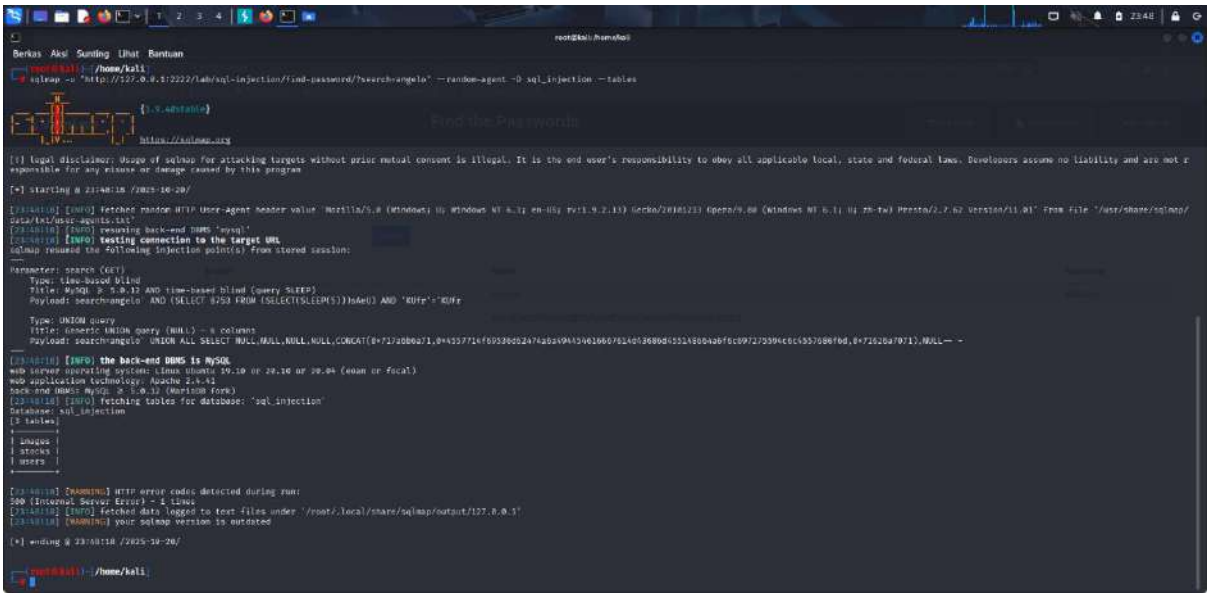
   sqlmap -u http://127.0.0.1:2222/lab/sql-injection/find-password/?search=angelo –random-agent –dbs





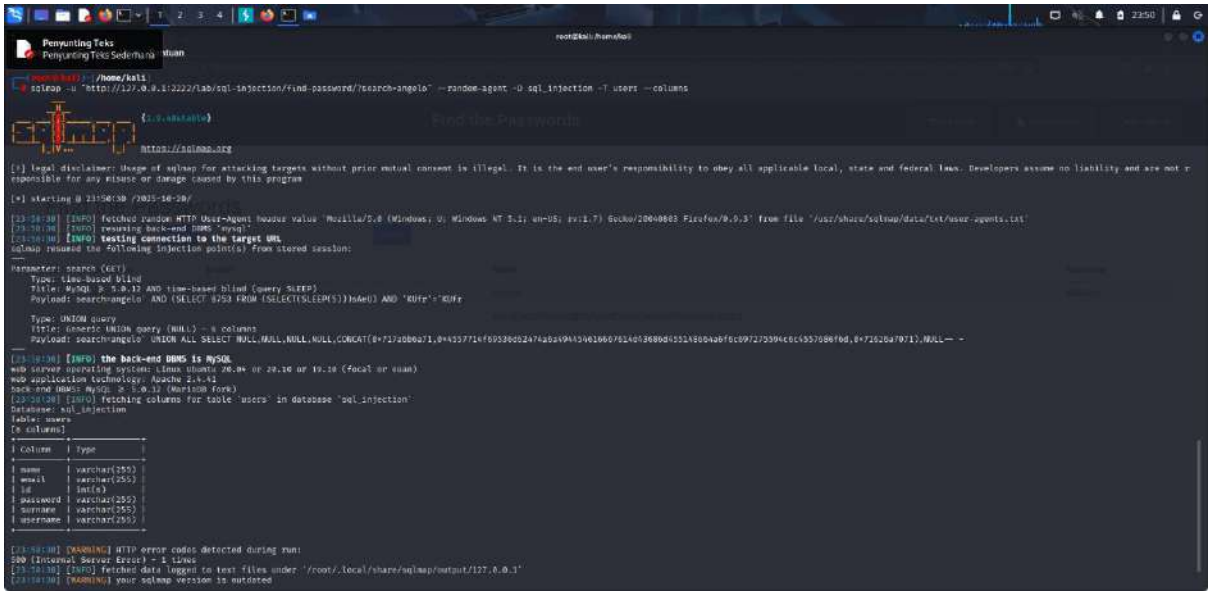
4. Enumerasi Tabel (SQLMap): Setelah database target diketahui, SQLMap digunakan untuk mendaftar semua tabel di dalamnya. Tabel users teridentifikasi sebagai target yang paling relevan.

   sqlmap -u "..." -D sql_injection –tables



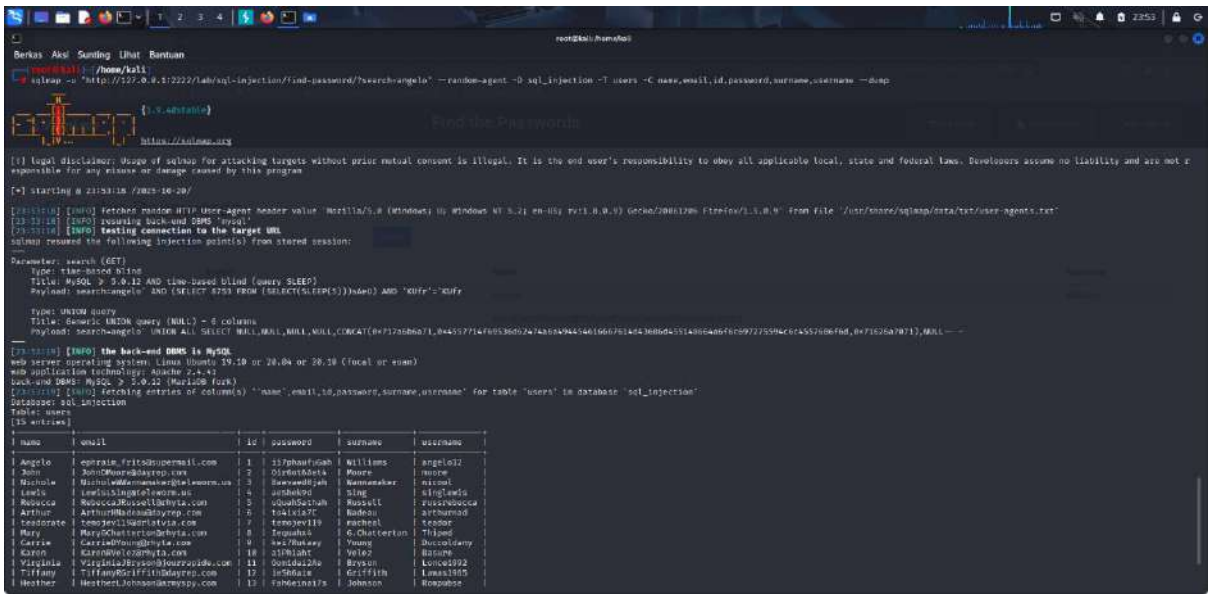
5. Enumerasi Kolom (SQLMap): Langkah selanjutnya adalah melihat struktur dari tabel users untuk menemukan kolom-kolom yang menarik, terutama password.

   sqlmap -u "..." -D sql_injection -T users –columns



6. Dumping Data (SQLMap): Dengan semua informasi yang dibutuhkan, perintah terakhir adalah untuk mengambil (*dump*) seluruh data dari kolom-kolom di dalam tabel users. SQLMap berhasil mengekstrak seluruh data, termasuk *password*.

   sqlmap -u "..." -D sql_injection -T users -C name,email,id,password,surname,username –dump



Rekomendasi untuk perbaikan kerentanan ini sama dengan kerentanan SQL Injection pada umumnya, yaitu:

1. Gunakan Parameterized Queries (Prepared Statements): Ini adalah cara paling efektif untuk memastikan input pengguna diperlakukan sebagai data, bukan sebagai bagian dari perintah SQL.

2. Terapkan Validasi Input Sisi Server: Pastikan semua input yang diterima dari pengguna divalidasi format dan tipenya.

3. Prinsip Hak Akses Terkecil: Batasi hak akses akun database yang digunakan oleh aplikasi agar tidak bisa mengakses informasi yang tidak seharusnya.

2.3 Lab Analisis Kerentanan /vuln/3

Pada bagian ini, akan dianalisis beberapa kerentanan yang ditemukan dalam serangkaian laboratorium di bawah direktori /vuln/3. Setiap sub-bagian akan membahas satu jenis kerentanan secara spesifik.

2.3.1 Invoices - Insecure Direct Object Reference (IDOR)

Aplikasi web ini rentan terhadap serangan *Insecure Direct Object Reference* (IDOR) pada fungsionalitas untuk melihat faktur (*invoice*). Aplikasi menggunakan referensi objek langsung (ID numerik sekuensial, contoh: invoice_id=1) di URL untuk mengambil dan menampilkan data faktur. Namun, tidak ada pemeriksaan kontrol akses yang memadai di sisi *backend* untuk memverifikasi apakah pengguna yang sedang login berhak untuk melihat faktur dengan ID yang diminta. Akibatnya, penyerang dapat dengan mudah memanipulasi parameter invoice_id di URL untuk mengakses dan melihat faktur milik pengguna lain.

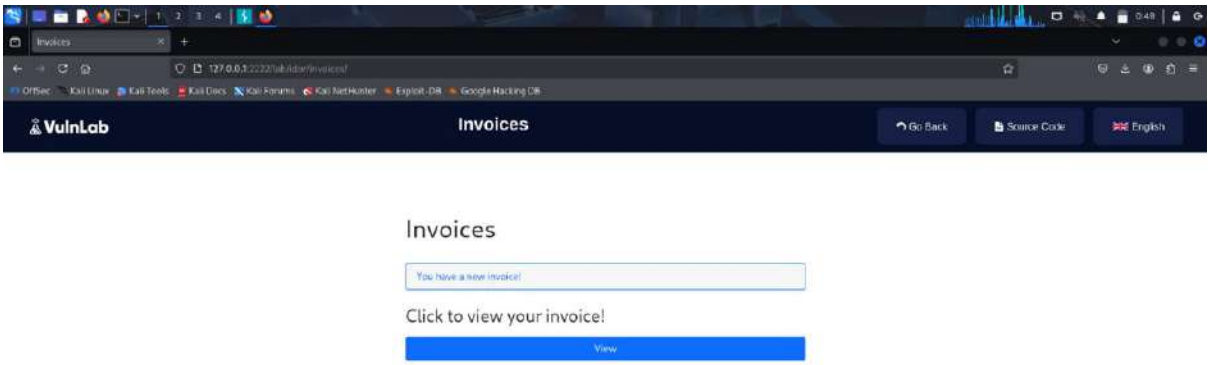
Url Affected/Endpoint	http://127.0.0.1:2222/lab/idor/invoices/index.php?invoice_id=...
Severity	**High**
CWE	CWE-639: *Authorization Bypass Through User-Controlled Key*
OWASP	A01:2021-Broken Access Control
CVSS Score	7.5
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

Eksploitasi kerentanan IDOR ini dapat menyebabkan:

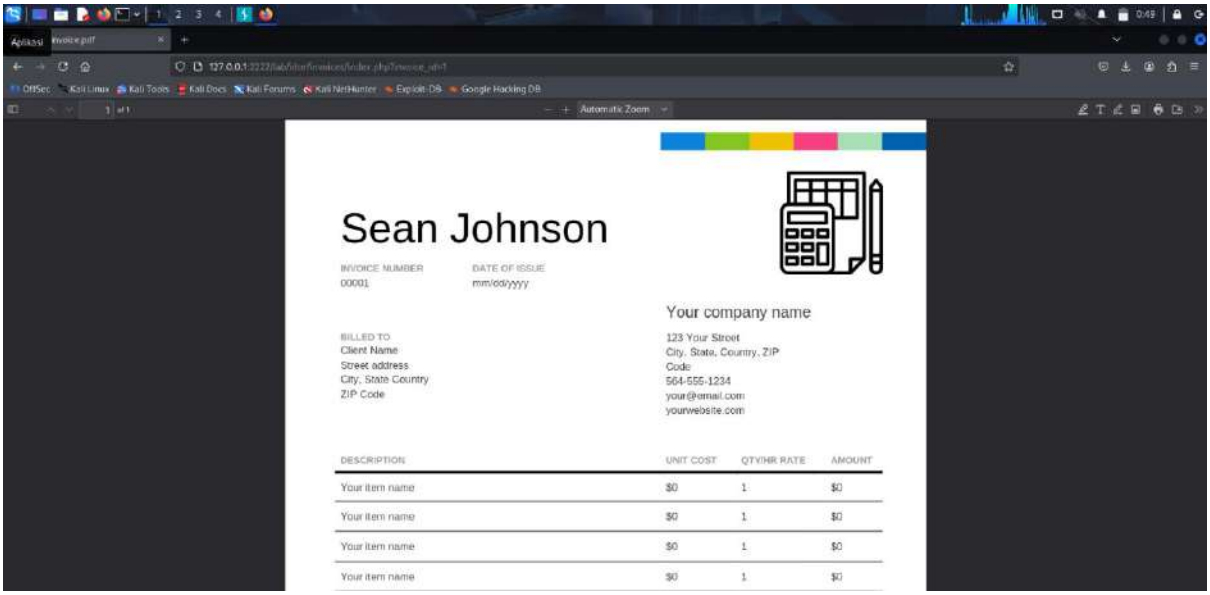
1. Pelanggaran Kerahasiaan Data: Penyerang dapat memperoleh akses tidak sah ke informasi sensitif milik pengguna lain, seperti detail transaksi, nama, dan alamat yang tertera di faktur.
2. Penyalahgunaan Informasi: Data yang bocor dapat disalahgunakan untuk kegiatan penipuan atau kejahatan lainnya.
3. Kehilangan Kepercayaan Pelanggan: Insiden kebocoran data dapat secara signifikan merusak reputasi dan kepercayaan pelanggan terhadap platform.

Step to Reproduce with POC (Proof of Concept)

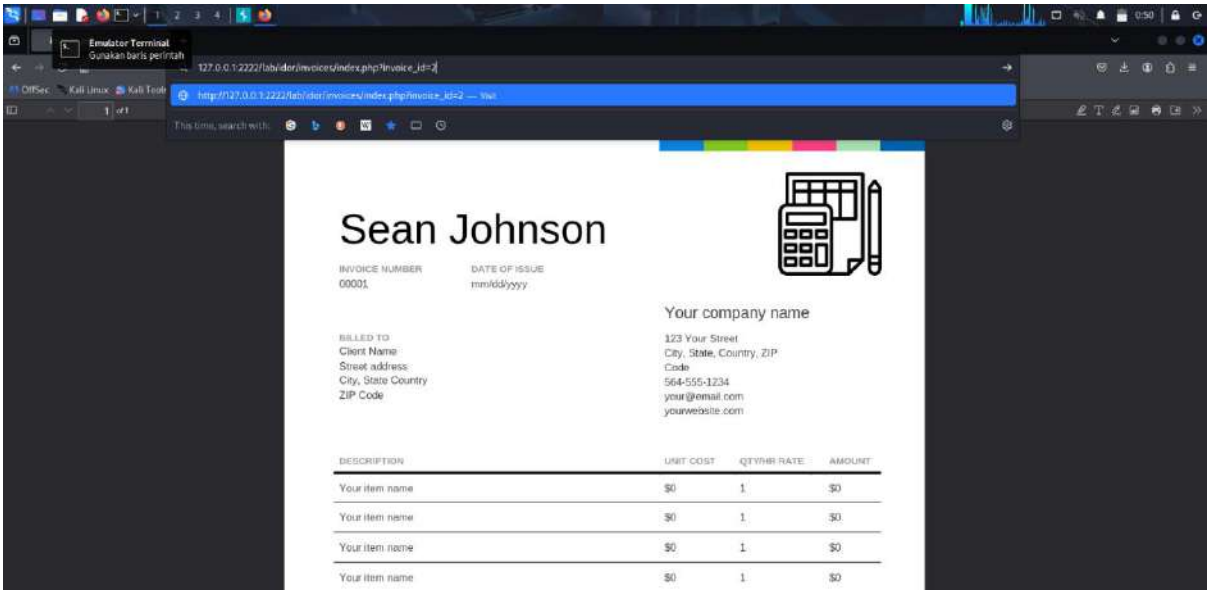
1. Login ke aplikasi dan navigasi ke halaman "Invoices".



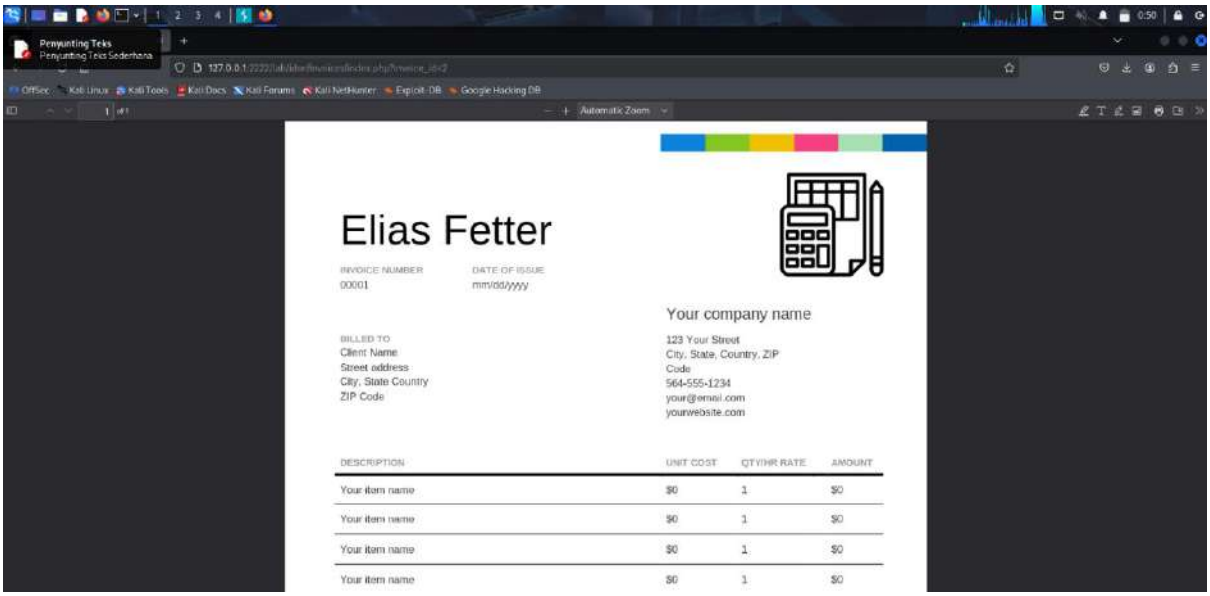
2. Klik tombol "View" untuk melihat faktur yang sah milik kita. Perhatikan URL di *address bar* menunjukkan .../index.php?invoice_id=1 dan menampilkan faktur atas nama "Sean Johnson".



3. Selanjutnya, manipulasi URL secara langsung. Ubah nilai parameter invoice_id dari 1 menjadi 2.



4. Tekan Enter. Aplikasi akan menampilkan faktur dengan invoice_id=2 yang ternyata milik pengguna lain, yaitu "Elias Fetter". Hal ini membuktikan bahwa tidak ada pemeriksaan hak akses yang valid.



Untuk mengatasi kerentanan IDOR ini, langkah-langkah berikut sangat direkomendasikan:

1. Implementasikan Pemeriksaan Kontrol Akses di Sisi Server: Ini adalah perbaikan yang paling krusial. Untuk setiap permintaan untuk mengakses objek (dalam hal ini, faktur), *backend* harus selalu memverifikasi bahwa pengguna yang sedang login memiliki izin untuk mengakses objek tersebut. Jika tidak, kembalikan respons HTTP error seperti 403 Forbidden atau 404 Not Found.

2. Gunakan Referensi Objek Tidak Langsung: Hindari penggunaan ID database yang berurutan dan mudah ditebak di URL. Sebagai gantinya, gunakan pengidentifikasi yang acak dan unik (seperti UUID) untuk setiap pengguna atau sesi. Meskipun ini bukan pengganti untuk pemeriksaan kontrol akses, ini membuat penyerang lebih sulit untuk menebak ID objek milik pengguna lain.

2.3.2 Ticket Sales - Insecure Direct Object Reference (IDOR)

Aplikasi penjualan tiket ini memiliki kerentanan logika bisnis yang kritis. Harga tiket dan jumlah uang yang harus dibayar oleh pengguna dikirimkan sebagai parameter dalam *request* POST (amount dan ticket_money). Aplikasi secara keliru memercayai nilai-nilai yang dikirim dari sisi klien ini untuk memproses transaksi, tanpa melakukan validasi ulang harga di sisi *server*. Akibatnya, seorang penyerang dapat memanipulasi parameter tersebut, misalnya mengubah harga tiket menjadi nilai yang sangat rendah (contoh: 1), dan berhasil melakukan pembelian dengan harga yang tidak sah.

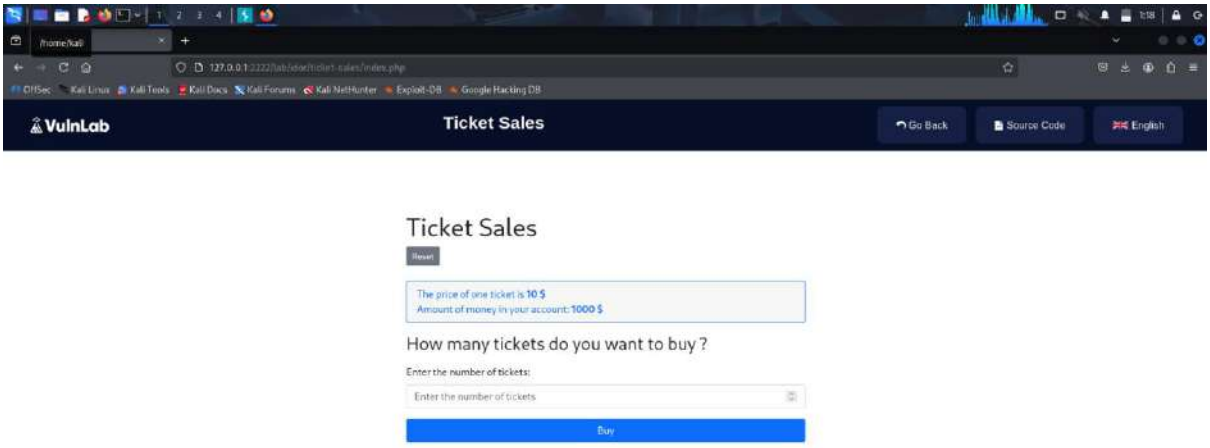
Url Affected/Endpoint	http://127.0.0.1:2222/lab/idor/ticket-sales/index.php
Severity	**High**
CWE	CWE-840: *Business Logic Errors*
OWASP	A05:2021-Security Misconfiguration
CVSS Score	8.1
CVSS String	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H

Eksploitasi kerentanan ini dapat menyebabkan kerugian finansial yang signifikan bagi perusahaan, karena:

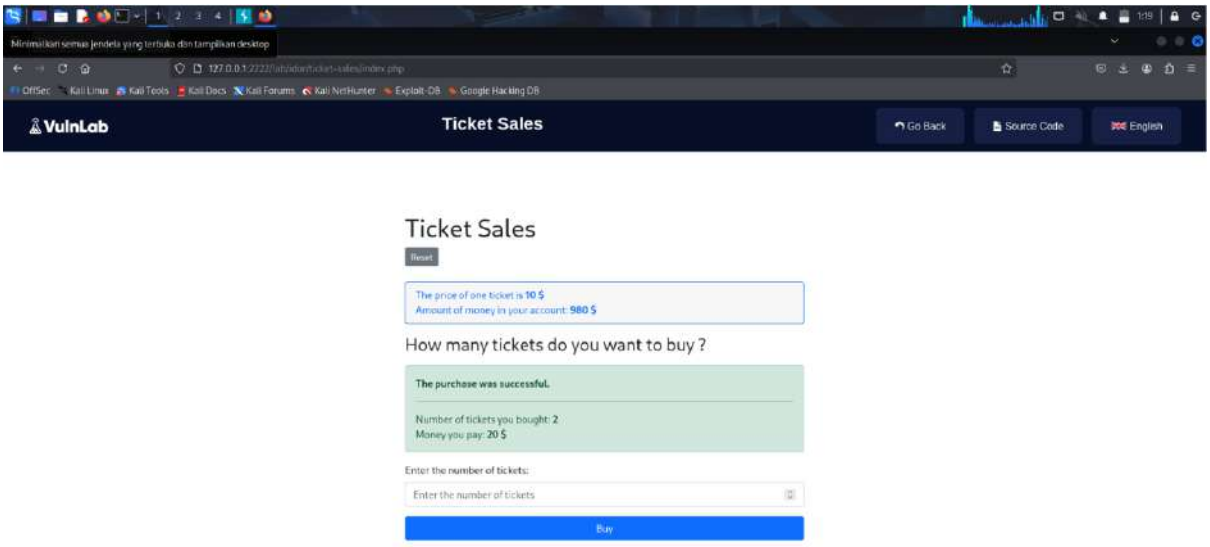
1. Pembelian Produk di Bawah Harga: Pengguna dapat membeli produk (tiket) dengan harga jauh di bawah harga yang seharusnya, bahkan gratis.
2. Kerugian Pendapatan: Jika kerentanan ini dieksploitasi secara massal, perusahaan dapat kehilangan pendapatan dalam jumlah besar.
3. Ketidakadilan Sistem: Merusak integritas sistem penjualan dan menciptakan ketidakadilan bagi pengguna yang jujur.

Step to Reproduce with POC (Proof of Concept)

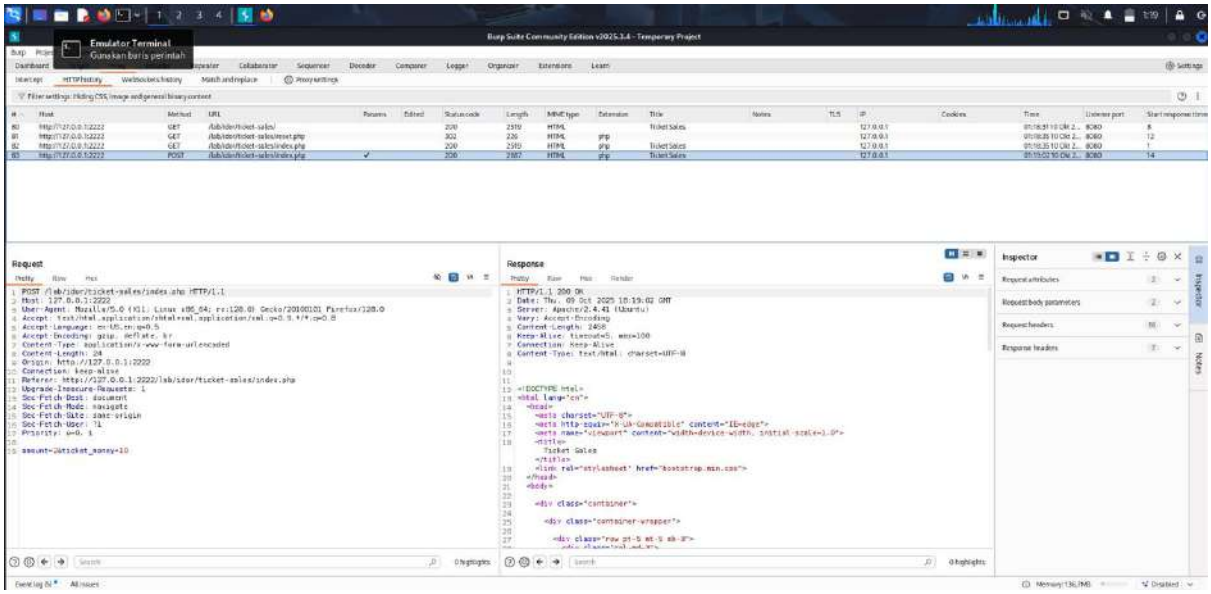
1. Buka halaman "Ticket Sales". Terlihat bahwa harga satu tiket adalah 10$ dan saldo akun kita adalah 1000$.



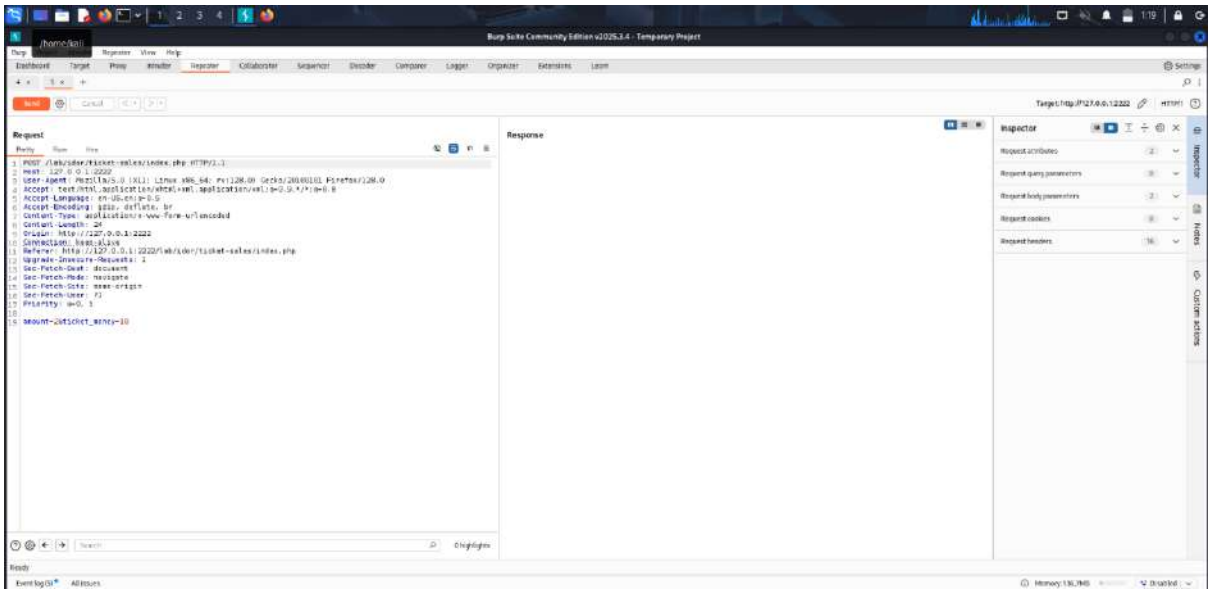
2. Lakukan pembelian normal, misalnya 2 tiket. Aplikasi akan memprosesnya dengan benar, mengurangi saldo sebesar 20$. Saldo tersisa menjadi 980$.



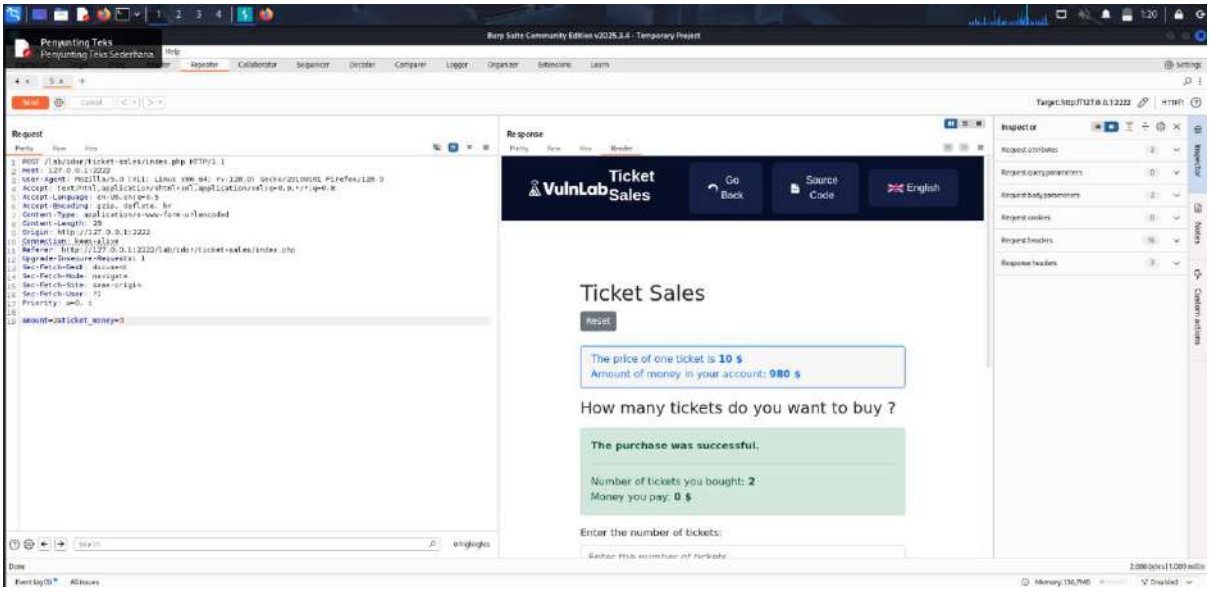
3. Gunakan Burp Suite untuk mencegat (*intercept*) *request* saat melakukan pembelian tiket.



4. Kirim *request* yang dicegat ke "Repeater" di Burp Suite untuk dianalisis dan dimodifikasi. Terlihat ada dua parameter kunci: amount (jumlah tiket) dan ticket_money (harga per tiket).



5. Lakukan manipulasi pada *request*. Ubah nilai parameter ticket_money dari 10 menjadi 0. Biarkan amount tetap 2.



6. Kirim (*send*) *request* yang telah dimodifikasi. Aplikasi memproses transaksi tersebut dan memberikan respons sukses. Dalam respons tersebut, terlihat bahwa pembelian 2 tiket memotong saldo sebesar 0$, bukan 20$. Ini membuktikan bahwa validasi harga hanya terjadi di sisi klien.

Untuk memperbaiki kerentanan logika bisnis ini, perbaikan berikut sangat penting:

1. Lakukan Validasi di Sisi Server (Server-Side Validation): Ini adalah perbaikan utama. Jangan pernah memercayai data yang dikirim dari klien, terutama data krusial seperti harga dan kuantitas. *Backend* harus selalu mengambil harga produk dari sumber yang terpercaya (database) pada saat transaksi diproses, mengabaikan nilai harga yang dikirim dalam *request*.
2. Integritas Data Transaksi: Pastikan semua kalkulasi terkait transaksi (harga x kuantitas = total) dilakukan secara eksklusif di sisi *server*.
3. Sesi dan Manajemen Status: Kelola status sesi pengguna dengan aman untuk memastikan bahwa data transaksi tidak dapat diubah di tengah jalan.

2.3.3 Changing Password - Insecure Direct Object Reference (IDOR)

Fungsionalitas penggantian password pada aplikasi ini memiliki kerentanan IDOR yang fatal. Saat pengguna mengganti password, *request* POST yang dikirimkan ke server menyertakan parameter tersembunyi (user_id) untuk mengidentifikasi akun mana yang akan diubah. Aplikasi di sisi *backend* secara keliru memercayai nilai user_id yang dikirim oleh klien ini tanpa melakukan validasi apakah pengguna yang sedang login (berdasarkan sesi) memiliki otorisasi untuk melakukan perubahan tersebut. Akibatnya, seorang penyerang yang sudah login sebagai pengguna biasa dapat memanipulasi nilai user_id untuk mengubah password pengguna lain, termasuk admin, dan mengambil alih akun mereka sepenuhnya.

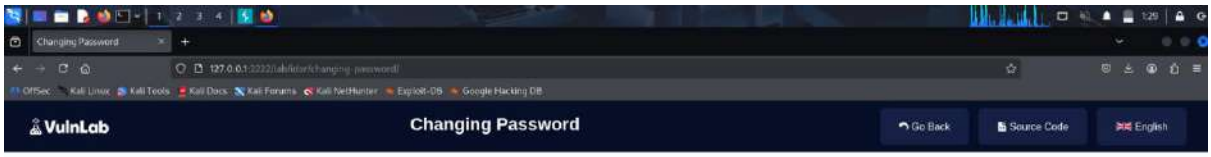
Url Affected/Endpoint	http://127.0.0.1:2222/lab/idor/changing-password/
Severity	**Critical**
CWE	CWE-639: *Authorization Bypass Through User-Controlled Key*
OWASP	A01:2021-Broken Access Control
CVSS Score	8.8
CVSS String	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

Dampak dari kerentanan ini sangat berbahaya, meliputi:

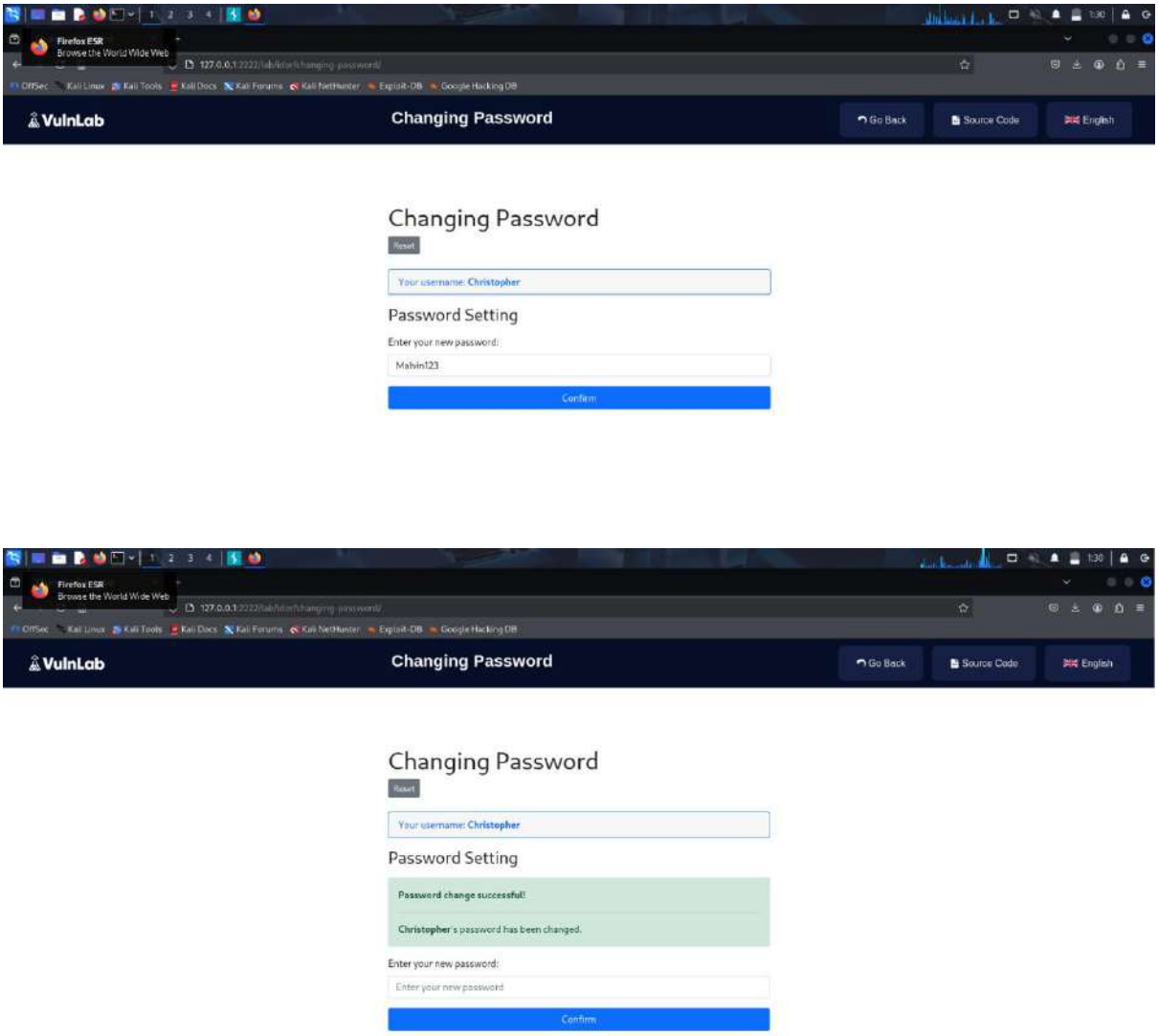
1. Pengambilalihan Akun Penuh: Penyerang dapat mengambil alih akun pengguna manapun, hanya dengan mengetahui user_id mereka yang bersifat sekuensial dan mudah ditebak.
2. Eskalasi Hak Istimewa: Jika penyerang berhasil mengambil alih akun administrator, mereka akan mendapatkan kontrol penuh atas seluruh aplikasi.
3. Pencurian dan Manipulasi Data: Dari akun yang diambil alih, penyerang dapat mencuri data sensitif, melakukan transaksi ilegal, atau merusak data.
4. Penguncian Akun Pengguna Sah: Pengguna yang sah akan terkunci dari akunnya sendiri karena passwordnya telah diubah oleh penyerang.

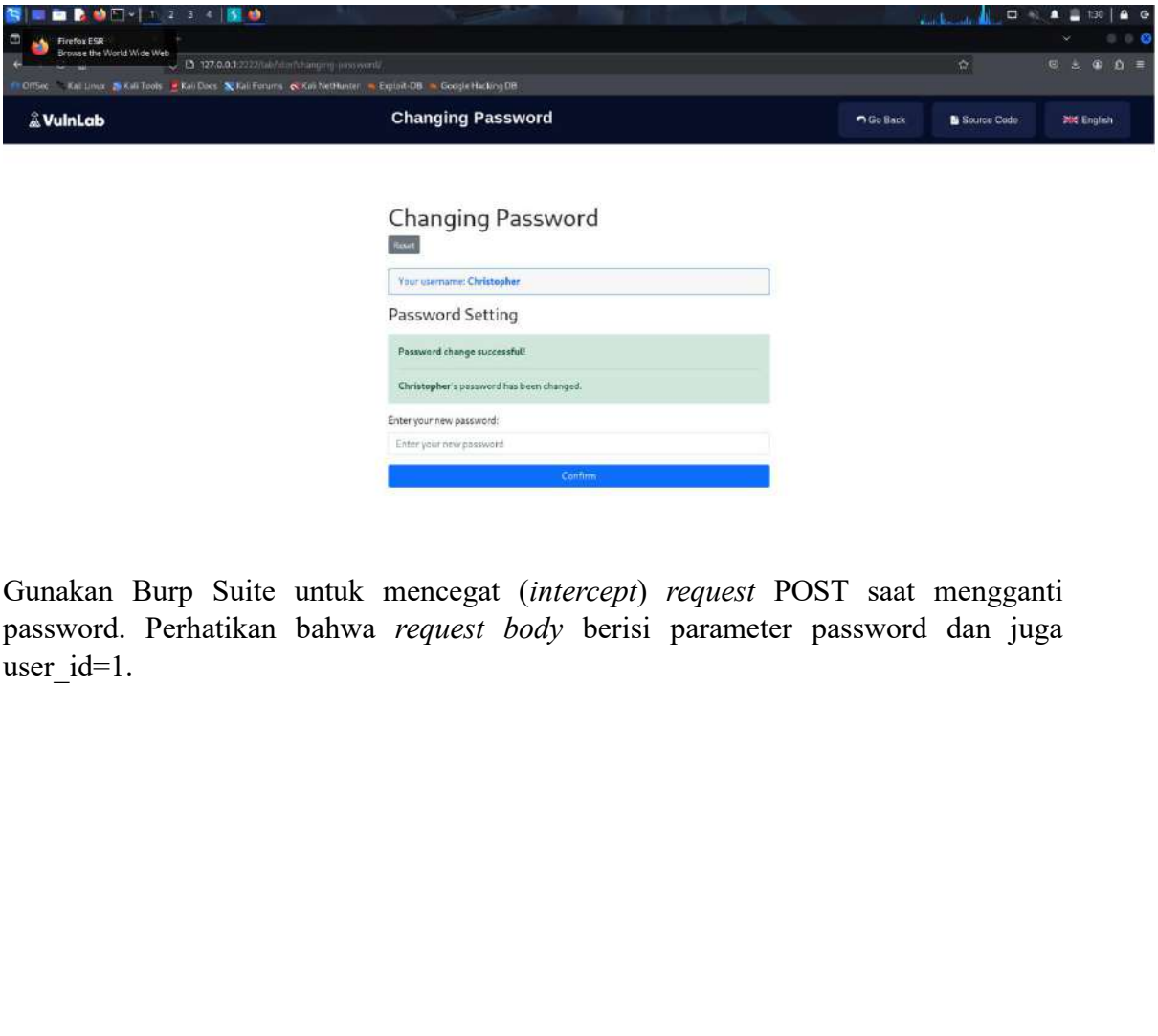
Step to Reproduce with POC (Proof of Concept)

1. Login sebagai pengguna "Christopher" dan buka halaman "Changing Password".

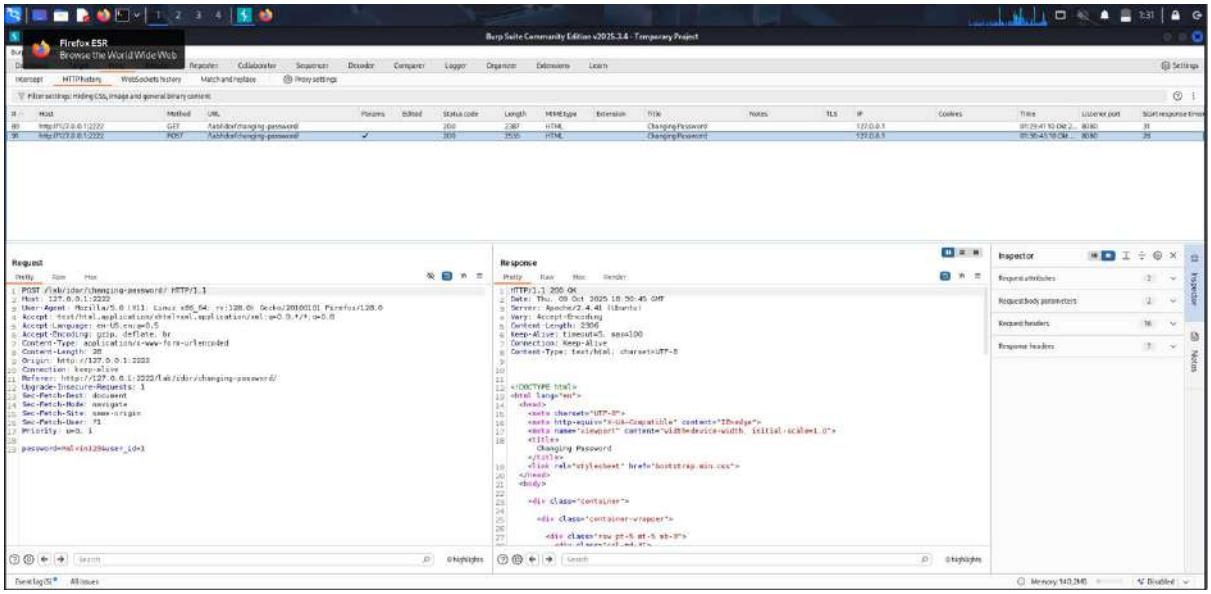


2. Lakukan proses penggantian password untuk akun kita sendiri secara normal untuk memahami alur kerja aplikasi.

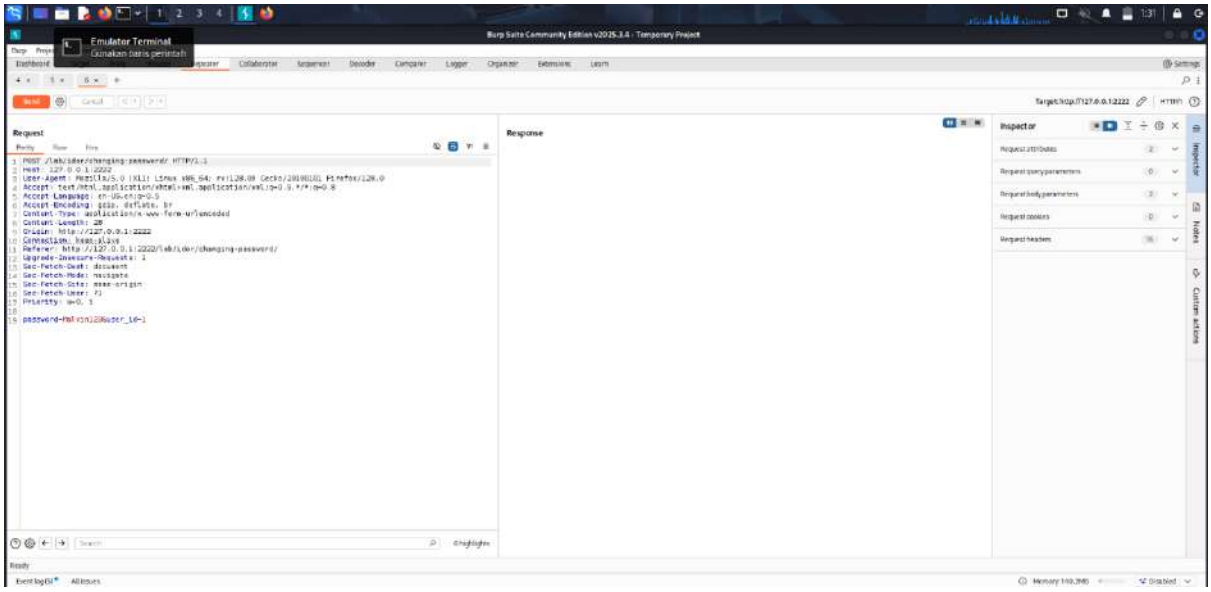




3. Gunakan Burp Suite untuk mencegat (*intercept*) *request* POST saat mengganti password. Perhatikan bahwa *request body* berisi parameter password dan juga user_id=1.

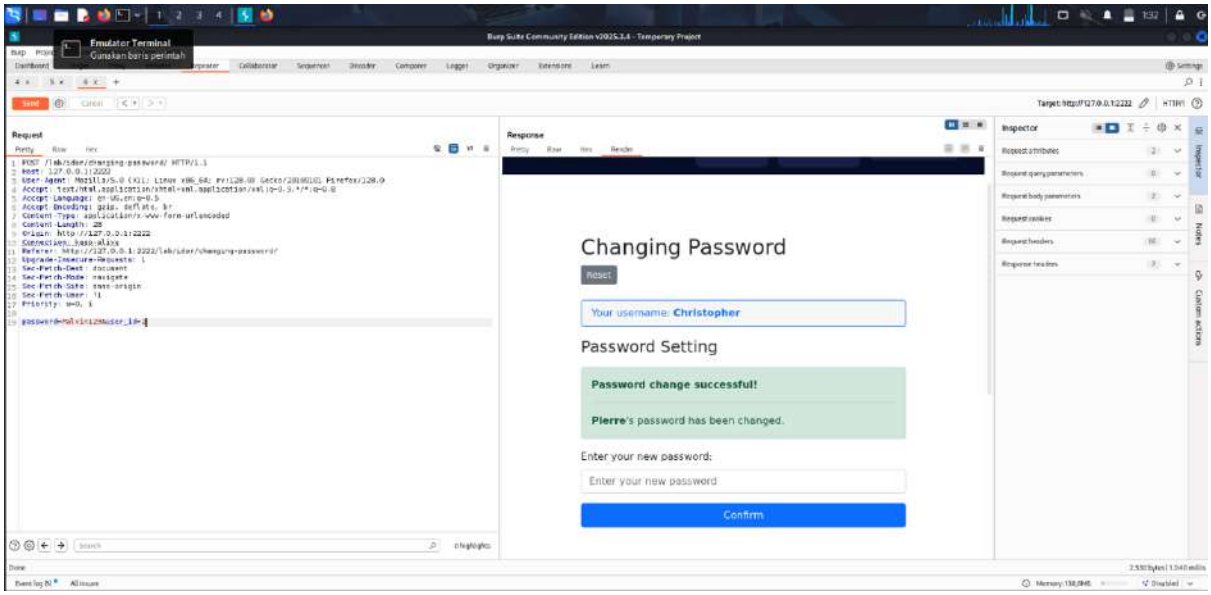


4. Kirim *request* ini ke Burp Suite Repeater untuk dimodifikasi.



5. Pada Repeater, ubah nilai parameter user_id dari 1 (milik Christopher) menjadi 2 (milik pengguna lain). Biarkan nilai password tetap sama.

6. Kirim *request* yang telah dimanipulasi. Aplikasi memberikan respons sukses yang menyatakan bahwa password milik "Pierre" (pengguna dengan ID 2) telah berhasil diubah. Penyerang kini dapat login sebagai "Pierre" menggunakan password yang baru saja ia atur.



Untuk mengatasi kerentanan kritis ini, perbaikan berikut harus segera diimplementasikan:

1. Gunakan Identifikasi Berbasis Sesi: Ini adalah perbaikan yang paling fundamental. Di sisi *server*, jangan pernah memercayai parameter user_id yang dikirim dari klien untuk operasi sensitif. Sebaliknya, identifikasi pengguna *hanya* dari informasi sesi (misalnya, *session cookie*) yang sudah terotentikasi. Logika *backend* harus mengambil ID pengguna dari sesi, lalu mengubah password untuk ID tersebut.
2. Wajibkan Konfirmasi Password Saat Ini: Sebagai lapisan keamanan tambahan (*defense-in-depth*), wajibkan pengguna untuk memasukkan password mereka yang lama (saat ini) sebelum diizinkan untuk mengatur password baru.

2.3.4 Money Transfer - Insecure Direct Object Reference (IDOR)

Fungsi transfer uang pada aplikasi ini memiliki kerentanan IDOR yang sangat berbahaya. Proses transfer dana mengandalkan parameter sender_id yang dikirim dari sisi klien untuk menentukan sumber dana. Aplikasi di sisi *backend* tidak melakukan validasi untuk memastikan bahwa sender_id yang dikirim dalam *request* sama dengan ID pengguna yang sedang login (berdasarkan sesi). Celah ini memungkinkan seorang penyerang untuk memanipulasi parameter sender_id dan mengirim uang dari akun pengguna lain ke akunnya sendiri atau akun lain, yang secara efektif merupakan tindakan pencurian dana.

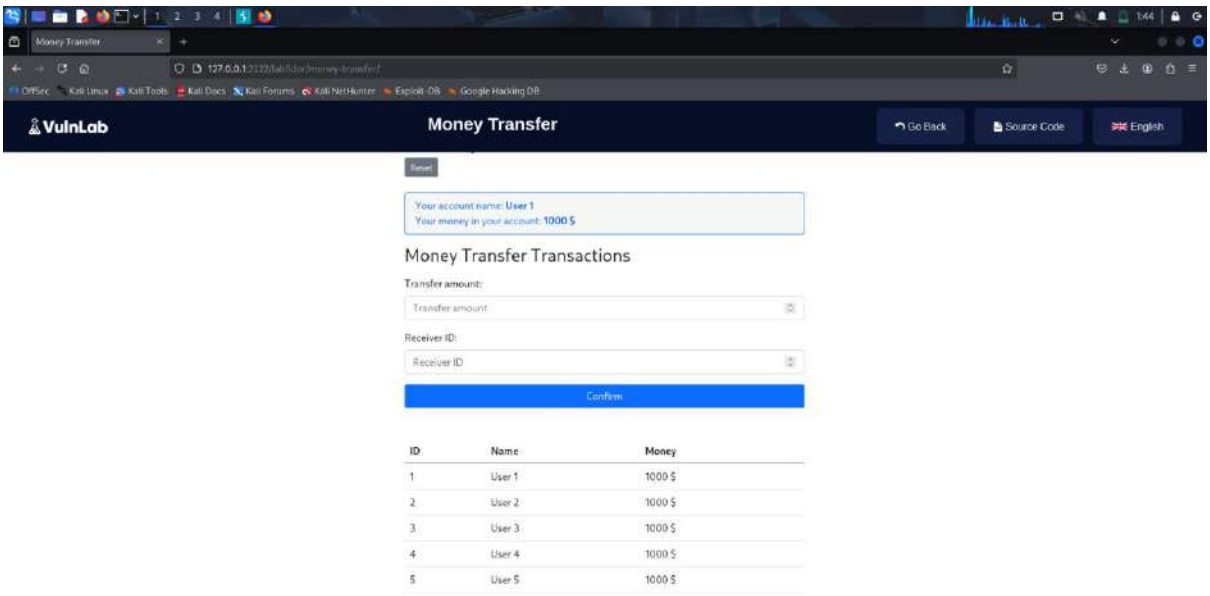
Url Affected/Endpoint	http://127.0.0.1:2222/lab/idor/money-transfer/
Severity	**Critical**
CWE	CWE-639: *Authorization Bypass Through User-Controlled Key*
OWASP	A01:2021-Broken Access Control
CVSS Score	8.1
CVSS String	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H

Dampak dari eksploitasi kerentanan ini sangat parah dan langsung, meliputi:

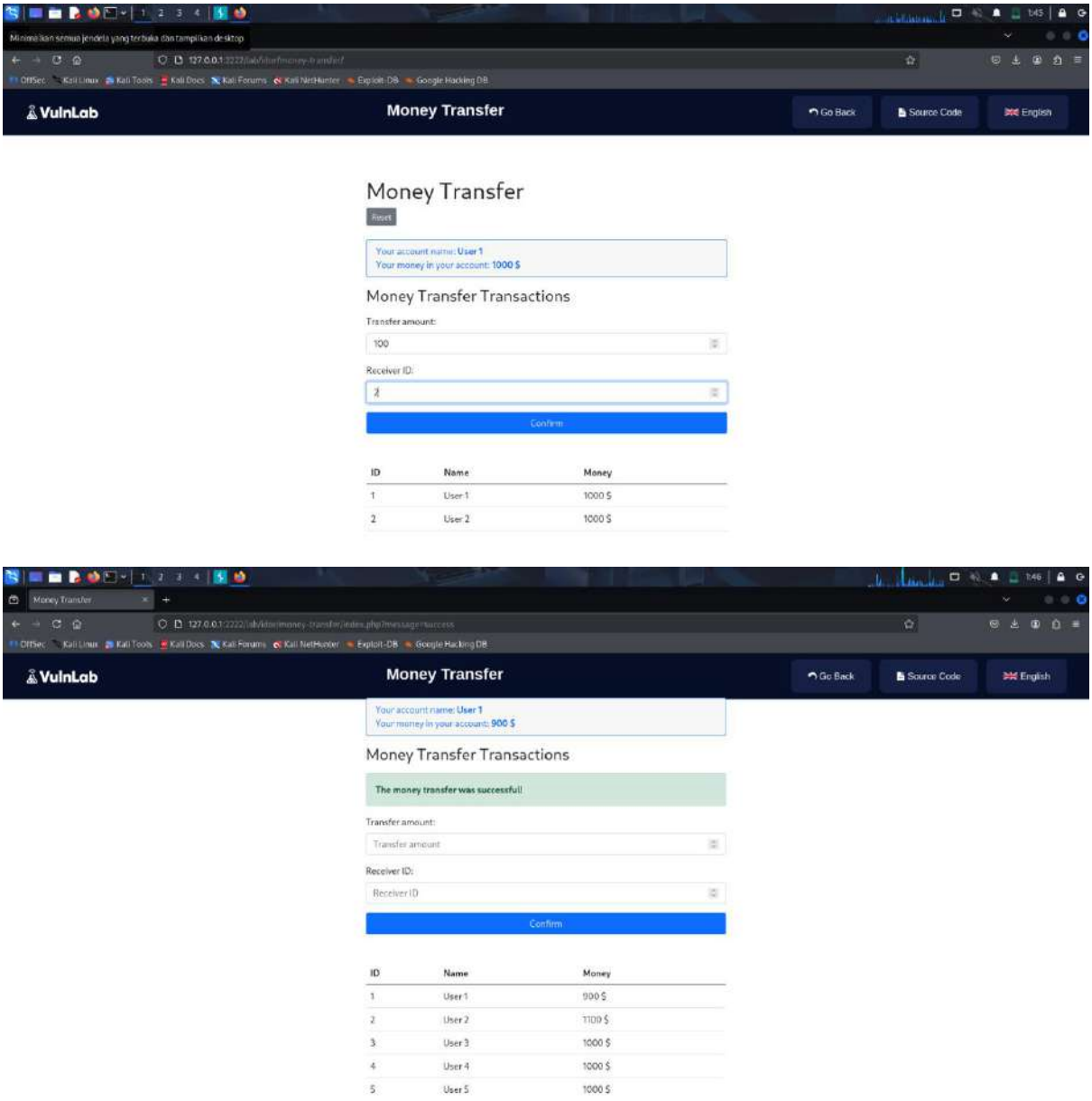
1. Kerugian Finansial Langsung: Penyerang dapat mencuri dana dari akun pengguna manapun di dalam sistem, menyebabkan kerugian finansial yang nyata.
2. Integritas Transaksi Rusak: Seluruh sistem transaksi keuangan menjadi tidak dapat dipercaya karena penyerang dapat membuat transaksi palsu atas nama orang lain.
3. Kerusakan Reputasi Parah: Insiden pencurian dana akan menghancurkan kepercayaan pengguna dan reputasi platform secara keseluruhan, yang dapat berujung pada konsekuensi hukum.

Step to Reproduce with POC (Proof of Concept)

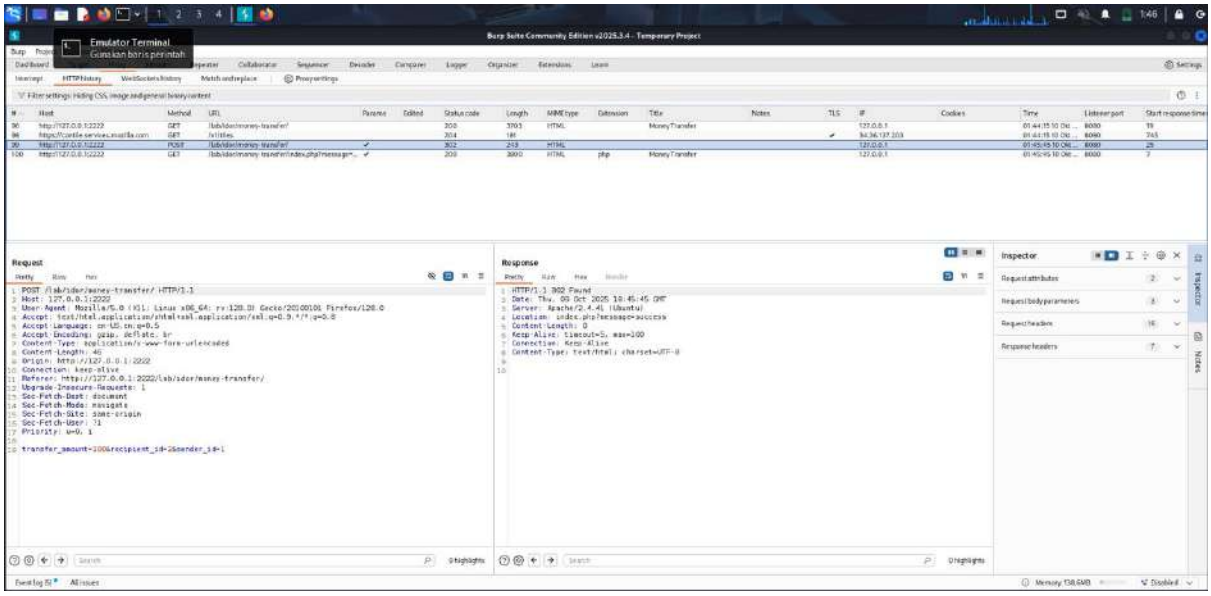
1. Login sebagai "User 1" dengan saldo awal 1000$. Halaman menampilkan form transfer dan saldo pengguna.



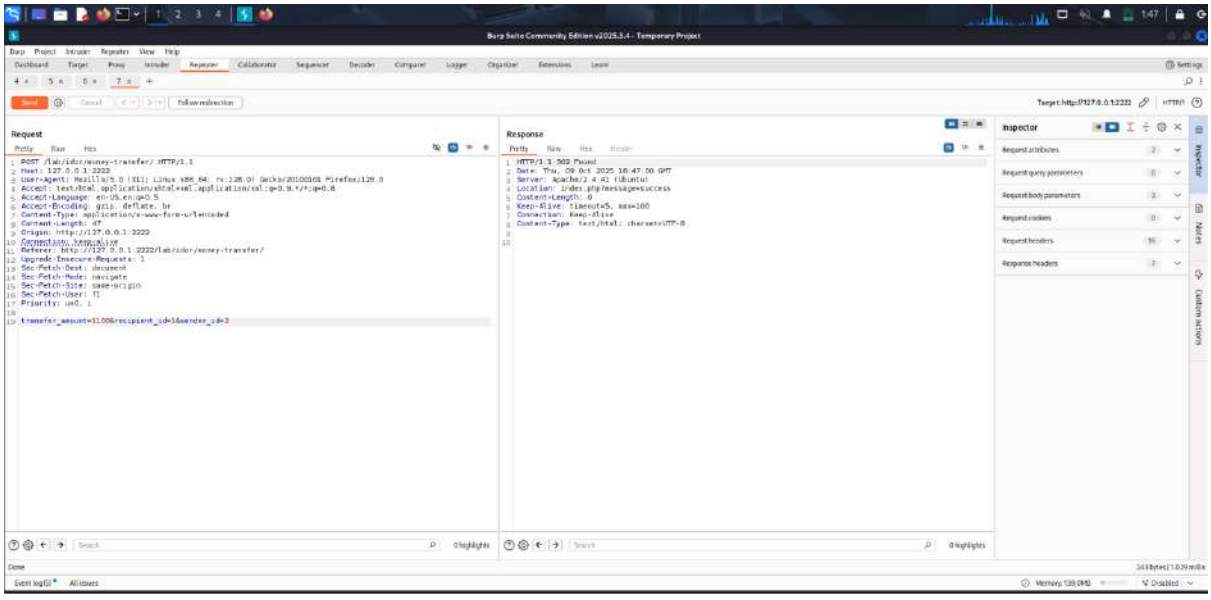
2. Lakukan transaksi yang sah, misalnya mengirim 100$ ke "User 2". Amati bahwa saldo "User 1" berkurang menjadi 900$ dan saldo "User 2" bertambah menjadi 1100$.



3. Gunakan Burp Suite untuk mencegat (*intercept*) *request* POST saat melakukan transaksi. Terlihat bahwa *request body* mengandung tiga parameter: transfer_amount, recipient_id, dan sender_id=1.

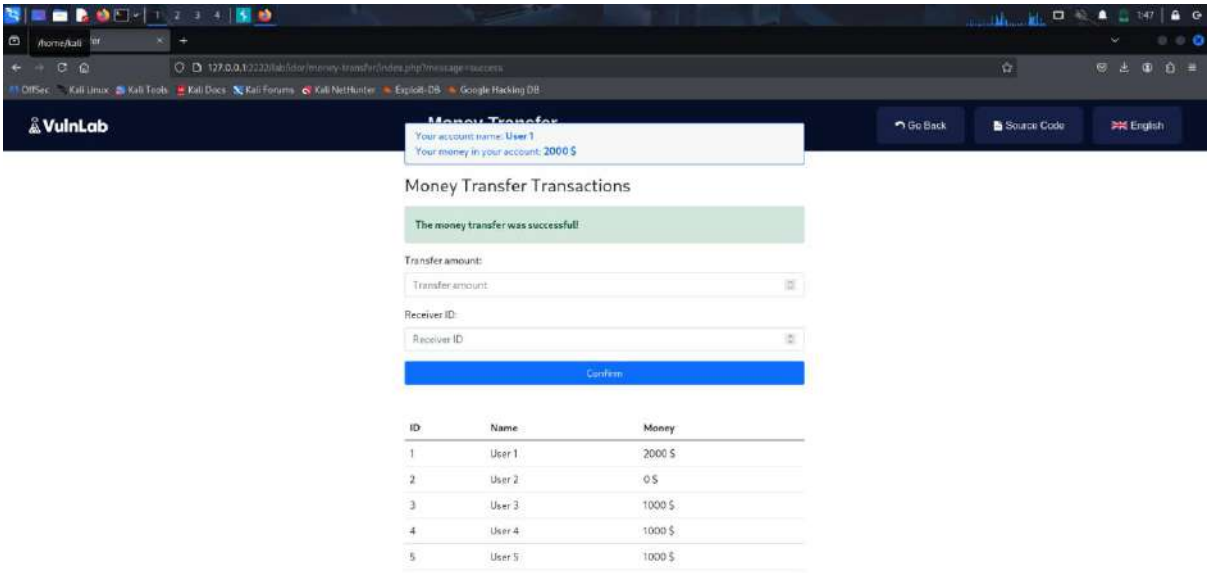


4. Kirim *request* tersebut ke Burp Suite Repeater. Modifikasi parameter untuk melakukan serangan. Ubah sender_id menjadi 2 (korban) dan recipient_id menjadi 1 (penyerang). Jumlah transfer bisa diatur sesuai keinginan, misalnya 1100$.



5. Kirim *request* yang telah dimanipulasi. Aplikasi memberikan respons sukses.

6. Periksa kembali halaman transfer. Terlihat bahwa transaksi "berhasil" dan saldo "User 1" kini menjadi 2000$, sementara saldo "User 2" menjadi 0$. Ini membuktikan bahwa dana berhasil dicuri dari akun "User 2".



Untuk memperbaiki kerentanan kritis ini, perbaikan berikut bersifat wajib:

1. Gunakan ID Pengguna dari Sesi Server: Di sisi *server*, jangan pernah memercayai parameter sender_id yang dikirim dari klien. Sumber dana untuk transaksi harus diambil secara eksklusif dari informasi sesi pengguna yang sedang terotentikasi. Abaikan parameter sender_id dari *request* dan gunakan ID pengguna dari sesi sebagai satu-satunya sumber kebenaran.
2. Validasi Transaksi Berlapis: Terapkan validasi di *backend* untuk memastikan akun pengirim memiliki saldo yang cukup sebelum transaksi diizinkan.

2.3.5 Address Entry - Insecure Direct Object Reference (IDOR)

Fungsionalitas pemesanan (*order*) pada aplikasi ini memiliki kerentanan IDOR. Ketika seorang pengguna melakukan pemesanan, *request* POST yang dikirim ke server dapat dimanipulasi untuk menyertakan parameter address_id. Aplikasi di sisi *backend* secara keliru memproses parameter ini dan mengambil alamat yang sesuai dengan ID tersebut dari database untuk dijadikan alamat pengiriman. Celah ini terjadi karena tidak ada validasi untuk memeriksa apakah address_id yang disertakan benar-benar milik pengguna yang sedang melakukan pemesanan (berdasarkan sesi). Akibatnya, penyerang dapat memesan barang namun mengirimkannya ke alamat terdaftar milik pengguna lain.

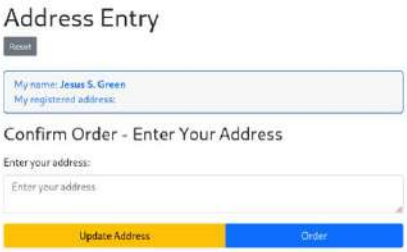
Url Affected/Endpoint	http://127.0.0.1:2222/lab/idor/address-entry/
Severity	**High**
CWE	CWE-639: *Authorization Bypass Through User-Controlled Key*
OWASP	A01:2021-Broken Access Control
CVSS Score	7.1
CVSS String	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:H/A:N

Meskipun tidak secara langsung mencuri data, dampak dari kerentanan ini tetap signifikan:

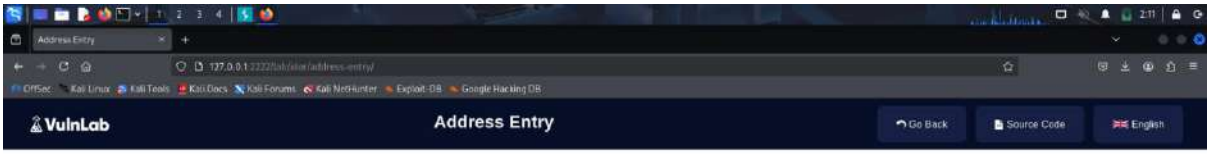
1. Pelanggaran Privasi dan Pelecehan: Penyerang dapat mengirim barang yang tidak diinginkan ke alamat korban, yang merupakan bentuk pelecehan dan pelanggaran privasi (doxing).
2. Potensi Penipuan: Kerentanan ini dapat menjadi bagian dari skema penipuan yang lebih kompleks, di mana penyerang mencoba mencegat paket yang dikirim ke alamat yang diketahui.
3. Integritas Pemesanan Rusak: Sistem gagal memastikan bahwa pesanan dari seorang pengguna dikirim ke alamat milik pengguna tersebut, merusak logika bisnis inti dari aplikasi.

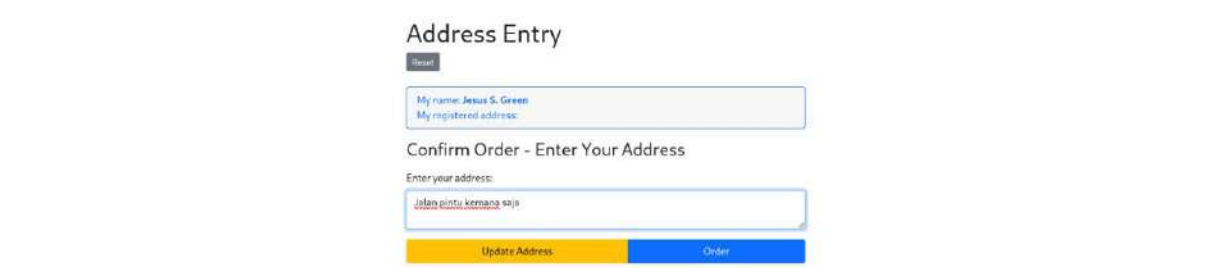
Step to Reproduce with POC (Proof of Concept)

1. Login sebagai pengguna "Jesus S. Green" dan buka halaman "Address Entry".

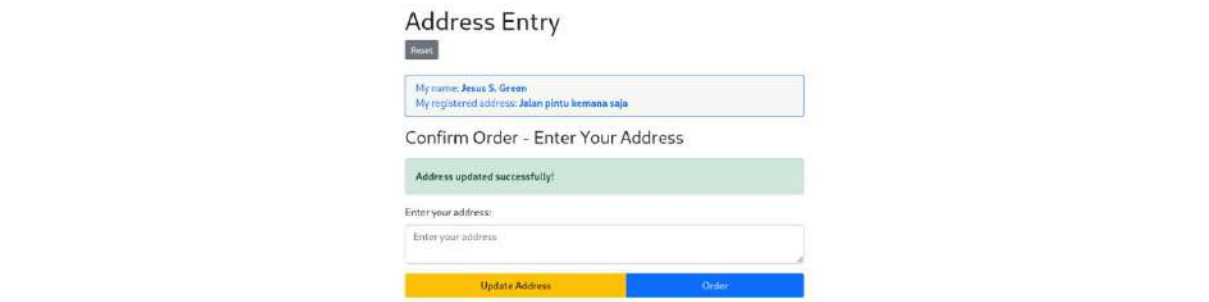


2. Lakukan pembaruan alamat dan pemesanan yang sah untuk memahami alur kerja normal dan menangkap *request* yang relevan.

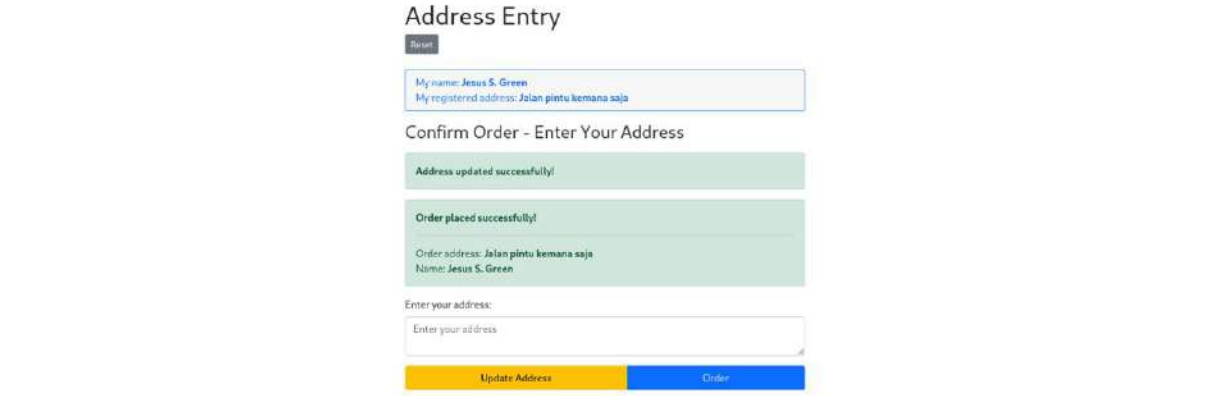

VulnLab
Address Entry
Go Back
Source Code
English


Address Entry
Reset
My name: Jesus S. Green
My registered address:
Confirm Order - Enter Your Address
Enter your address:
Jalan pintu kemana saja
Update Address
Order

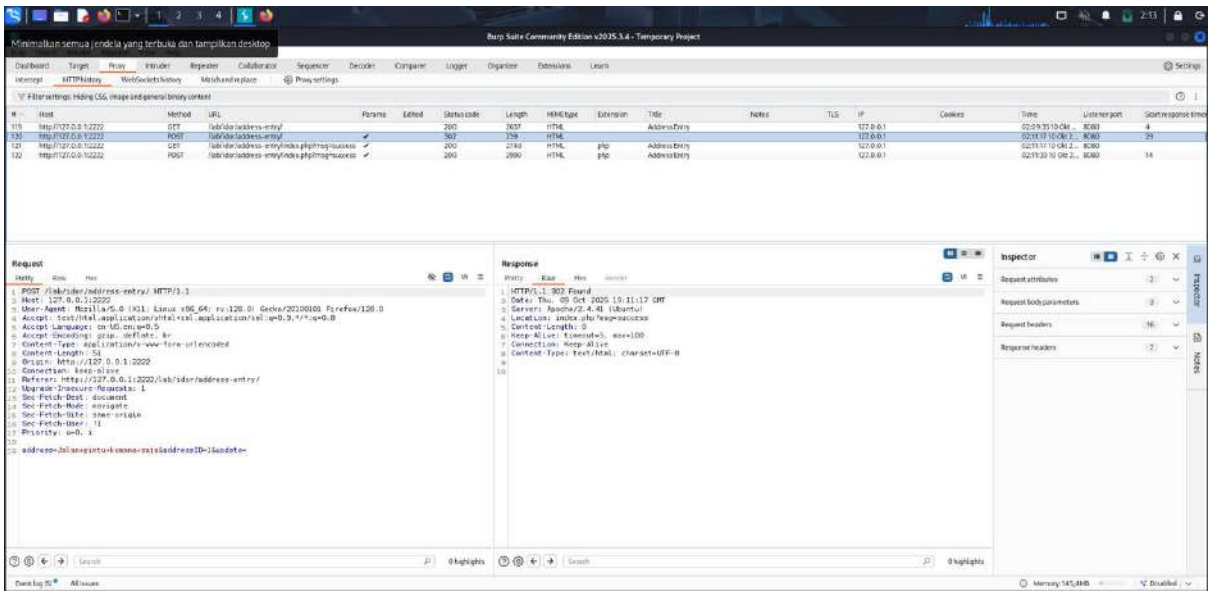

VulnLab
Address Entry
Go Back
Source Code
English

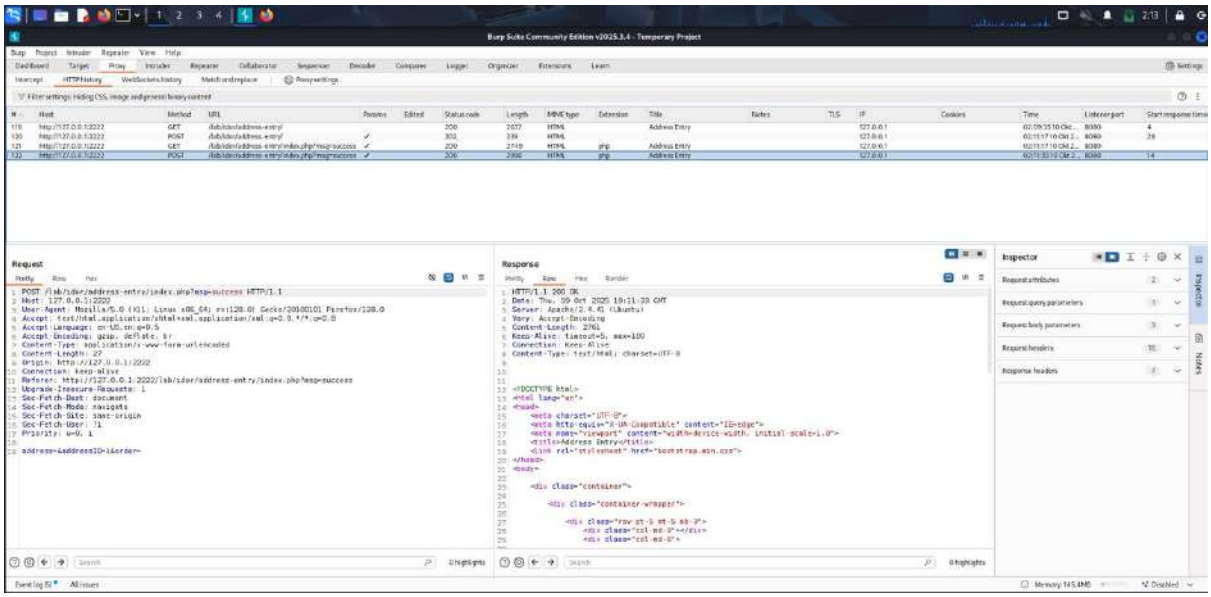

Address Entry
Reset
My name: Jesus S. Green
My registered address: Jalan pintu kemana saja
Confirm Order - Enter Your Address
Address updated successfully!
Enter your address:
Enter your address
Update Address
Order


Firefox ESR
Browse the World Wide Web
VulnLab
Address Entry
Go Back
Source Code
English

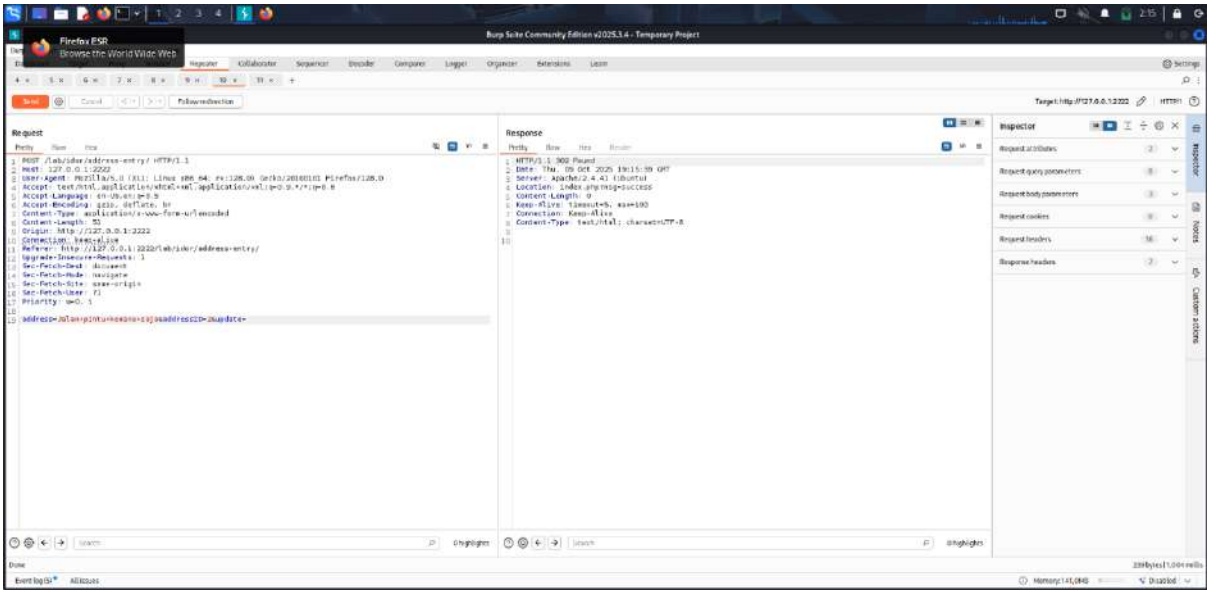

Address Entry
Reset
My name: Jesus S. Green
My registered address: Jalan pintu kemana saja
Confirm Order - Enter Your Address
Address updated successfully!
Order placed successfully!
Order address: Jalan pintu kemana saja
Name: Jesus S. Green
Enter your address:
Enter your address
Update Address
Order

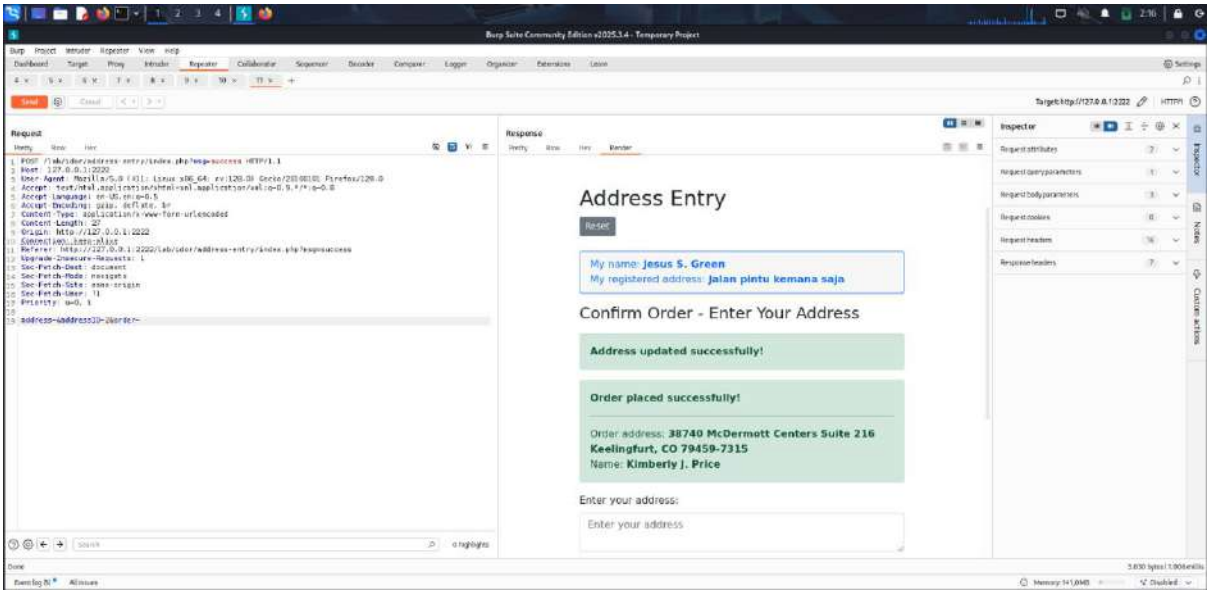
3. Saya mengulangi proses tersebut sambil mencegat permintaan dengan Burp Suite. Saya menemukan bahwa baik tindakan "Update Address" maupun "Order" mengirimkan permintaan POST. Keduanya menyertakan parameter addressId=1, yang secara langsung merujuk ke ID akun saya. Kehadiran parameter ini pada permintaan "Order" menunjukkan potensi kerentanan.





4. Saya mengirim permintaan "Order" yang telah ditangkap ke Burp Suite Repeater. Di Repeater, saya mengubah nilai parameter addressId dari 1 (akun saya) menjadi 2 (akun target). Permintaan yang dimodifikasi terlihat seperti berikut: Address=&addressId=2&order=.





5. Saya mengirimkan permintaan yang telah dimodifikasi. Server memberikan respons "Order placed successfully!". Menariknya, detail pesanan yang ditampilkan adalah untuk pengguna yang berbeda, yaitu Kimberly J. Price, dengan alamat pengiriman yang terdaftar di akunnya. Ini membuktikan bahwa saya berhasil memicu tindakan pemesanan atas nama pengguna lain hanya dengan mengubah nilai ID.Perhatikan bahwa alamat pengiriman yang ditampilkan dalam respons bukanlah milik "Jesus S. Green", melainkan milik "Kimberly J. Price", yang beralamat di "38740 McDermott Centers...". Ini membuktikan bahwa pesanan berhasil dialihkan ke alamat pengguna lain.

Untuk mengatasi kerentanan ini dan menjaga integritas proses pemesanan:

1. Gunakan Alamat dari Sesi Pengguna: Di sisi *server*, logika pemrosesan pesanan harus mengabaikan parameter address_id atau address apa pun yang dikirim dari klien. Alamat pengiriman harus diambil secara eksklusif dari database berdasarkan ID pengguna yang terkait dengan sesi yang sedang aktif.

2. Validasi Kepemilikan Alamat: Jika aplikasi mengizinkan pengguna memiliki banyak alamat, saat pengguna memilih satu alamat untuk pengiriman, *backend* harus tetap memvalidasi bahwa address_id yang dipilih memang milik pengguna yang sedang login sebelum melanjutkan proses.

2.4 Listing Database Contents (Non-Oracle)

Aplikasi web ini memiliki kerentanan SQL Injection pada fungsionalitas filter kategori produk. Parameter category pada URL tidak divalidasi dengan benar, sehingga memungkinkan penyerang untuk menyisipkan dan mengeksekusi *query* SQL berbahaya. Dengan menggunakan teknik UNION, penyerang dapat melakukan enumerasi skema database, menemukan tabel dan kolom yang berisi kredensial pengguna, dan akhirnya mengekstrak *username* dan *password* milik administrator untuk mendapatkan akses penuh ke akun tersebut.

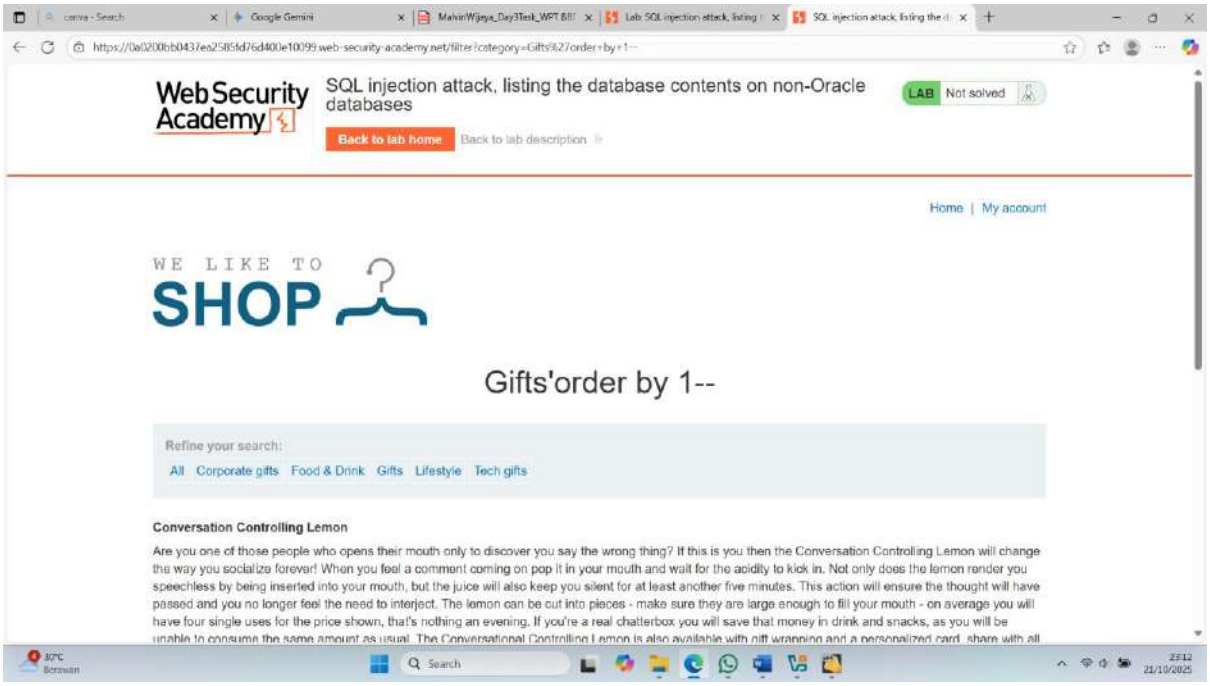
Url Affected/Endpoint	https://[LAB_ID].web-security-academy.net/filter?category=...
Severity	**Critical**
CWE	CWE-89: *Improper Neutralization of Special Elements used in an SQL Command*
OWASP	A03:2021-Injection
CVSS Score	9.8
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

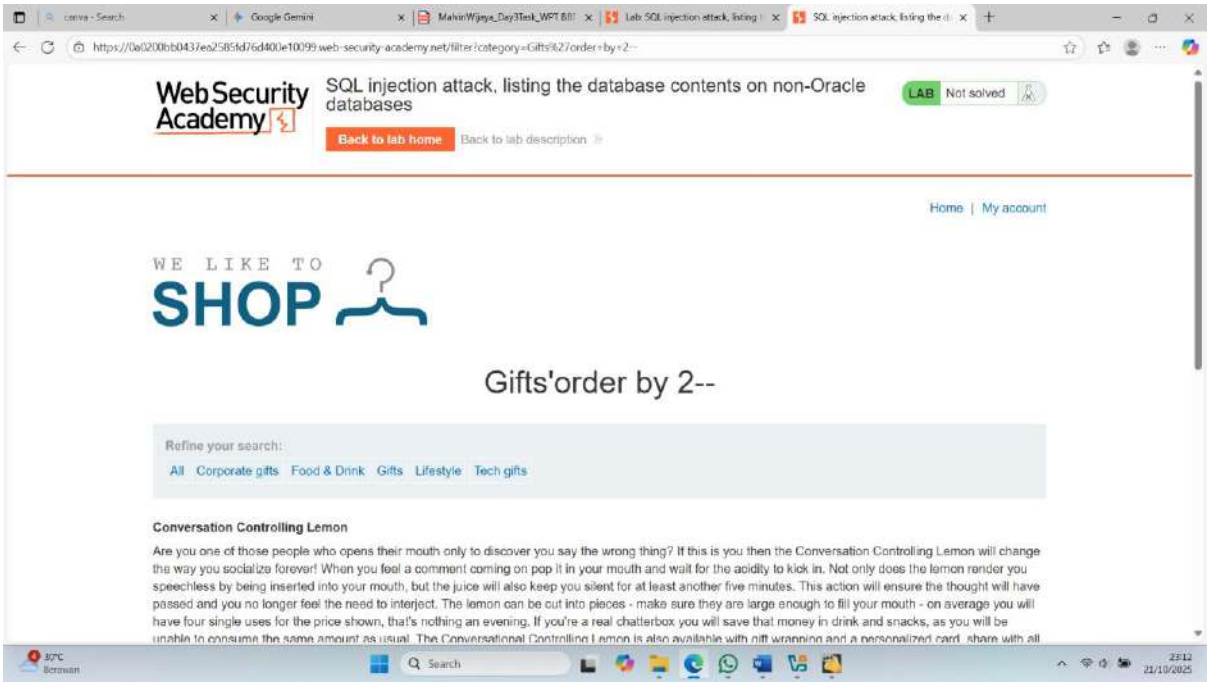
Dampak dari kerentanan ini sangat fatal, yaitu:

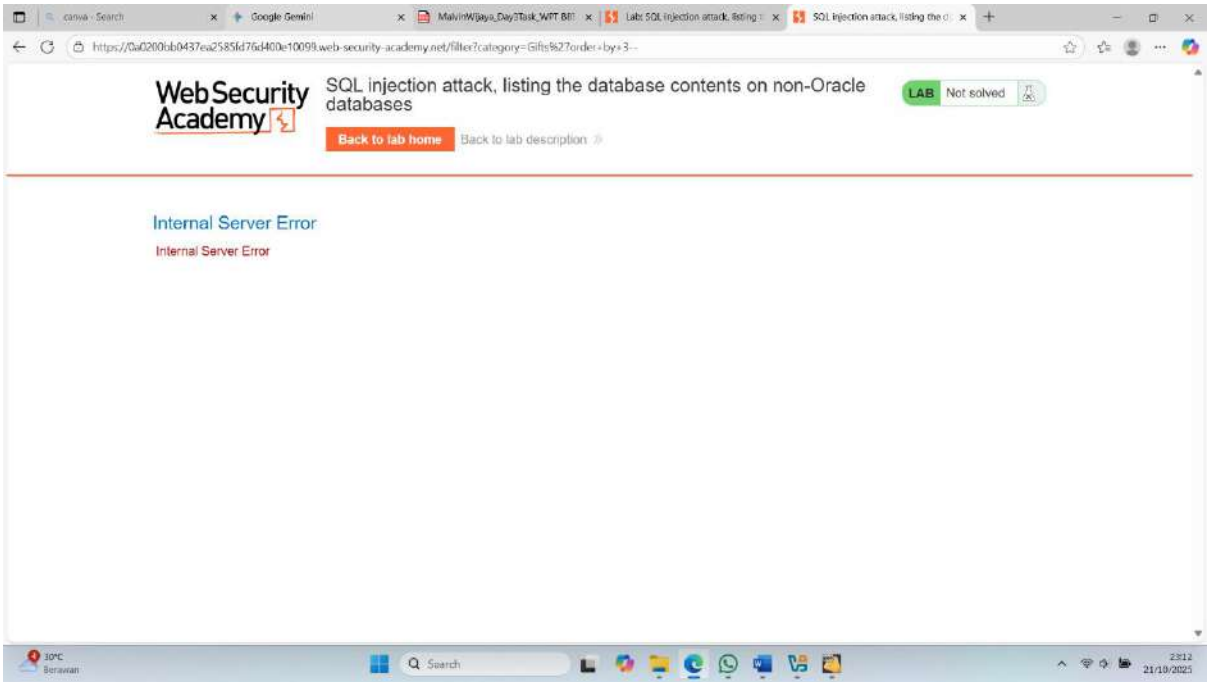
1. Kompromi Akun Administrator: Penyerang dapat memperoleh kredensial login administrator, memberikan mereka level akses tertinggi pada aplikasi.
2. Kebocoran Data Total: Dengan akses admin, seluruh data sensitif pengguna lain dapat diakses, diubah, atau dihapus.
3. Pengambilalihan Aplikasi Penuh: Penyerang berpotensi mengontrol fungsionalitas inti aplikasi, merusak data, atau melakukan tindakan merugikan lainnya.

Proses eksploitasi dilakukan secara bertahap untuk memetakan database dan mengekstrak data yang diinginkan.

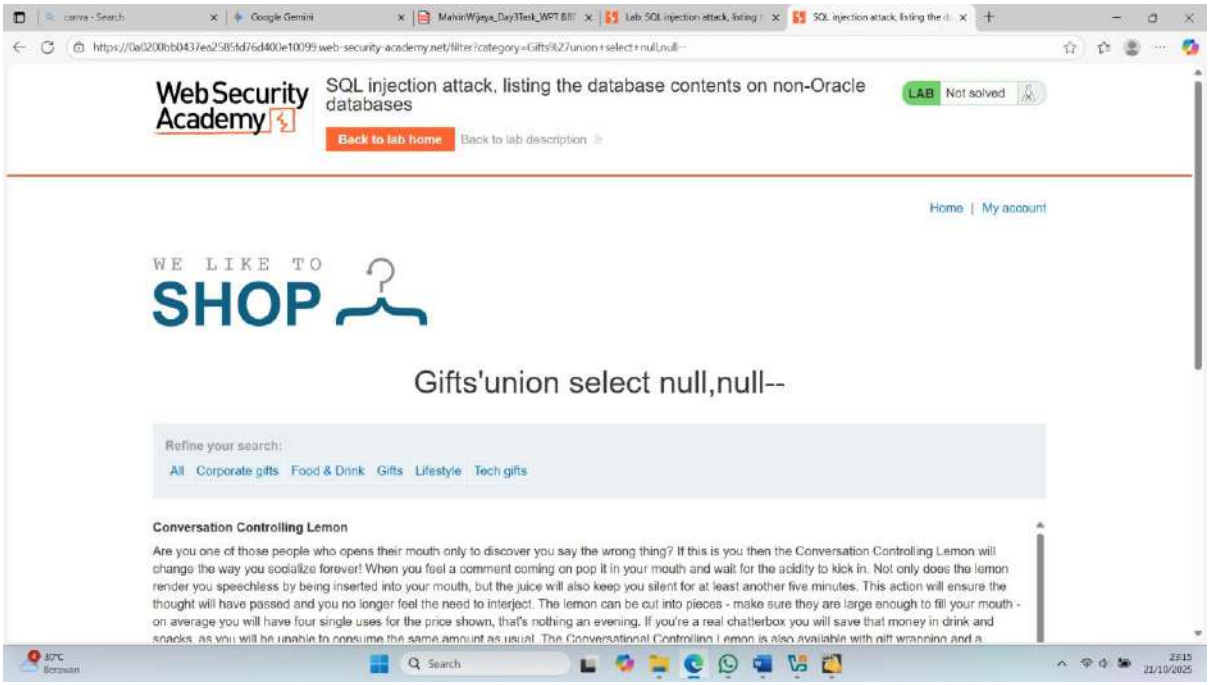
1. Menentukan Jumlah Kolom: *Payload* ORDER BY digunakan untuk menentukan jumlah kolom yang dikembalikan oleh *query* asli. Percobaan ' ORDER BY 3-- menghasilkan *error*, yang mengindikasikan bahwa *query* hanya mengembalikan 2 kolom.

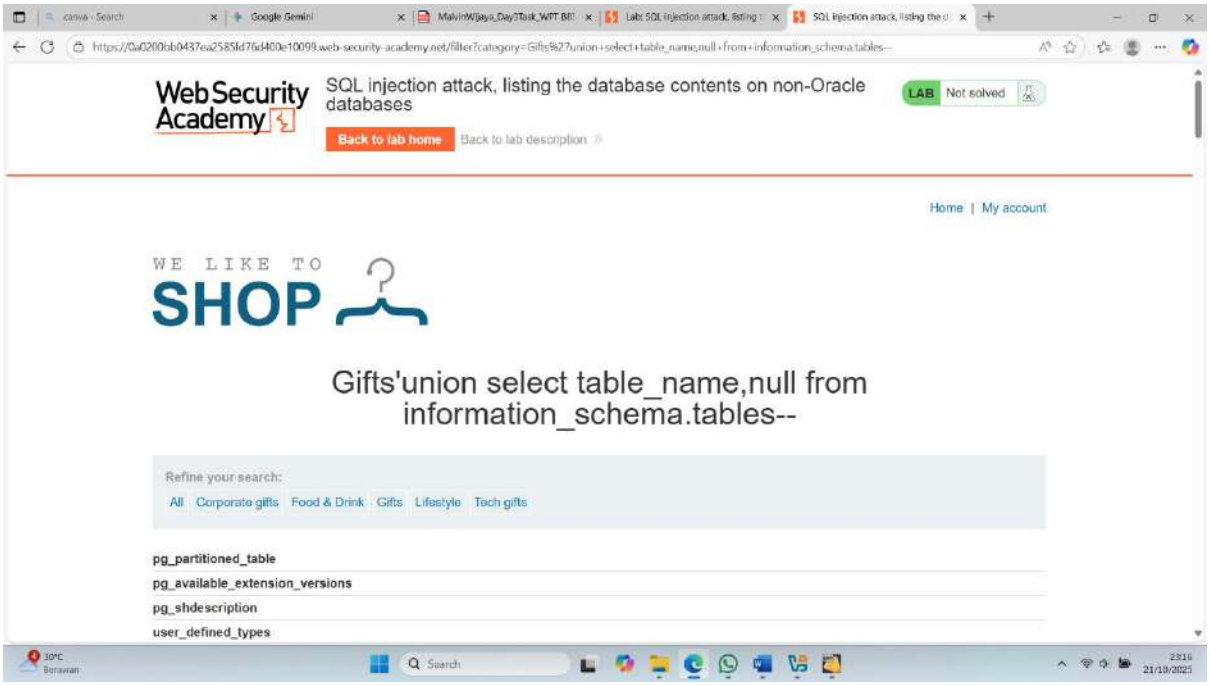

https://0a0200bb0437ea2585fd76d400e10099.web-security-academy.net/filter?category=Gifts%27order+by+1--
Web Security Academy
SQL injection attack, listing the database contents on non-Oracle databases
LAB Not solved
Back to lab home
Back to lab description
Home | My account
WE LIKE TO SHOP
Gifts'order by 1--
Refine your search:
All Corporate gifts Food & Drink Gifts Lifestyle Tech gifts
Conversation Controlling Lemon
Are you one of those people who opens their mouth only to discover you say the wrong thing? If this is you then the Conversation Controlling Lemon will change the way you socialize forever! When you feel a comment coming on pop it in your mouth and wait for the acidity to kick in. Not only does the lemon render you speechless by being inserted into your mouth, but the juice will also keep you silent for at least another five minutes. This action will ensure the thought will have passed and you no longer feel the need to interject. The lemon can be cut into pieces - make sure they are large enough to fill your mouth - on average you will have four single uses for the price shown, that's nothing an evening. If you're a real chatterbox you will save that money in drink and snacks, as you will be

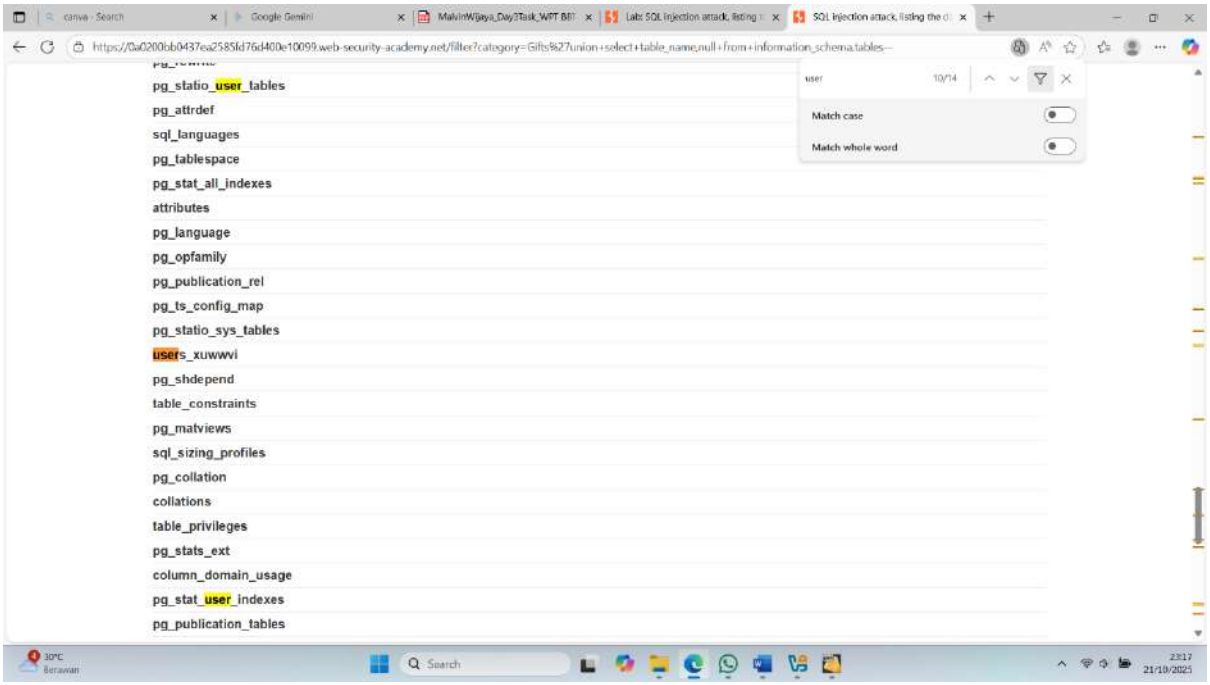

https://0a0200bb0437ea2585fd76d400e10099.web-security-academy.net/filter?category=Gifts%27order+by+2--
Web Security Academy
SQL injection attack, listing the database contents on non-Oracle databases
LAB Not solved
Back to lab home
Back to lab description
Home | My account
WE LIKE TO SHOP
Gifts'order by 2--
Refine your search:
All Corporate gifts Food & Drink Gifts Lifestyle Tech gifts
Conversation Controlling Lemon
Are you one of those people who opens their mouth only to discover you say the wrong thing? If this is you then the Conversation Controlling Lemon will change the way you socialize forever! When you feel a comment coming on pop it in your mouth and wait for the acidity to kick in. Not only does the lemon render you speechless by being inserted into your mouth, but the juice will also keep you silent for at least another five minutes. This action will ensure the thought will have passed and you no longer feel the need to interject. The lemon can be cut into pieces - make sure they are large enough to fill your mouth - on average you will have four single uses for the price shown, that's nothing an evening. If you're a real chatterbox you will save that money in drink and snacks, as you will be



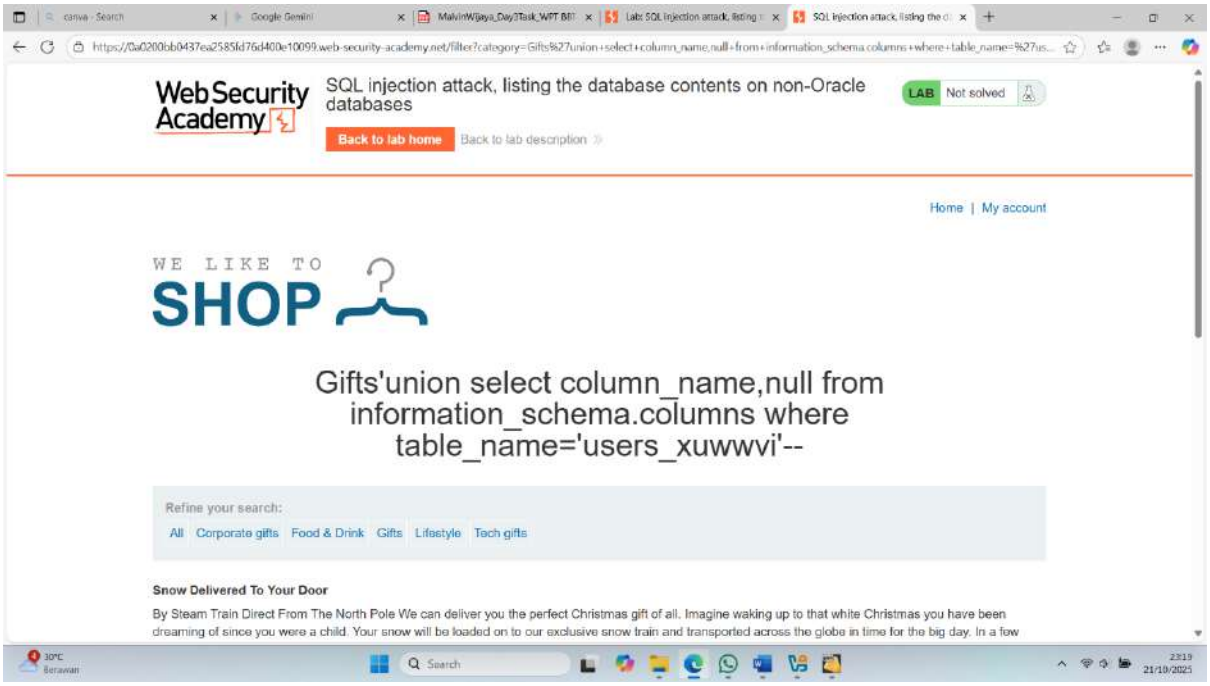
2. Mencari Tabel Pengguna: Dengan mengetahui jumlah kolom, *payload* UNION digunakan untuk membaca skema database. *Query* ke information_schema.tables berhasil menampilkan semua nama tabel. Dari daftar tersebut, ditemukan tabel yang relevan bernama users_xuwwvi.

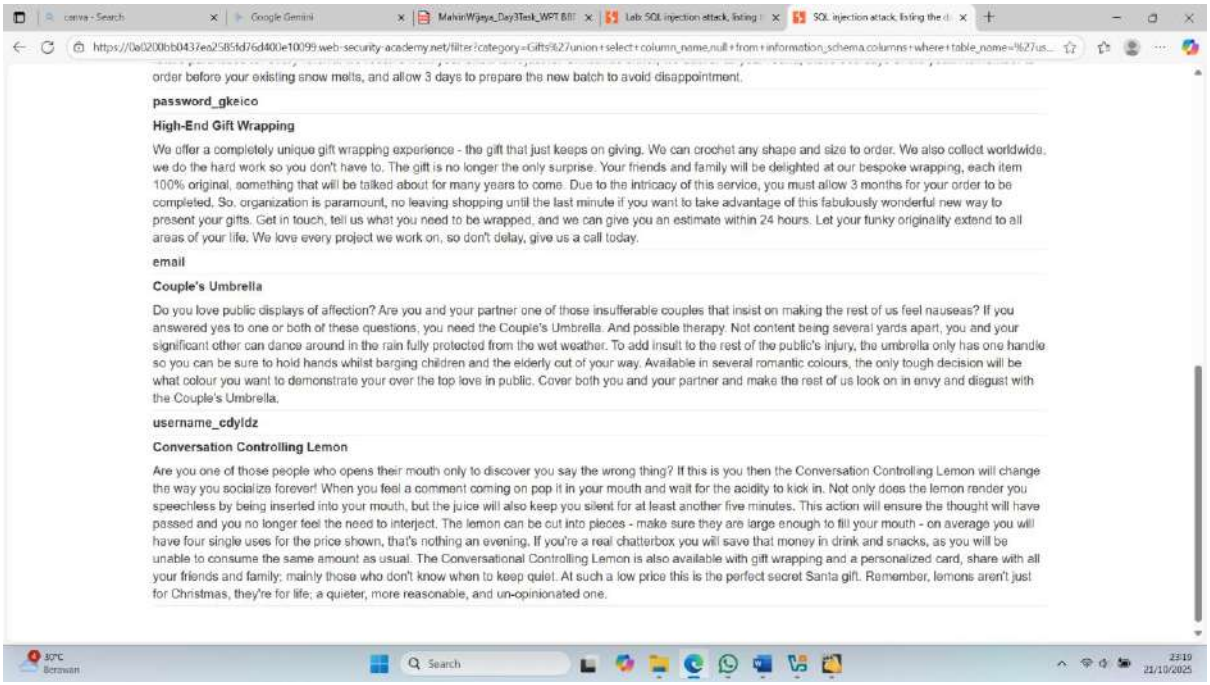




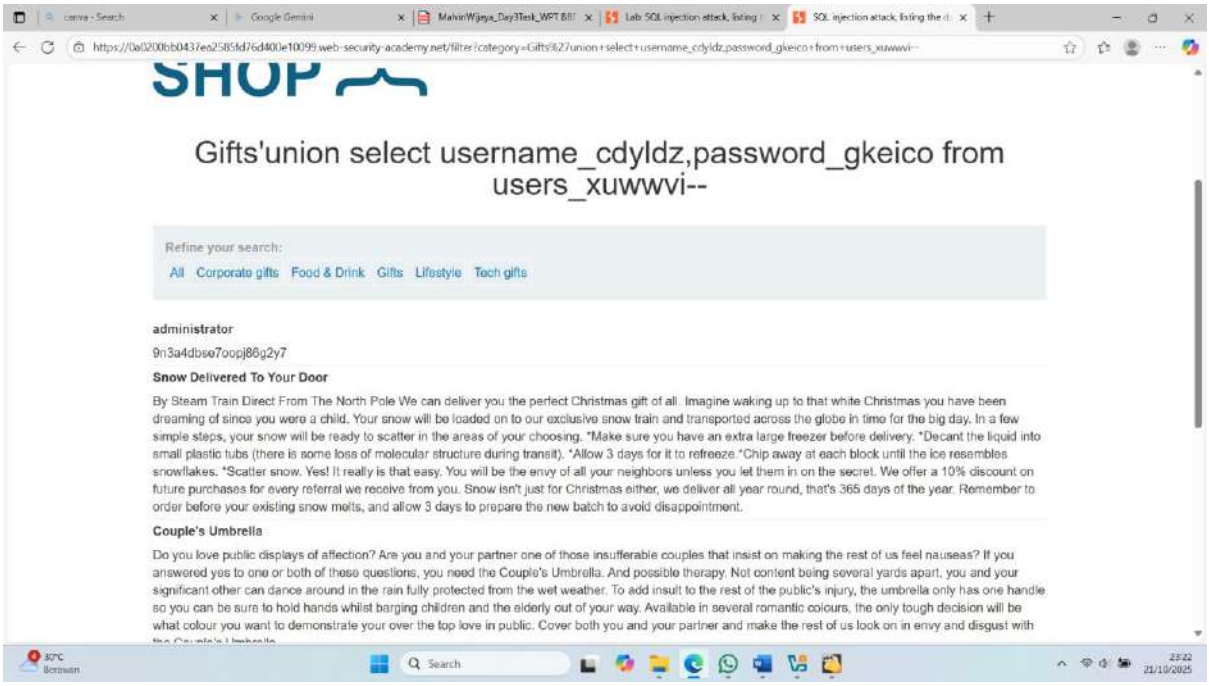


3. Mencari Kolom Kredensial: Setelah tabel target ditemukan, *query* selanjutnya ditujukan ke information_schema.columns untuk mendaftar semua kolom di dalam tabel users_xuwwvi. Dari sini, ditemukan dua kolom yang relevan: username_ctyldz dan password_gkeico.

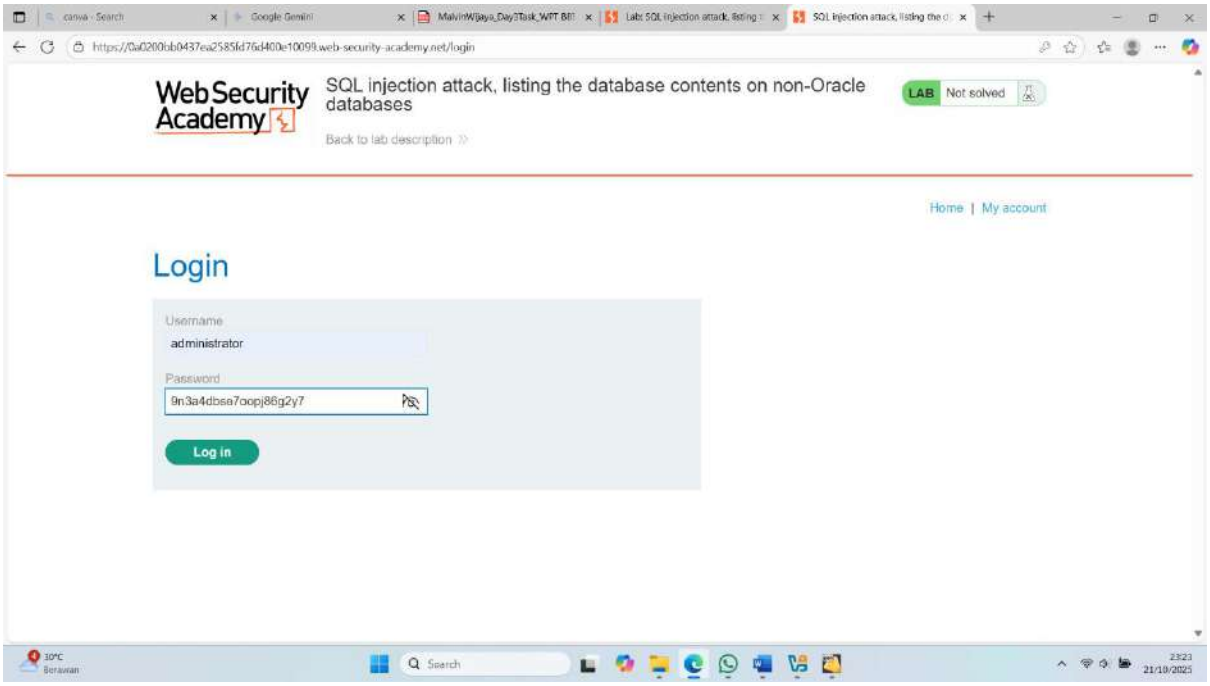




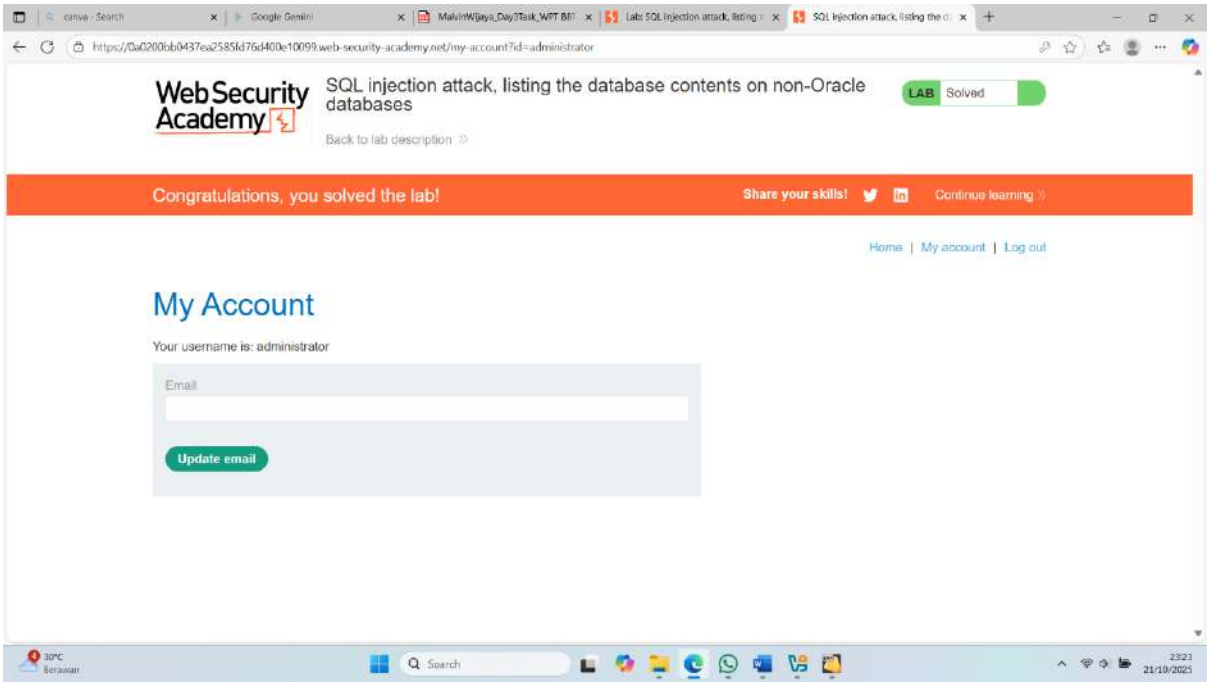
4. Mengekstrak Data Kredensial: *Payload* UNION terakhir dibuat untuk memilih data langsung dari kedua kolom yang telah ditemukan di dalam tabel users_xuwwvi. Hasilnya berhasil menampilkan kredensial untuk pengguna administrator.



5. Login Sebagai Administrator: Kredensial yang ditemukan (administrator dan passwordnya) kemudian digunakan untuk login ke aplikasi.



6. Konfirmasi Keberhasilan: Setelah login, aplikasi mengonfirmasi bahwa akses berhasil didapatkan dan lab dinyatakan "Solved".



Untuk mencegah serangan SQL Injection, metode pertahanan yang paling efektif adalah:

1. Gunakan Parameterized Queries (Prepared Statements): Ini adalah pendekatan utama yang harus selalu digunakan. Dengan memisahkan *query* SQL dari data, input pengguna tidak akan pernah bisa dieksekusi sebagai perintah, sehingga serangan injeksi menjadi tidak mungkin.
2. Hindari Membangun Query Secara Manual: Jangan pernah menggabungkan *string* untuk membuat *query* SQL, karena ini adalah akar dari sebagian besar kerentanan injeksi. Selalu gunakan *framework* yang aman yang mendukung *prepared statements*.

2.5 Listing Database Contents (Oracle)

Aplikasi web ini memiliki kerentanan SQL Injection pada parameter category yang berinteraksi dengan *backend* database Oracle. Kurangnya sanitasi input memungkinkan penyerang untuk memanipulasi *query* menggunakan teknik UNION. Dengan memanfaatkan *data dictionary* bawaan Oracle (seperti all_tables dan all_tab_columns), penyerang dapat memetakan struktur database, mengidentifikasi tabel yang menyimpan kredensial pengguna, dan mengekstrak *username* serta *password* milik administrator untuk mengambil alih akun tersebut.

Url Affected/Endpoint	https://[LAB_ID].web-security-academy.net/filter?category=[INJECTION_POINT]
Severity	**Critical**
CWE	CWE-89: *Improper Neutralization of Special Elements used in an SQL Command*

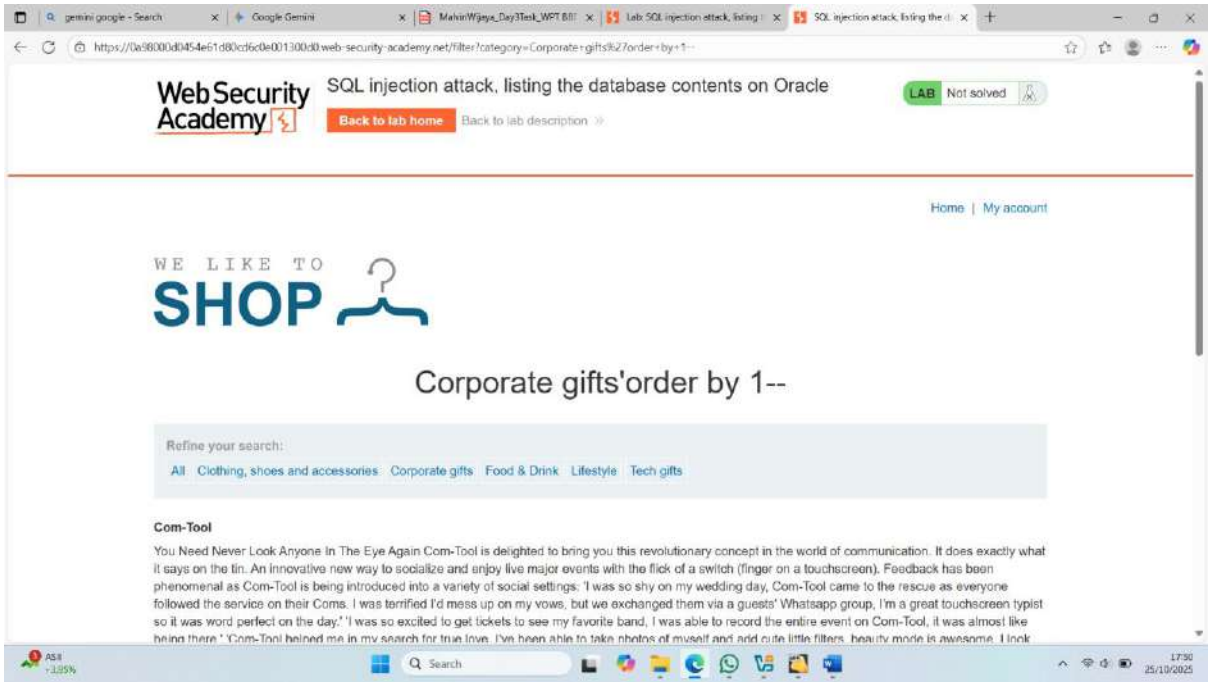
OWASP	A03:2021-Injection
CVSS Score	9.8
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

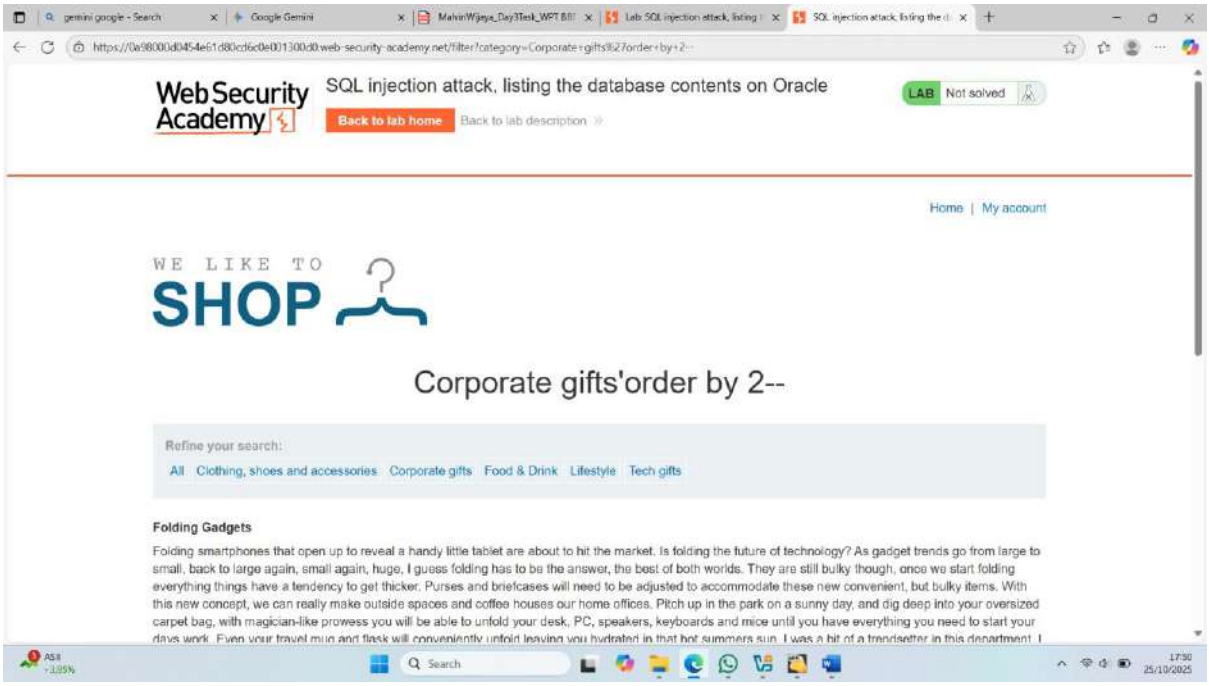
Serupa dengan lab sebelumnya, dampak dari kerentanan ini sangat fatal:

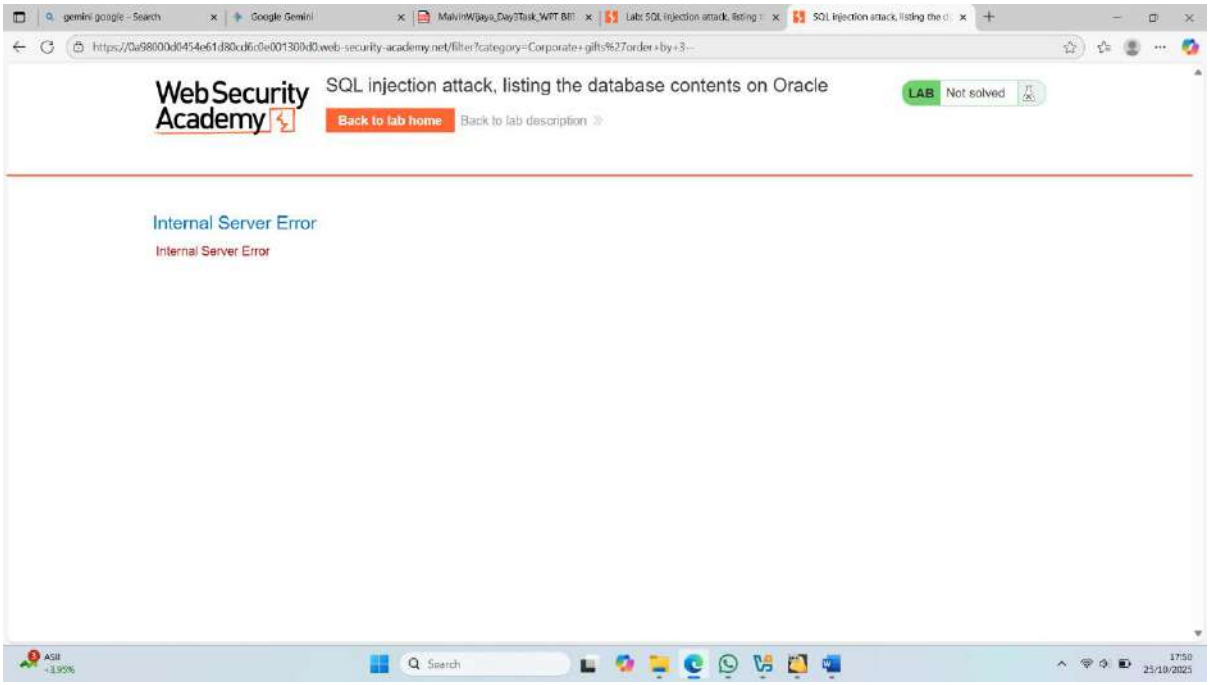
1. Kompromi Akun Administrator: Penyerang dapat memperoleh kredensial login administrator, memberikan mereka level akses tertinggi pada aplikasi.
2. Kebocoran Data Total: Seluruh data sensitif pengguna lain dapat diakses, diubah, atau dihapus oleh penyerang.

Proses eksploitasi ini memerlukan adaptasi *payload* agar sesuai dengan sintaks database Oracle.

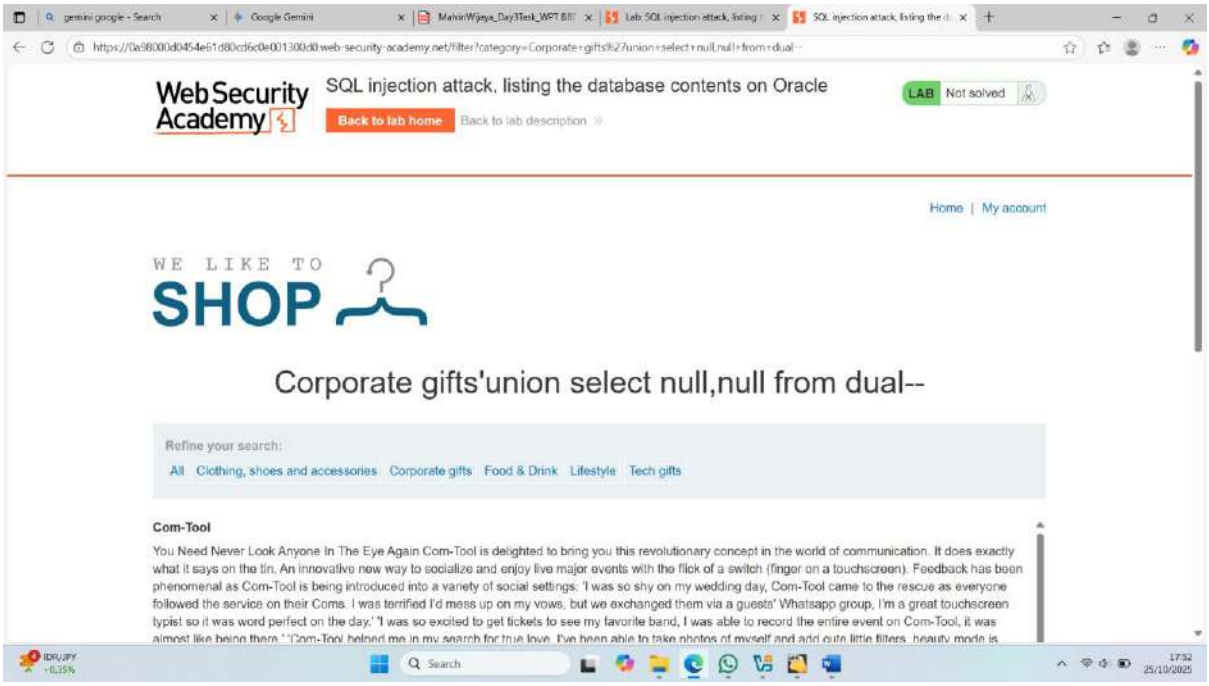
1. Menentukan Jumlah Kolom: *Payload* ORDER BY digunakan untuk menentukan jumlah kolom. Percobaan ' ORDER BY 3-- menghasilkan *Internal Server Error*, sedangkan ' ORDER BY 2-- berhasil. Ini mengonfirmasi bahwa *query* asli mengembalikan 2 kolom.



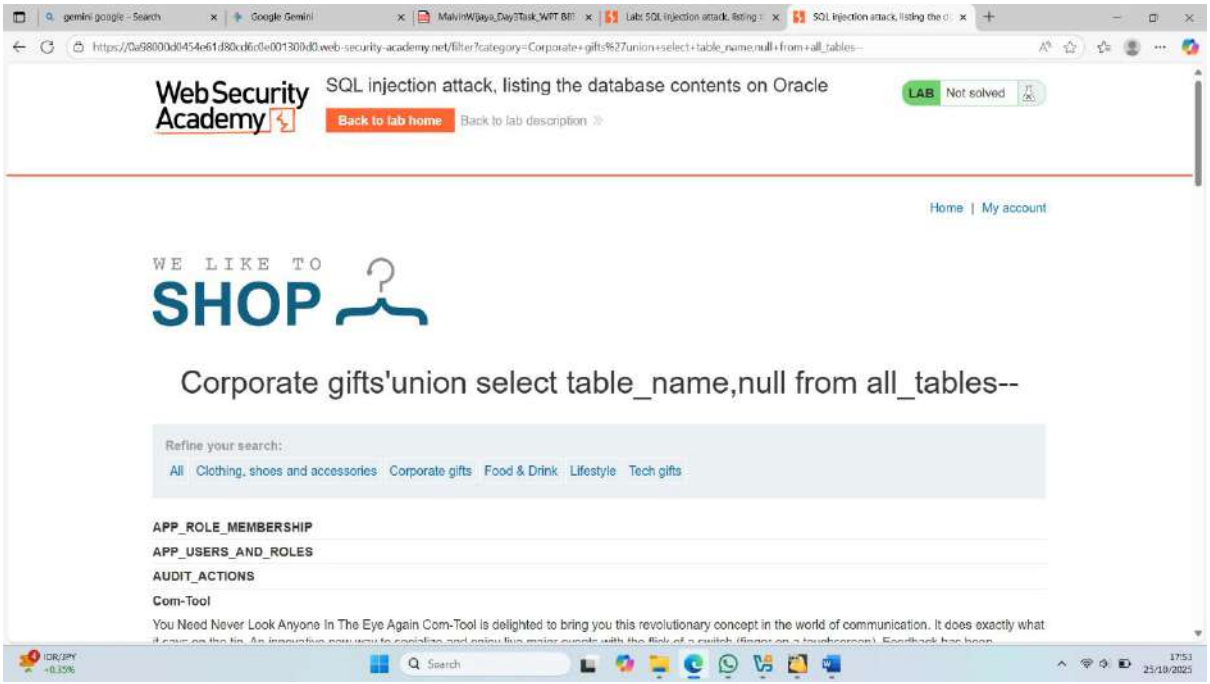


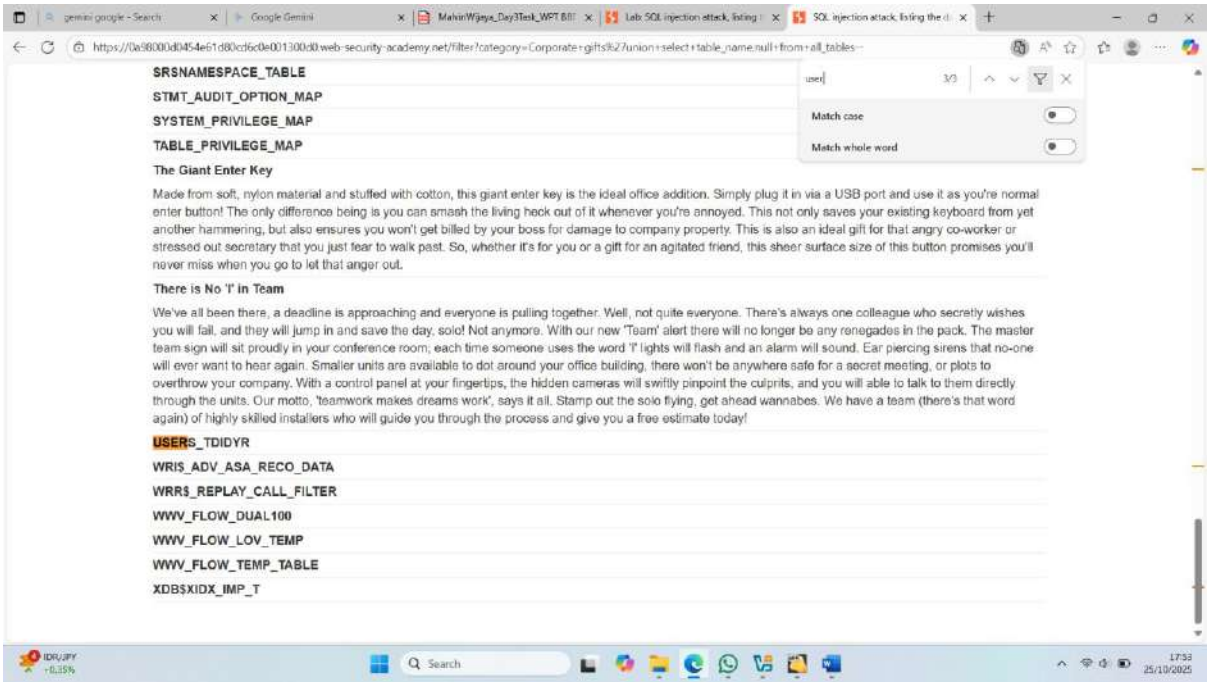


2. Mengonfirmasi Sintaks UNION: Database Oracle memerlukan *query* SELECT dalam UNION untuk menyertakan tabel FROM yang valid. Tabel dual digunakan untuk tujuan ini. *Payload* ' UNION SELECT NULL,NULL FROM dual-- berhasil dieksekusi, mengonfirmasi kerentanan.

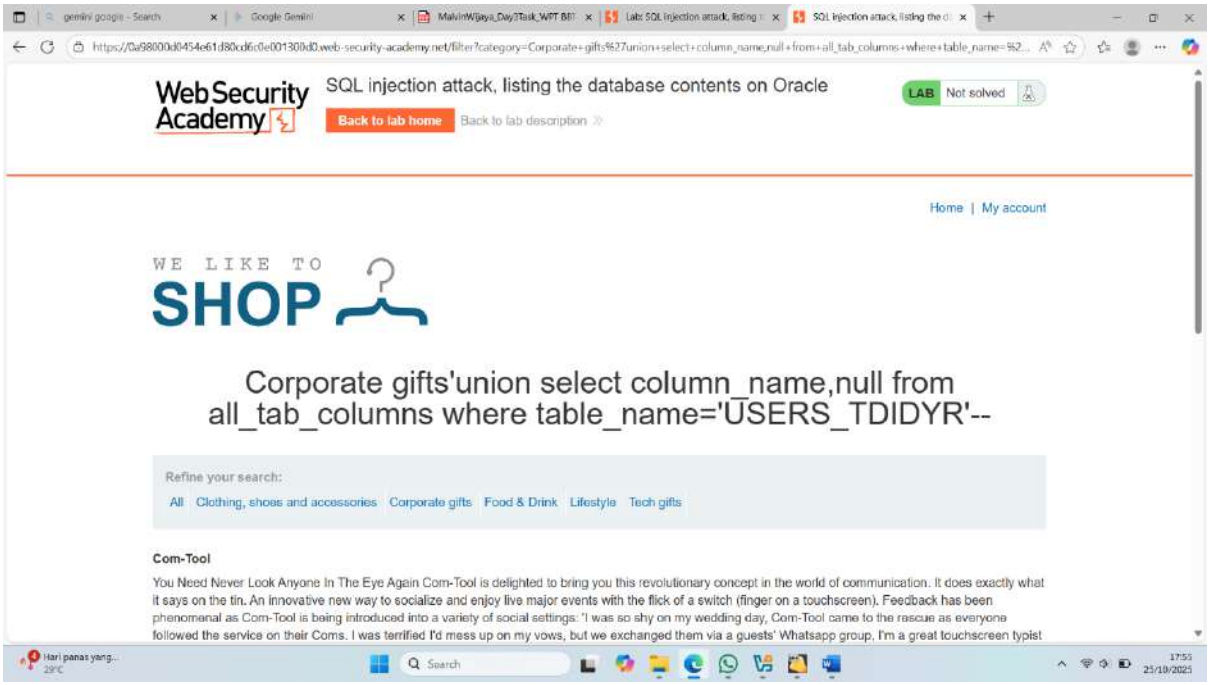


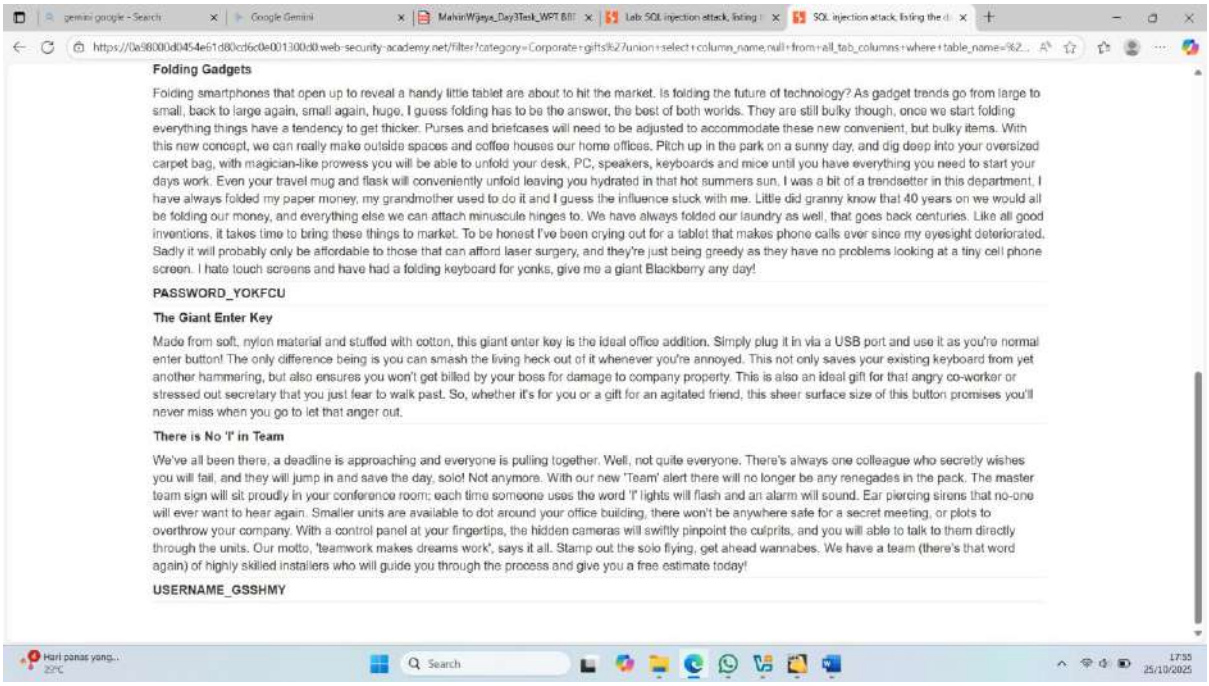
3. Mencari Tabel Pengguna: *Payload* UNION dimodifikasi untuk membaca *data dictionary* all_tables dan mendaftar semua nama tabel. Dari daftar yang panjang, tabel yang relevan berhasil diidentifikasi, yaitu USERS_TDIDYR.



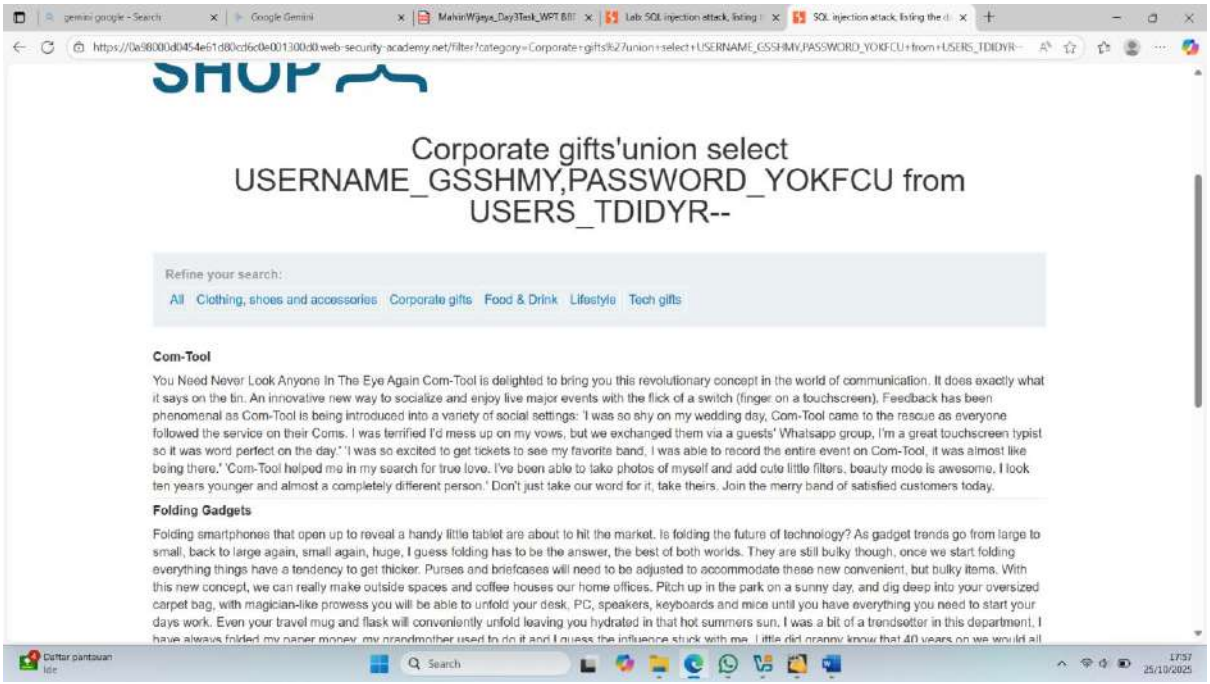


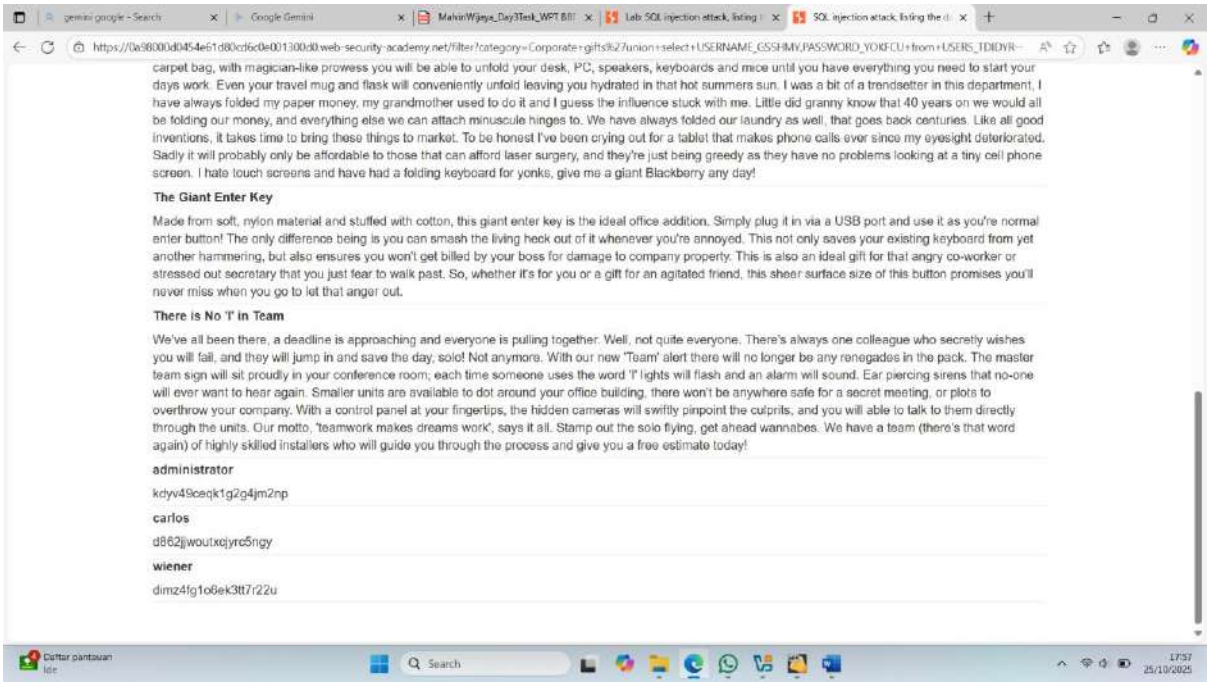
4. Mencari Kolom Kredensial: Setelah tabel target diketahui, *query* selanjutnya ditujukan ke all_tab_columns untuk mendaftar semua kolom di dalam tabel USERS_TDIDYR. Dari sini, ditemukan dua kolom kredensial: USERNAME_GSSHMY dan PASSWORD_YOKFCU.



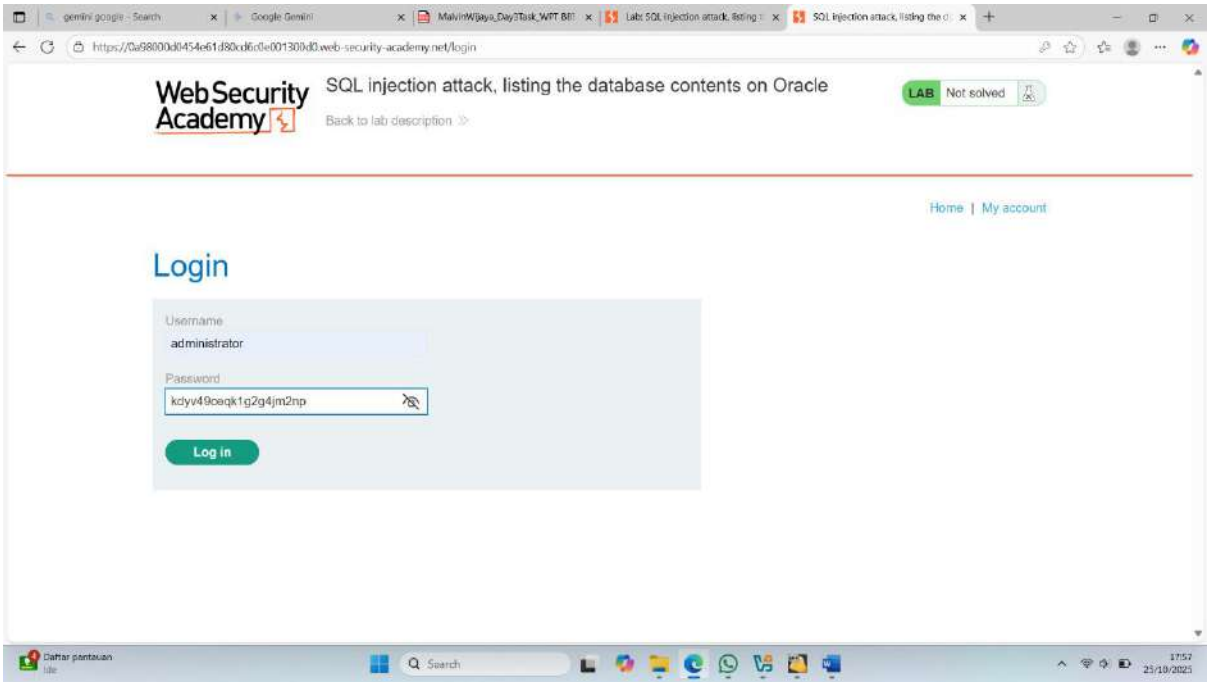


5. Mengekstrak Data Kredensial: *Payload* UNION terakhir dibuat untuk memilih data langsung dari kedua kolom tersebut di dalam tabel USERS_TDIDYR. *Payload* ini berhasil mengekstrak daftar *username* dan *password* dari semua pengguna.

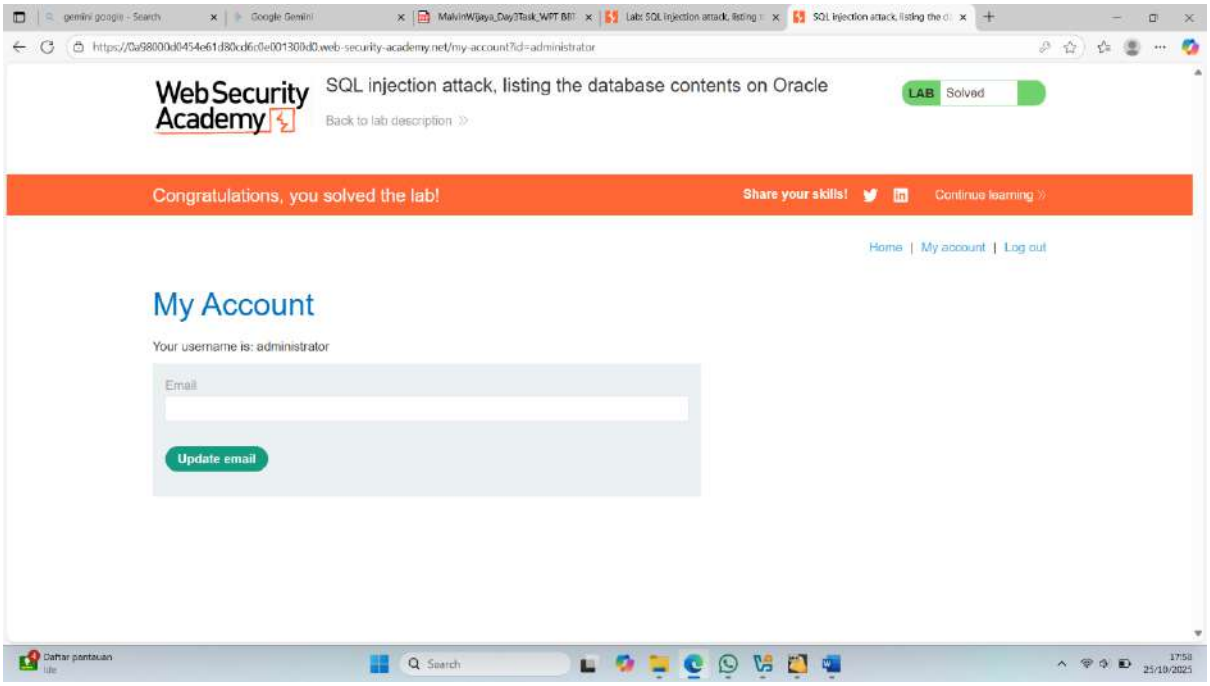




6. Login Sebagai Administrator: Kredensial milik administrator yang ditemukan (kdyv49ceqk1g2g4jm2np) digunakan untuk login ke aplikasi.



7. Konfirmasi Keberhasilan: Aplikasi berhasil memvalidasi kredensial tersebut, memberikan akses ke akun administrator, dan lab dinyatakan "Solved".



Mitigasi untuk kerentanan ini tetap sama, terlepas dari jenis database *backend*-nya:

1. Gunakan Parameterized Queries (Prepared Statements): Ini adalah metode pertahanan paling efektif dan universal untuk mencegah semua jenis serangan SQL Injection.
2. Validasi Input: Terapkan validasi input yang ketat untuk memastikan input sesuai dengan format yang diharapkan (misalnya, daftar kategori yang diizinkan).

2.6 Path Traversal

Bagian ini akan mendokumentasikan pengerjaan serangkaian lab yang berfokus pada kerentanan *Path Traversal*. Kerentanan ini memungkinkan penyerang untuk membaca file-file di luar direktori yang seharusnya diakses oleh aplikasi web.

2.6.1 File path traversal, simple case

web ini memiliki kerentanan *Path Traversal* (juga dikenal sebagai *directory traversal*) pada fungsionalitas untuk menampilkan gambar produk. Aplikasi menggunakan parameter filename di URL untuk menentukan file gambar mana yang akan diambil dari *filesystem* server. Namun, aplikasi ini gagal melakukan sanitasi atau validasi yang memadai pada input yang diberikan. Penyerang dapat menyisipkan sekuens "dot-dot-slash" (../) ke dalam parameter filename untuk menavigasi ke direktori induk dan membaca file-file sensitif dari sistem operasi server, seperti /etc/passwd.

Url Affected/Endpoint	https://[LAB_ID].web-security-academy.net/image?filename=[INJECTION_POINT]
Severity	**High**

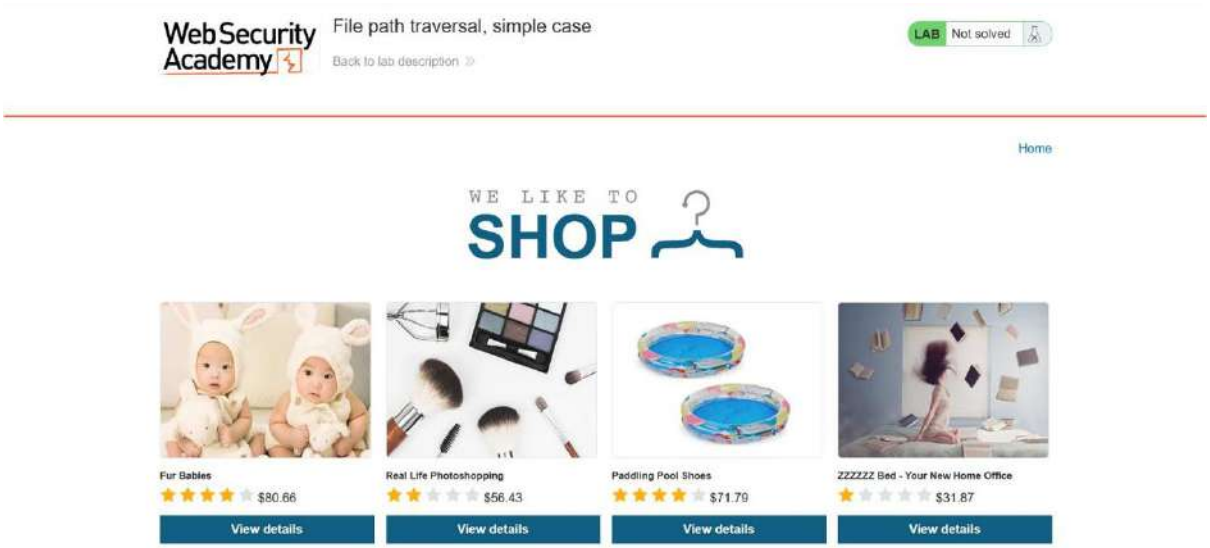
CWE	CWE-22: *Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')*
OWASP	A01:2021-Broken Access Control
CVSS Score	7.5
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

Dampak dari kerentanan ini sangat serius. Penyerang dapat memperoleh akses baca ke file-file sensitif di server, yang dapat mencakup:

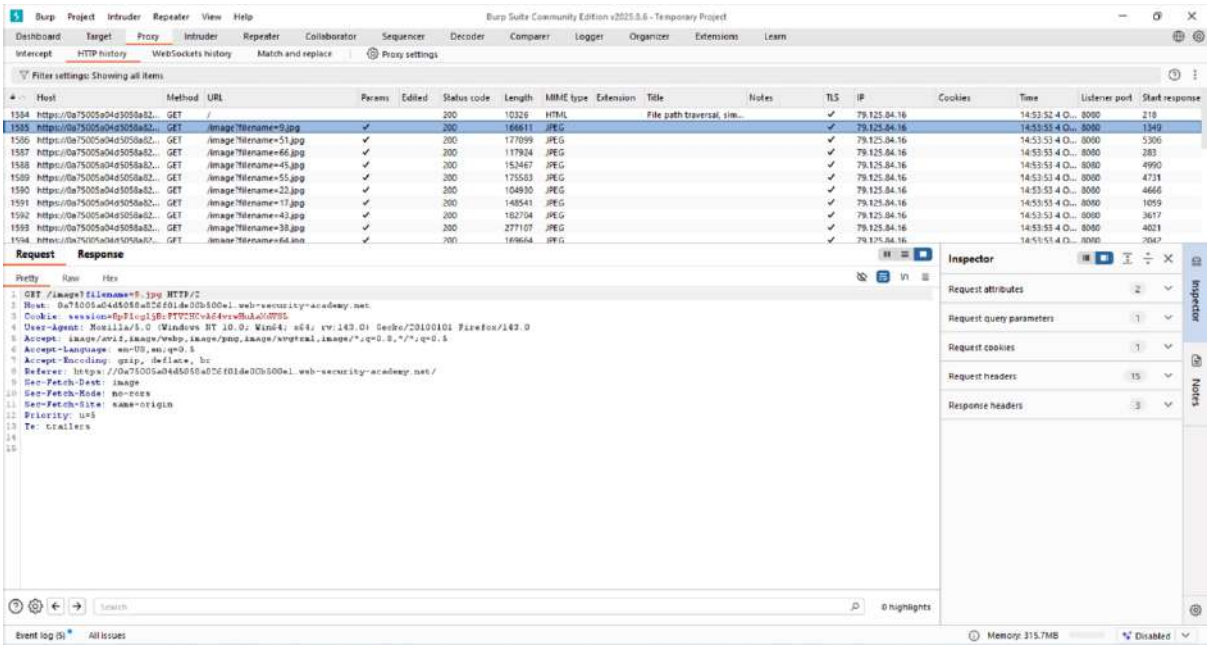
1. Informasi Pengguna Sistem: Seperti file /etc/passwd yang mendaftar semua pengguna di sistem.
2. Kode Sumber Aplikasi: Penyerang dapat mencuri kode sumber untuk menemukan kerentanan lain.
3. File Konfigurasi: Membaca file konfigurasi yang mungkin berisi *database credentials*, *API keys*, atau rahasia lainnya.
4. Data Aplikasi: Membaca file data lain yang disimpan di server.

Step to Reproduce with POC (Proof of Concept)

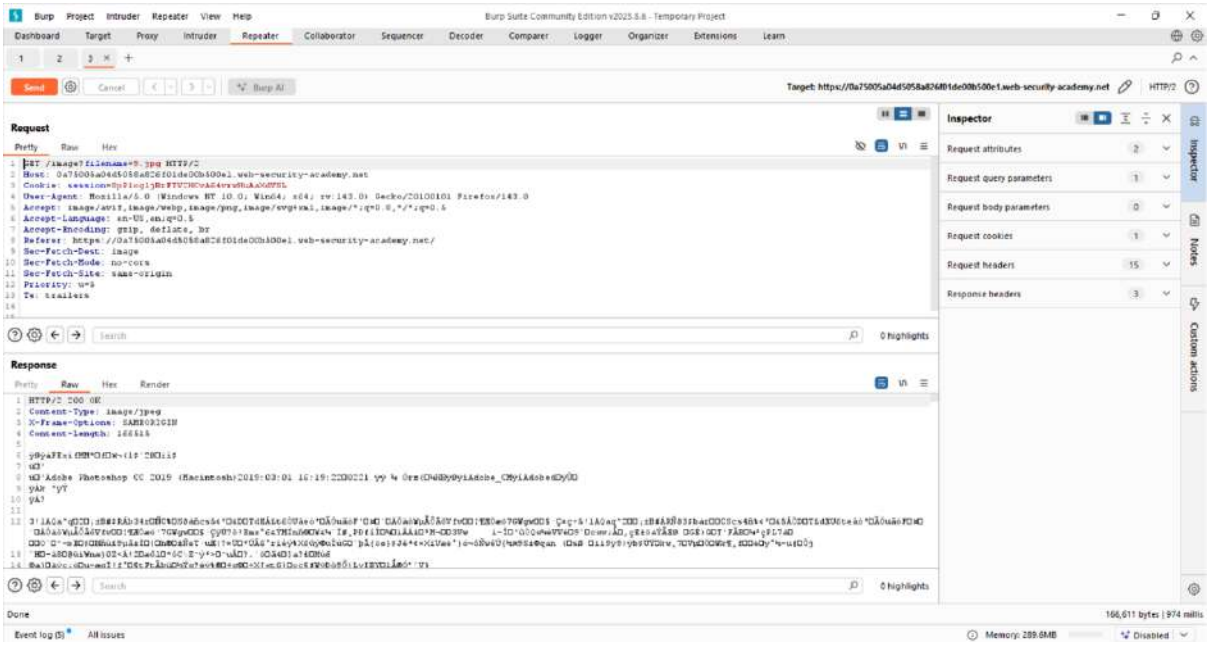
1. Buka halaman lab. Terlihat sebuah situs e-commerce yang menampilkan gambar-gambar produk.



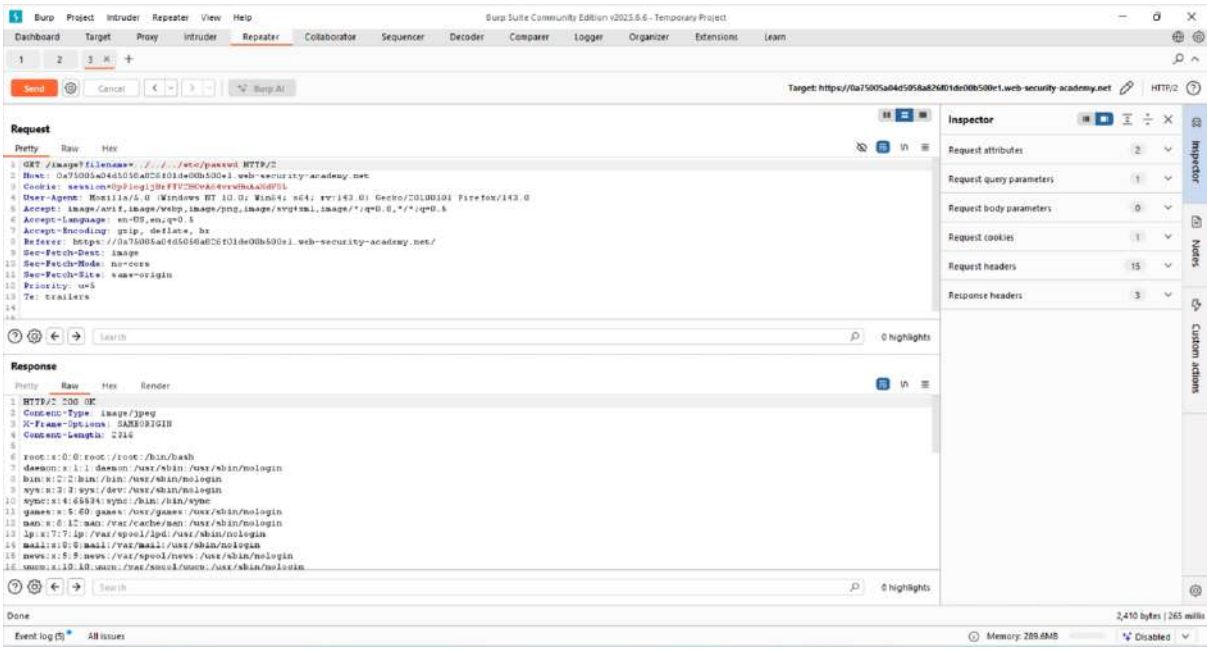
2. Gunakan Burp Suite untuk mencegat lalu lintas. Terlihat bahwa gambar-gambar diambil menggunakan *endpoint* /image dengan parameter filename (contoh: /image?filename=9.jpg).



3. Kirim salah satu *request* yang sah ke Burp Suite Repeater untuk dijadikan *baseline*. Saat dikirim, *request* ini mengembalikan data gambar yang valid.

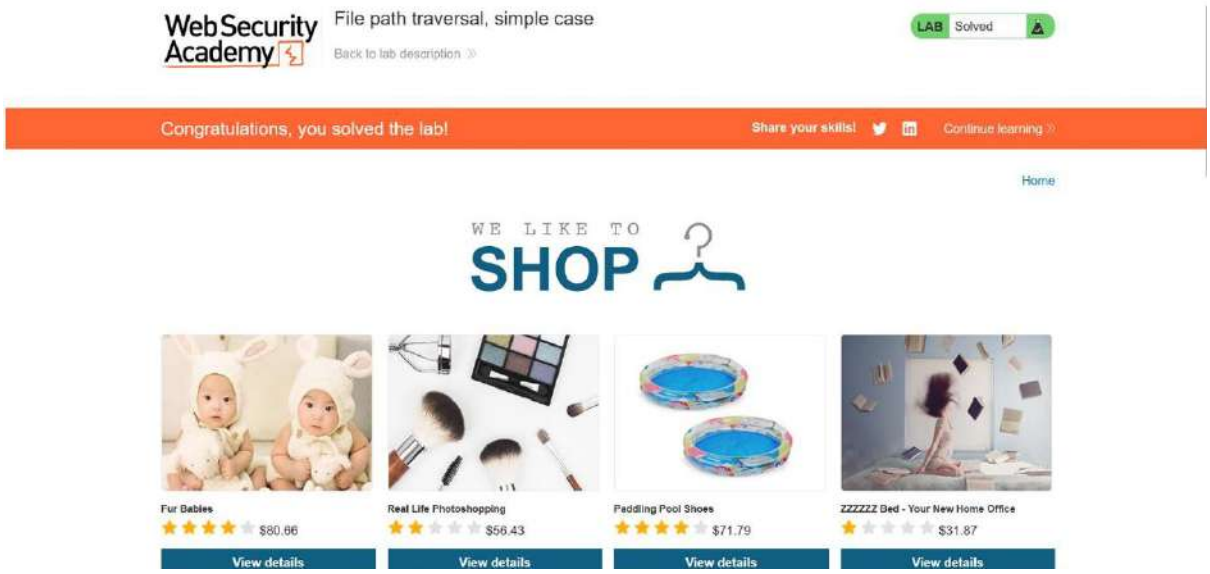


4. Modifikasi nilai parameter filename dengan *payload Path Traversal* sederhana untuk mencoba membaca file /etc/passwd. *Payload* yang digunakan adalah: ../../../etc/passwd.



5. Kirim *request* yang telah dimanipulasi. Server merespons dengan HTTP 200 OK dan di dalam *response body*, terlihat isi dari file /etc/passwd, yang mengonfirmasi keberhasilan eksploitasi.

6. Setelah *request* berhasil, halaman lab otomatis diperbarui menjadi "Solved".



Untuk mencegah kerentanan *Path Traversal*, langkah-langkah berikut harus diimplementasikan:

1. Validasi Input (Allow-List): Metode terbaik adalah dengan tidak menerima input berupa nama file dari pengguna. Jika harus, terapkan *allow-list* yang ketat yang hanya mengizinkan karakter alfanumerik dan ekstensi file yang diharapkan (misal: .png, .jpg). Blokir semua karakter lain, terutama . dan /.

2. Sanitasi Input: Jika *allow-list* tidak memungkinkan, lakukan sanitasi input untuk menghapus atau mengganti sekuens berbahaya seperti ../ dan ..\.

3. Gunakan Path Absolut yang Aman: Di sisi *backend*, gabungkan input yang *telah divalidasi* dengan path direktori dasar yang telah di-*hardcode* (misal: /var/www/images/). Pastikan path yang dihasilkan tidak pernah bisa keluar dari direktori yang diinginkan.
4. Prinsip Hak Akses Terkecil: Konfigurasikan aplikasi web agar berjalan dengan akun pengguna yang memiliki izin *filesystem* seminimal mungkin. Akun tersebut seharusnya hanya memiliki izin baca ke direktori yang benar-benar dibutuhkannya (misal: /var/www/images/) dan tidak bisa membaca file sistem seperti /etc/passwd.

2.6.2 File path traversal, traversal sequences blocked with absolute path bypass

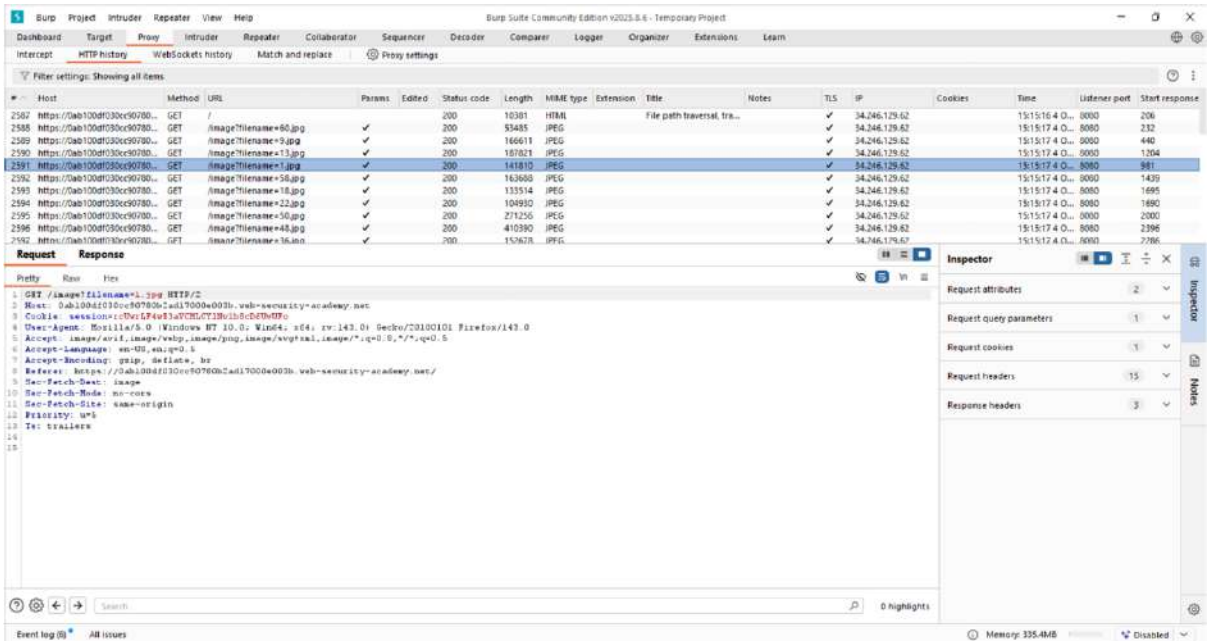
Aplikasi web ini mencoba menerapkan mekanisme pertahanan terhadap serangan *Path Traversal* dengan cara memblokir sekuens "dot-dot-slash" (`../`). Namun, mekanisme pertahanan ini tidak memadai. Aplikasi gagal memvalidasi atau memblokir *payload* yang menggunakan *absolute file path* (path yang dimulai dengan karakter `/`). Penyerang dapat memanfaatkan kelemahan ini dengan memberikan *absolute path* langsung ke file yang diinginkan (misalnya `/etc/passwd`) di dalam parameter `filename` untuk membaca file sistem yang sensitif.

Url Affected/Endpoint	https://[LAB_ID].web-security-academy.net/image?filename=[INJECTION_POINT]
Severity	**High**
CWE	CWE-22: *Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')*
OWASP	A01:2021-Broken Access Control
CVSS Score	7.5
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

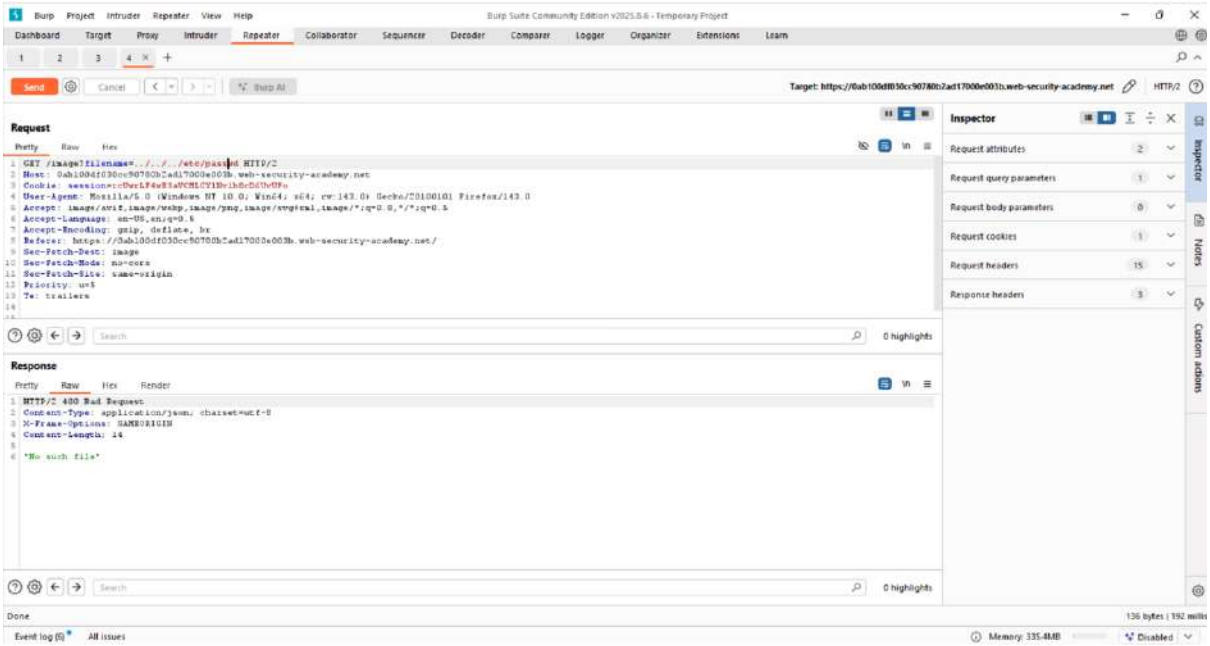
Meskipun ada upaya mitigasi, dampaknya tetap sama dengan kasus sederhana: penyerang dapat membaca file-file sensitif di server, yang dapat mencakup file konfigurasi, *credential*, kode sumber, atau informasi pengguna sistem.

Step to Reproduce with POC (Proof of Concept)

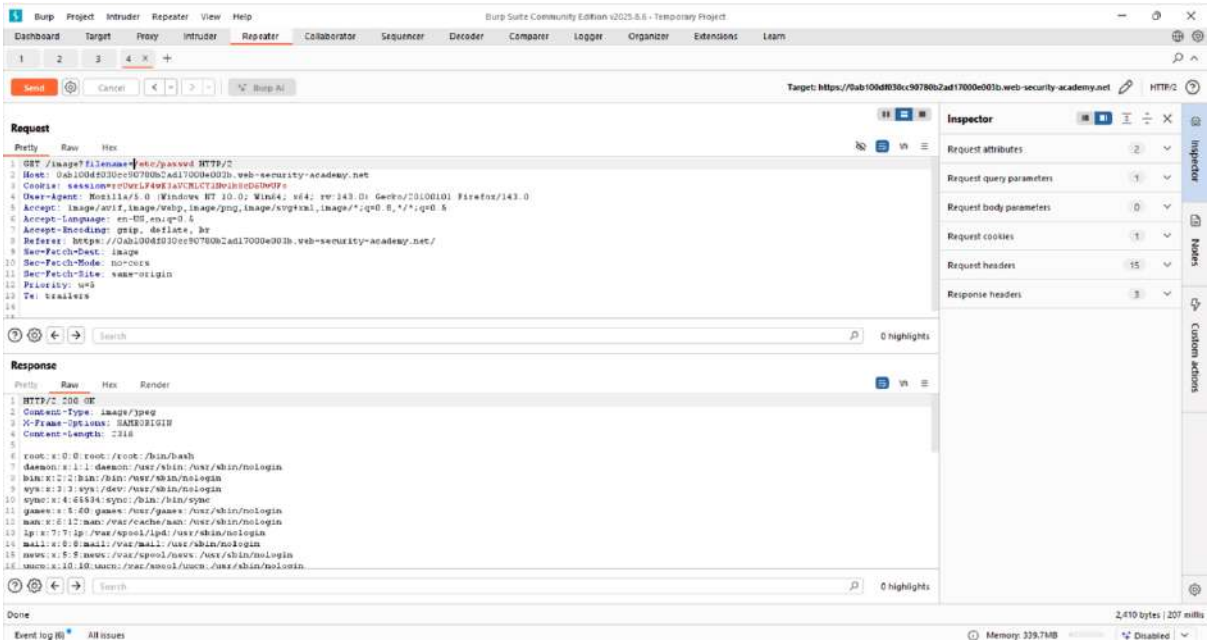
1. Observasi awal menggunakan Burp Suite menunjukkan *request* yang sah ke *endpoint* /image menggunakan nama file yang valid (contoh: filename=1.jpg).



2. Percobaan pertama dilakukan menggunakan *payload traversal* klasik: ../../../etc/passwd. *Request* ini gagal dan server merespons dengan 400 Bad Request serta pesan "No such file", yang mengindikasikan bahwa sekuens ../ diblokir oleh filter.

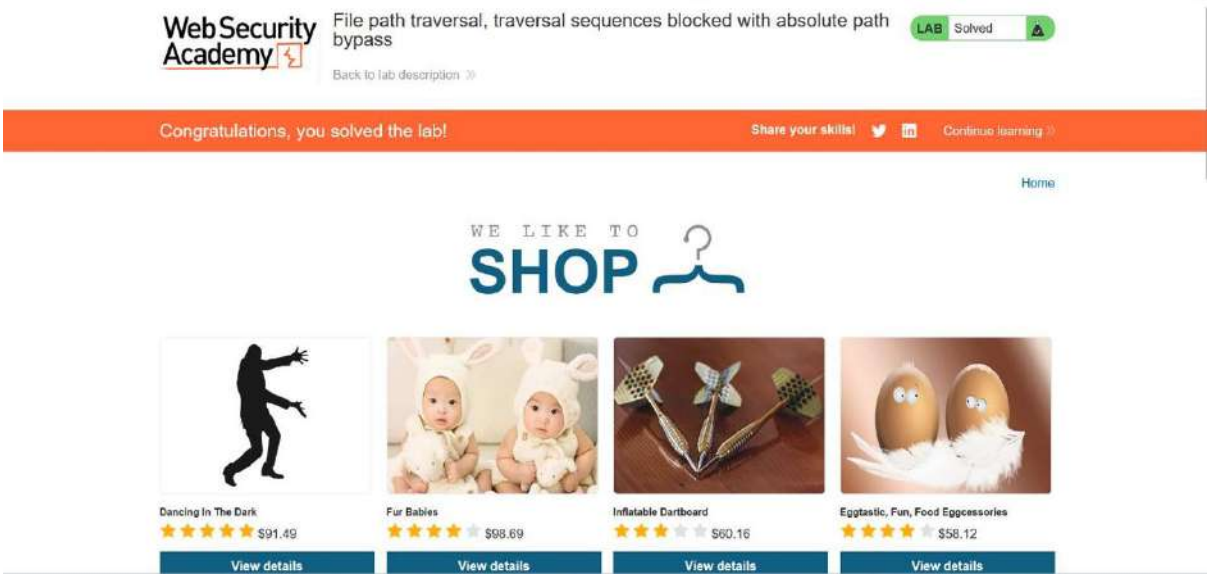


3. Percobaan kedua dilakukan dengan mengubah strategi, yaitu menggunakan *payload absolute path*: /etc/passwd.



4. *Request* dengan *payload* /etc/passwd ini berhasil. Server merespons dengan HTTP 200 OK dan mengembalikan isi lengkap dari file /etc/passwd di dalam *response body*.

5. Keberhasilan eksploitasi ini secara otomatis menyelesaikan lab.



Mitigasi untuk kerentanan ini memerlukan pendekatan yang lebih kuat daripada sekadar memblokir ../:

1. Validasi Path Input: Validasi input dari pengguna untuk memastikan bahwa path tersebut *tidak* dimulai dengan / atau \ (mencegah *absolute path*).

2. Gunakan Path Direktori Dasar: Di sisi *server*, *selalu* gabungkan input pengguna yang *telah divalidasi* dengan path direktori dasar yang sudah di-*hardcode* (contoh: /var/www/images/). Jangan pernah menggunakan input pengguna sebagai path file yang lengkap.

3. Implementasikan *Allow-List*: Cara terbaik adalah dengan hanya mengizinkan nama file yang sesuai dengan pola yang aman (misalnya, karakter alfanumerik, tanda hubung, dan ekstensi .jpg/.png) dan menolak semua yang lain.

2.6.3 File path traversal, traversal sequences stripped non-recursively

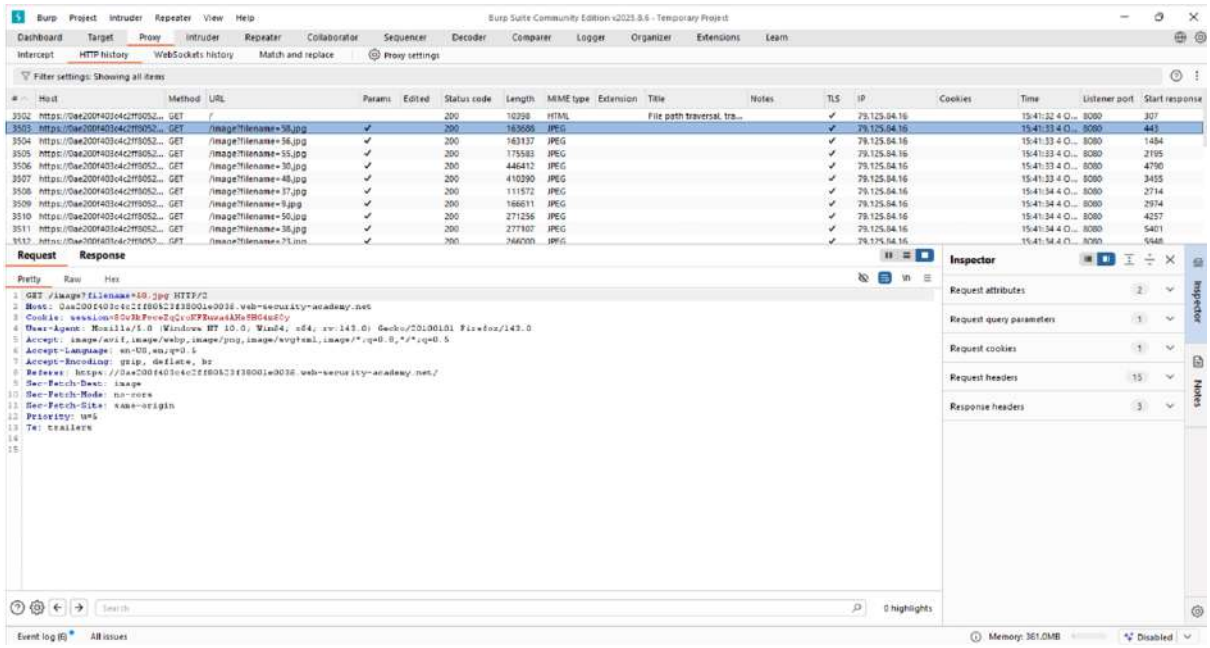
Aplikasi web ini mencoba mencegah serangan *Path Traversal* dengan menerapkan filter yang memblokir sekuens *traversal* standar (../). Namun, filter ini tidak memadai karena hanya mencari *string* literal ../ dan gagal memblokir sekuens alternatif yang memiliki fungsi setara. Penyerang dapat memanfaatkan kelemahan ini dengan menggunakan *payload* seperti ....//, yang diabaikan oleh filter aplikasi namun ditafsirkan oleh *filesystem* server sebagai ../ (setara dengan ../../). Hal ini memungkinkan penyerang untuk tetap menavigasi ke direktori induk dan membaca file-file sensitif.

Url Affected/Endpoint	https://[LAB_ID].web-security-academy.net/image?filename=[INJECTION_POINT]
Severity	**High**
CWE	CWE-23: *Relative Path Traversal*
OWASP	A01:2021-Broken Access Control
CVSS Score	7.5
CVSS String	CVSS:3.1/AVN/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

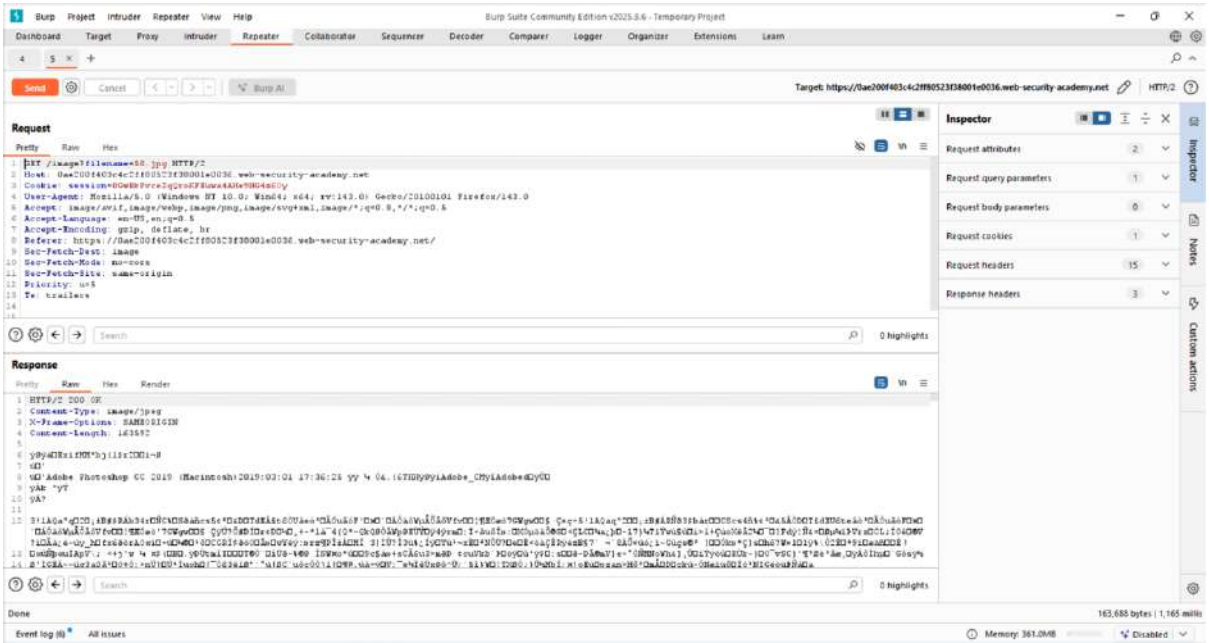
Meskipun ada filter, dampaknya tetap sama: penyerang dapat membaca file-file sistem yang sensitif di luar direktori web yang seharusnya, yang dapat mengarah pada kebocoran kode sumber, file konfigurasi, atau kredensial.

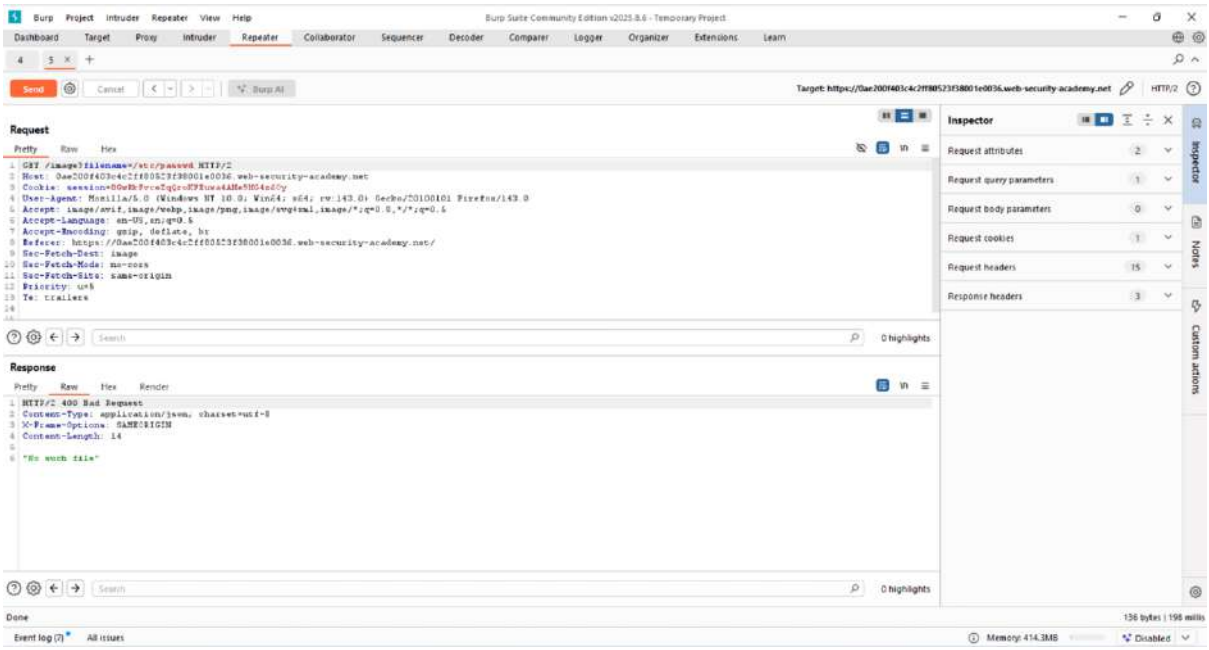
Step to Reproduce with POC (Proof of Concept)

1. Observasi awal pada Burp Suite Repeater menunjukkan *request* yang sah (contoh: filename=58.jpg) mengembalikan data gambar.

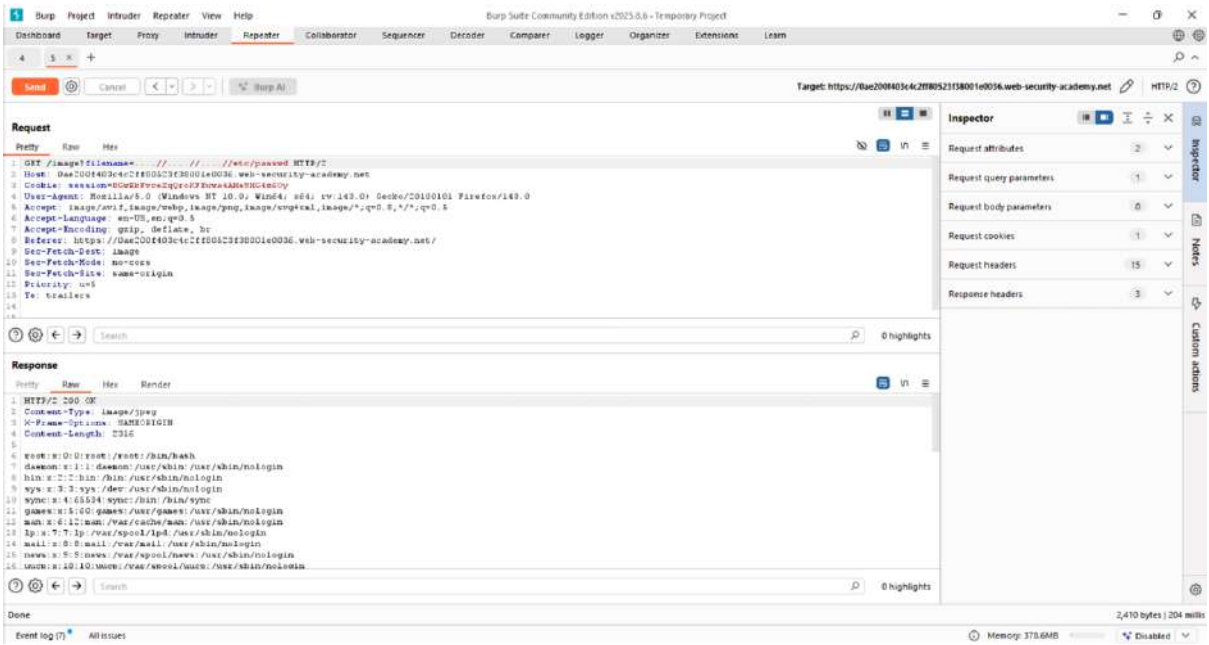


2. Percobaan pertama dilakukan menggunakan *payload traversal* absolut: /etc/passwd. *Request* ini gagal dan server merespons dengan 400 Bad Request, yang mengonfirmasi adanya filter yang memblokir.



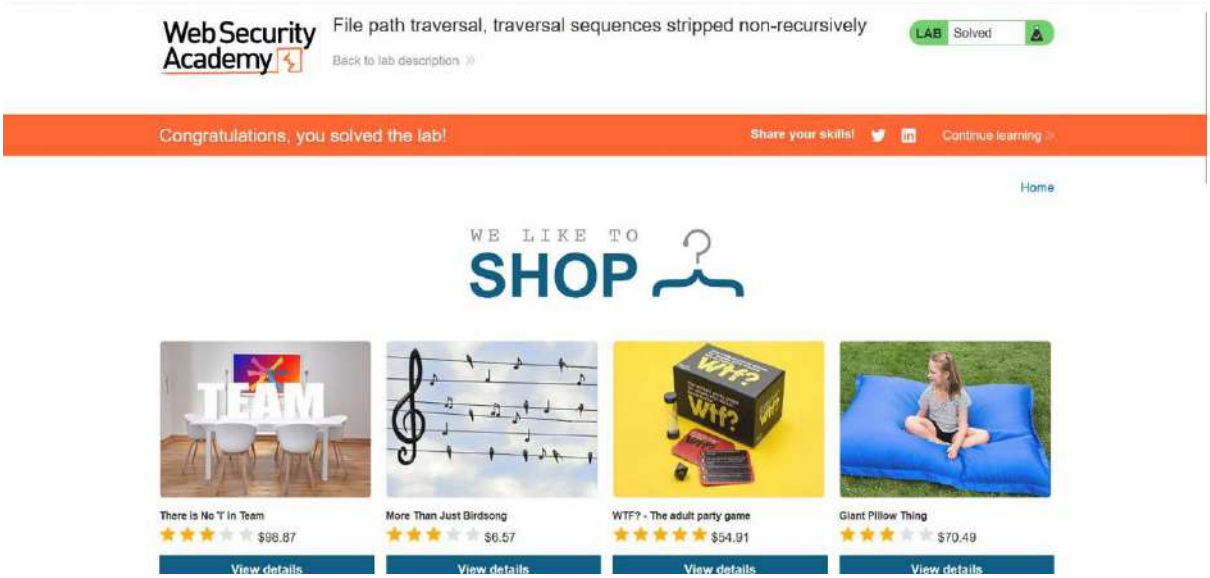


3. Percobaan kedua dilakukan dengan strategi *filter bypass*. *Payload* diubah menjadi ....//....//....//etc/passwd. *Payload* ini menggunakan sekuens ....// yang akan diinterpretasikan oleh *filesystem* sebagai ../ tetapi tidak terdeteksi oleh filter aplikasi.



4. *Request* dengan *payload* alternatif ini berhasil. Server merespons dengan HTTP 200 OK dan mengembalikan isi lengkap dari file /etc/passwd.

5. Keberhasilan eksploitasi ini secara otomatis menyelesaikan lab.



Mitigasi untuk kerentanan ini memerlukan filter yang lebih kuat:

1. Normalisasi Path: Sebelum melakukan validasi, normalisasikan input path dari pengguna. Ubah semua enkode alternatif (seperti ....// atau .././) menjadi bentuk kanonikalnya (../). Setelah dinormalisasi, barulah terapkan filter untuk memblokir sekuens *traversal*.
2. Validasi *Allow-List*: Seperti sebelumnya, cara terbaik adalah dengan hanya mengizinkan nama file yang sesuai dengan pola yang sangat ketat (misalnya, hanya alfanumerik dan satu titik) dan menolak semua yang lain.
3. Gunakan Path Absolut dari Sisi Server: Jangan pernah menggabungkan input dari pengguna secara langsung. Gunakan path direktori dasar yang aman di *backend* dan pastikan path yang dihasilkan tetap berada di dalam direktori tersebut.

2.6.4 File path traversal, traversal sequences stripped with superfluous URL-decode

Aplikasi web ini menerapkan mekanisme filter yang cacat untuk mencegah *Path Traversal*. Logika aplikasi bekerja sebagai berikut:

1. Aplikasi menerima input dari parameter filename.
2. Aplikasi melakukan **satu kali URL-decode** pada input tersebut.
3. Aplikasi memindai dan menghapus sekuens ../ dari input yang telah di-decode.
4. Aplikasi meneruskan hasil yang "bersih" tersebut ke API *filesystem* untuk mengambil file.

Kelemahannya terletak pada proses *decode* yang berlebihan (*superfluous*). Penyerang dapat memanfaatkan ini dengan melakukan *double encoding* pada sekuens *traversal*. Misalnya, karakter / di-encode menjadi %2f, lalu karakter % di-encode lagi menjadi %25, menghasilkan %252f.

Ketika *payload* ..%252f..%252fetc/passwd dikirim, aplikasi (langkah 2) akan men-decode-nya menjadi ..%2f..%2fetc/passwd. Filter (langkah 3) tidak akan menemukan *string* ../

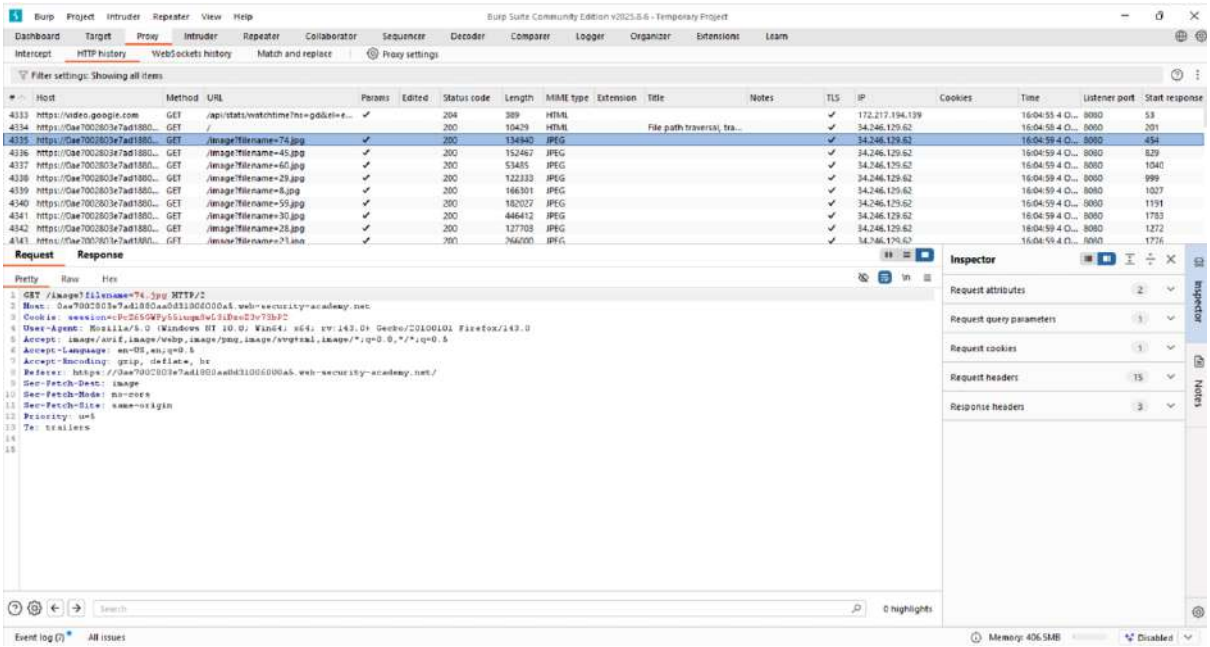
dan melewatkannya. Akhirnya, API *filesystem* (langkah 4) menerima ..%2f..%2fetc/passwd, melakukan *decode* sekali lagi, dan mengeksekusi path ../../../etc/passwd.

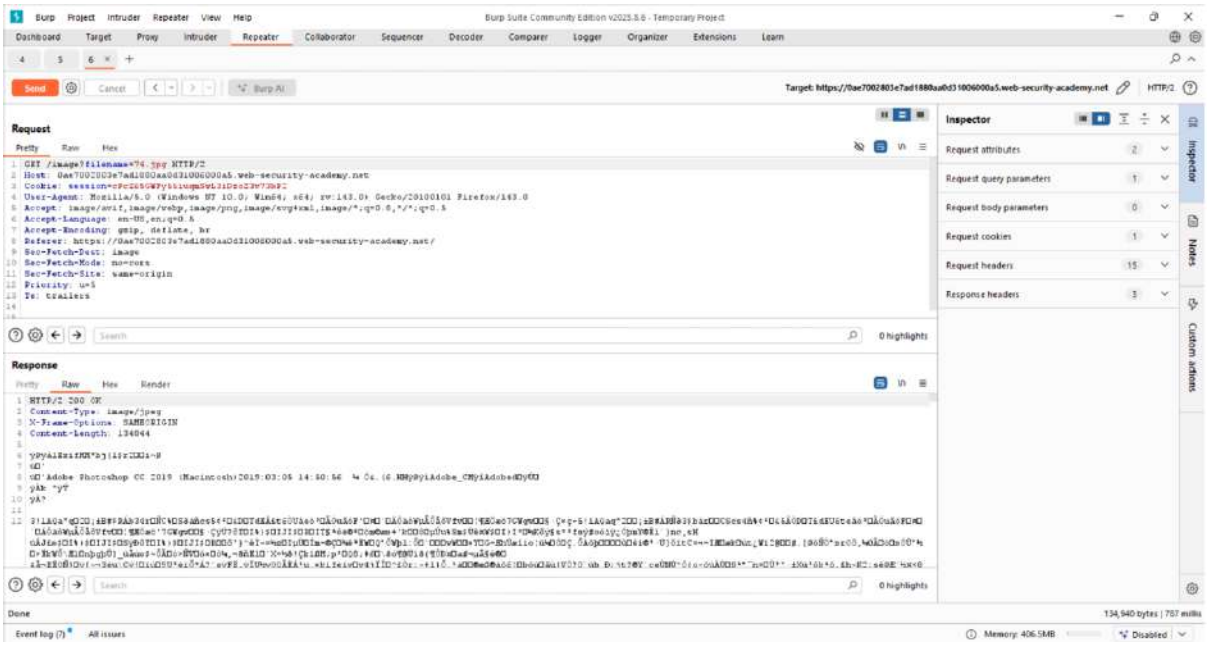
Url Affected/Endpoint	https://[LAB_ID].web-security-academy.net/image?filename=[INJECTION_POINT]
Severity	**High**
CWE	CWE-22: *Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')*
OWASP	A01:2021-Broken Access Control
CVSS Score	7.5
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

Meskipun ada filter yang kompleks, dampaknya tetap sama: penyerang dapat membaca file-file sistem yang sensitif, yang dapat mengarah pada kebocoran data, kode sumber, atau kredensial.

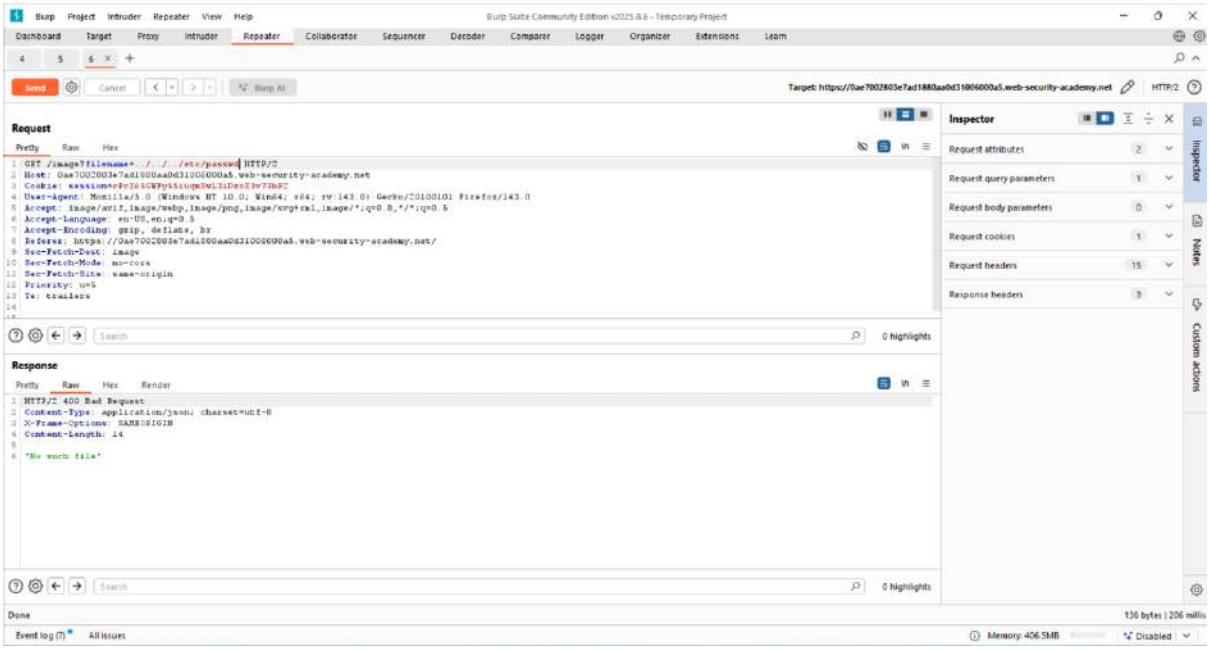
Step to Reproduce with POC (Proof of Concept)

1. Observasi awal pada Burp Suite Repeater menunjukkan *request* yang sah (contoh: filename=74.jpg) mengembalikan data gambar.

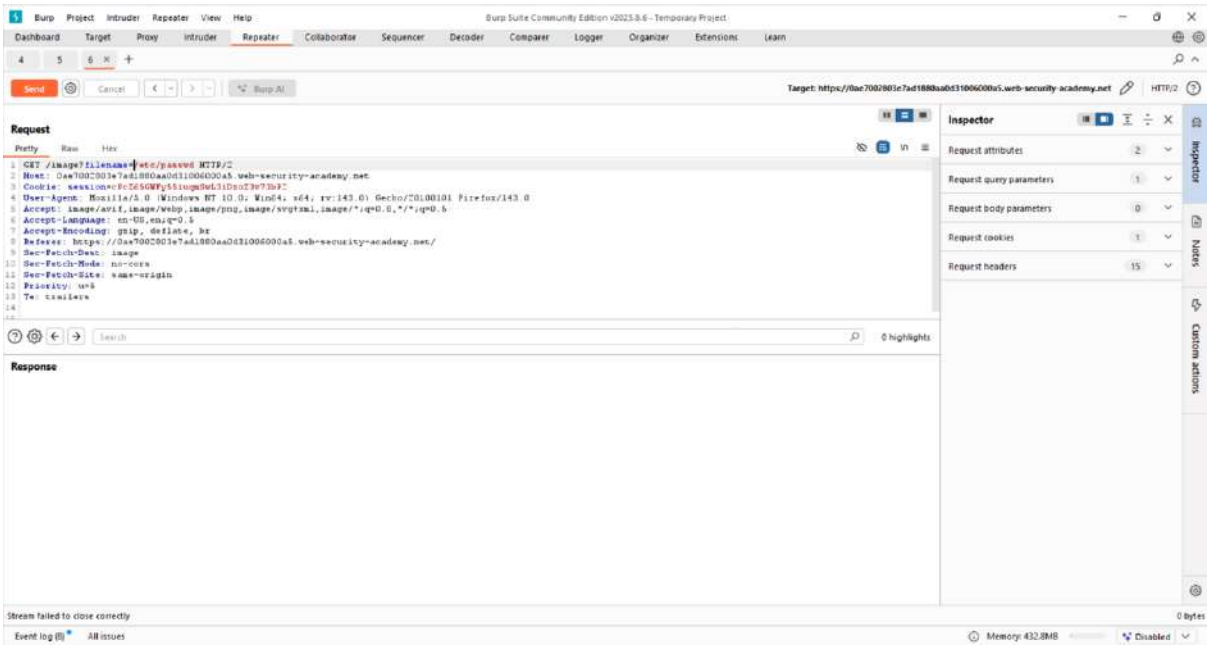


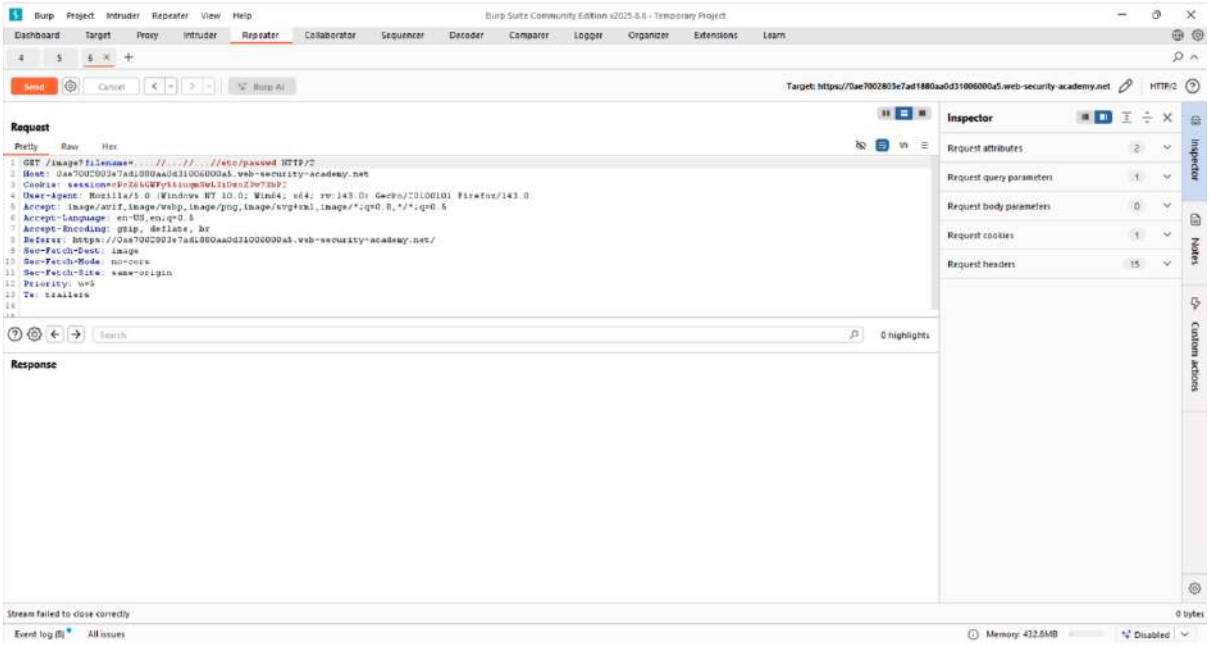


2. Percobaan pertama dilakukan menggunakan *payload traversal* standar: ../../../etc/passwd. *Request* ini gagal dan server merespons dengan 400 Bad Request ("No such file"), mengonfirmasi adanya filter.

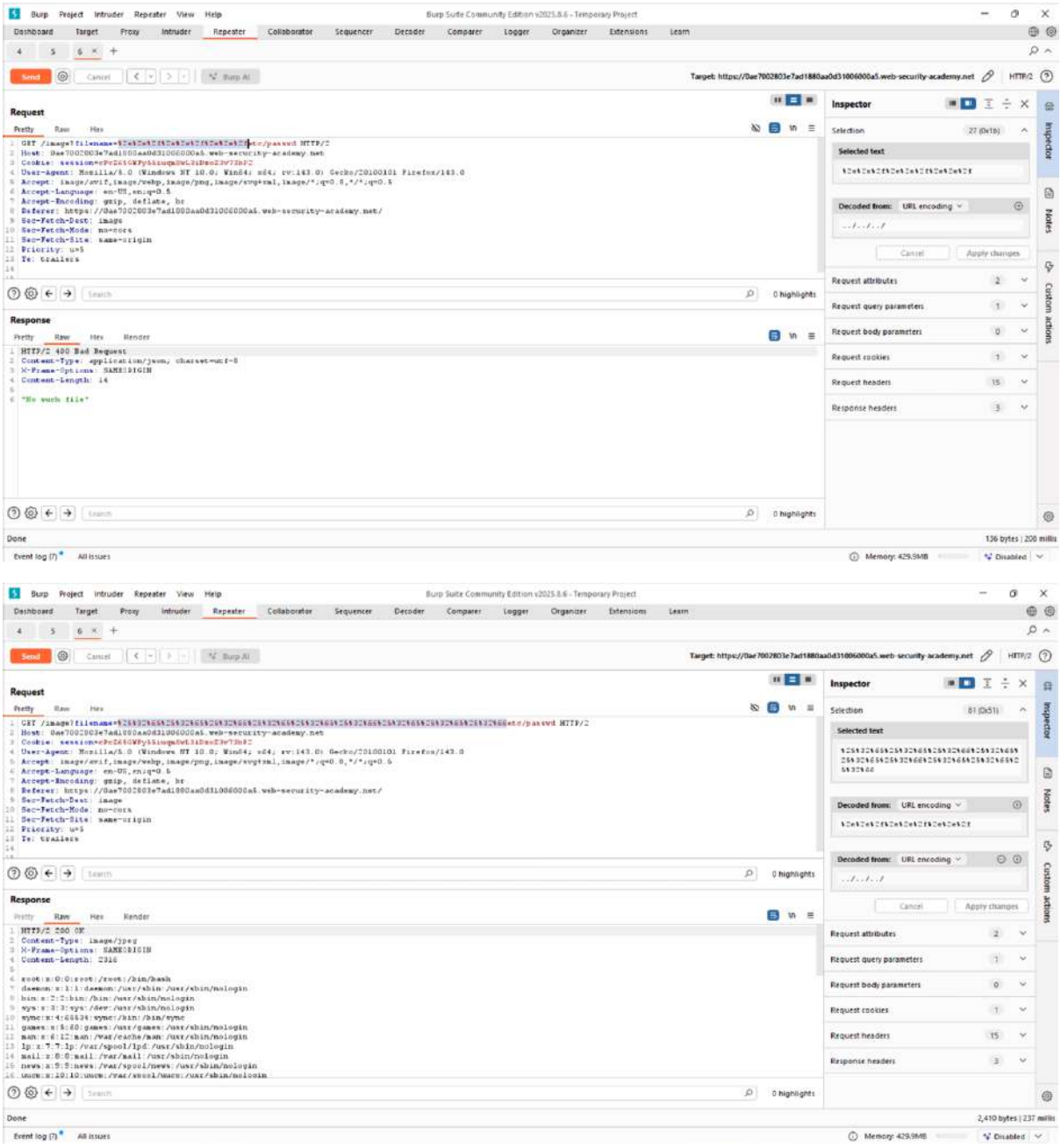


3. Percobaan dengan *bypass* lain seperti ....// dan *absolute path* /etc/passwd juga gagal, menunjukkan filter yang lebih canggih.

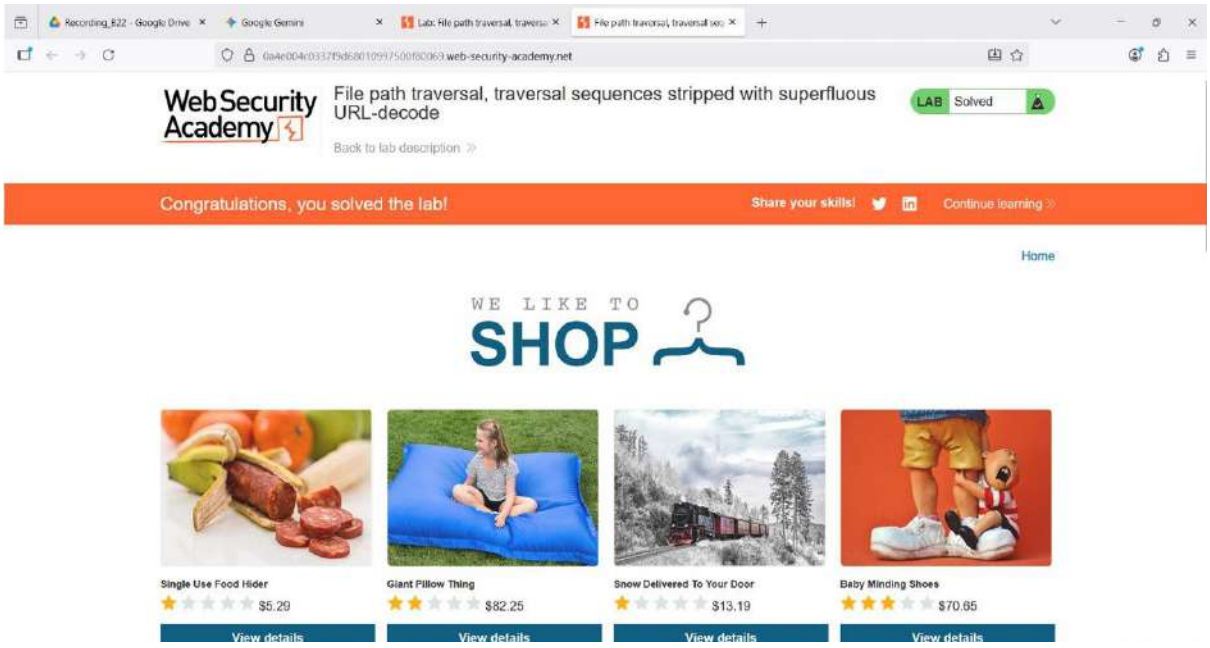




4. Strategi diubah untuk mengeksploitasi proses *decode* yang salah. *Payload* yang digunakan adalah *double-encoded traversal*. Karakter / di-encode menjadi %2f, dan karakter % di-encode menjadi %25. *Payload* akhirnya adalah: ..%252f..%252f..%252fetc/passwd.



5. *Request* dengan *payload double-encoded* ini berhasil mengelabui filter. Server merespons dengan HTTP 200 OK dan mengembalikan isi lengkap dari file /etc/passwd.
6. Keberhasilan eksploitasi ini secara otomatis menyelesaikan lab.



Mitigasi untuk kerentanan ini memerlukan logika validasi yang sangat ketat:

1. Normalisasi Penuh (Full Normalization): Sebelum melakukan validasi keamanan, input harus dinormalisasi sepenuhnya. Lakukan URL-decode secara rekursif hingga tidak ada lagi karakter yang ter-encode. *Setelah* itu, barulah terapkan filter untuk memblokir sekuens *traversal*.
2. Validasi *Allow-List*: Cara teraman adalah dengan menolak semua input yang tidak sesuai dengan *allow-list* yang ketat (misalnya, hanya izinkan nama file yang terdiri dari alfanumerik, tanda hubung, dan ekstensi .jpg atau .png).
3. Gunakan Path Direktori Dasar: Jangan gunakan input pengguna sebagai path lengkap. Selalu gunakan path direktori dasar yang aman di *backend* dan gabungkan dengan input pengguna yang *sudah divalidasi*.

2.6.5 File path traversal, validation of start of path

Aplikasi web ini menerapkan mekanisme pertahanan yang tidak memadai terhadap *Path Traversal*. Aplikasi memvalidasi bahwa input pada parameter filename harus dimulai dengan path yang diharapkan, yaitu /var/www/images/. Namun, validasi ini hanya terjadi pada awal *string* dan aplikasi gagal untuk memvalidasi atau membersihkan sekuens *traversal* (../) yang muncul *setelah* path yang divalidasi tersebut. Penyerang dapat dengan mudah melewati filter ini dengan membuat *payload* yang mematuhi aturan "starts-with" tetapi kemudian keluar dari direktori tersebut untuk membaca file sistem yang sensitif.

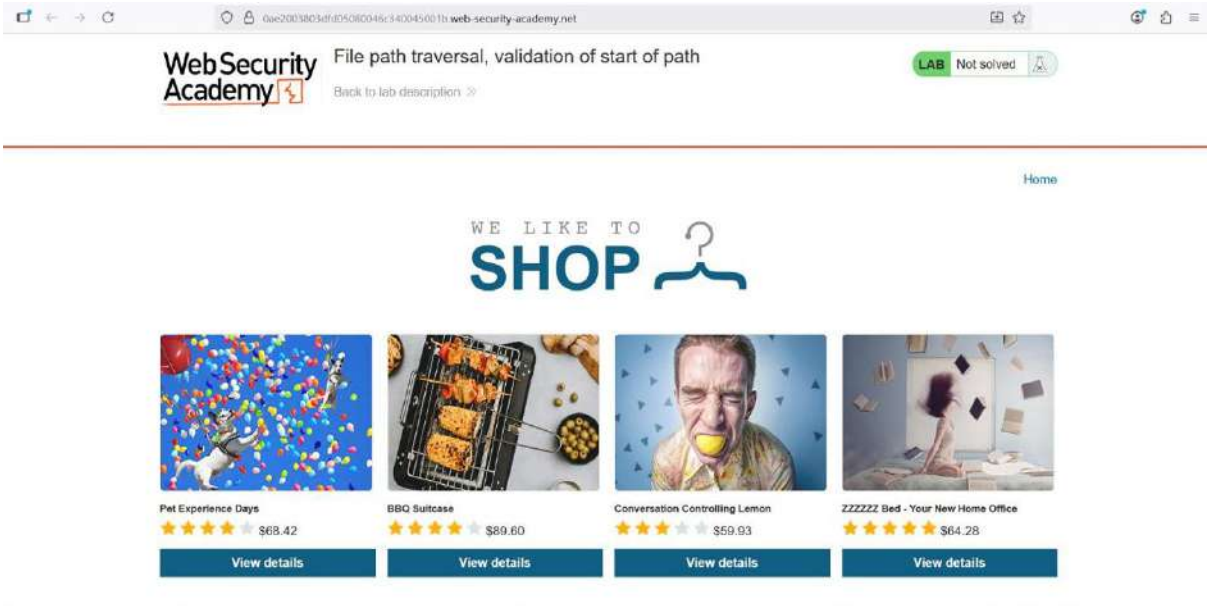
Url Affected/Endpoint	https://[LAB_ID].web-security-academy.net/image?filename=[INJECTION_POINT]
Severity	**High**

CWE	CWE-28: *Improper Validation of Input "Starts-With" Check* (Variasi dari CWE-22 Path Traversal)
OWASP	A01:2021-Broken Access Control
CVSS Score	7.5
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

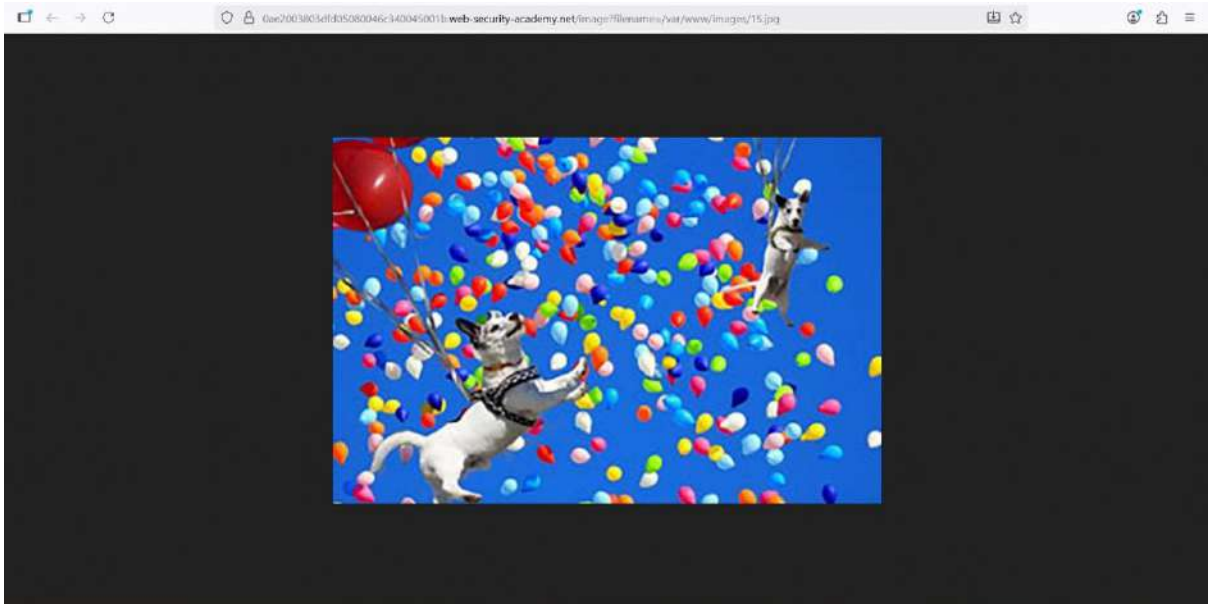
Meskipun ada upaya mitigasi, dampaknya tetap sama dengan serangan *Path Traversal* lainnya. Penyerang dapat membaca file-file arbitrer di server, yang dapat mengarah pada kebocoran file konfigurasi, *credential*, kode sumber, atau informasi pengguna sistem.

Step to Reproduce with POC (Proof of Concept)

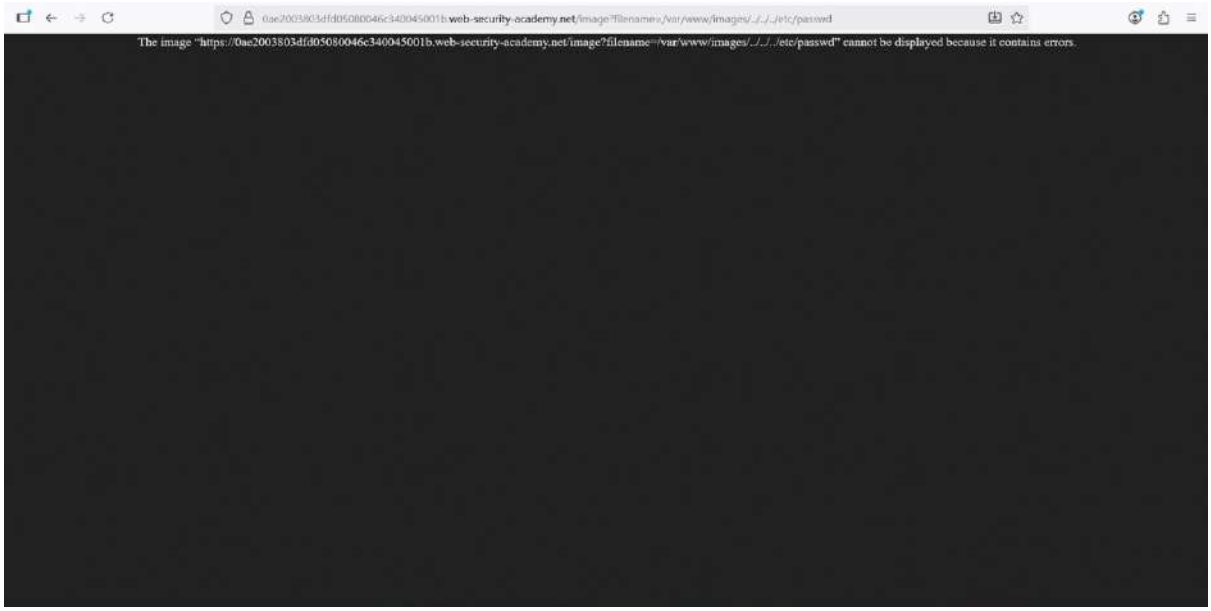
1. Buka halaman lab. Terlihat sebuah situs e-commerce.



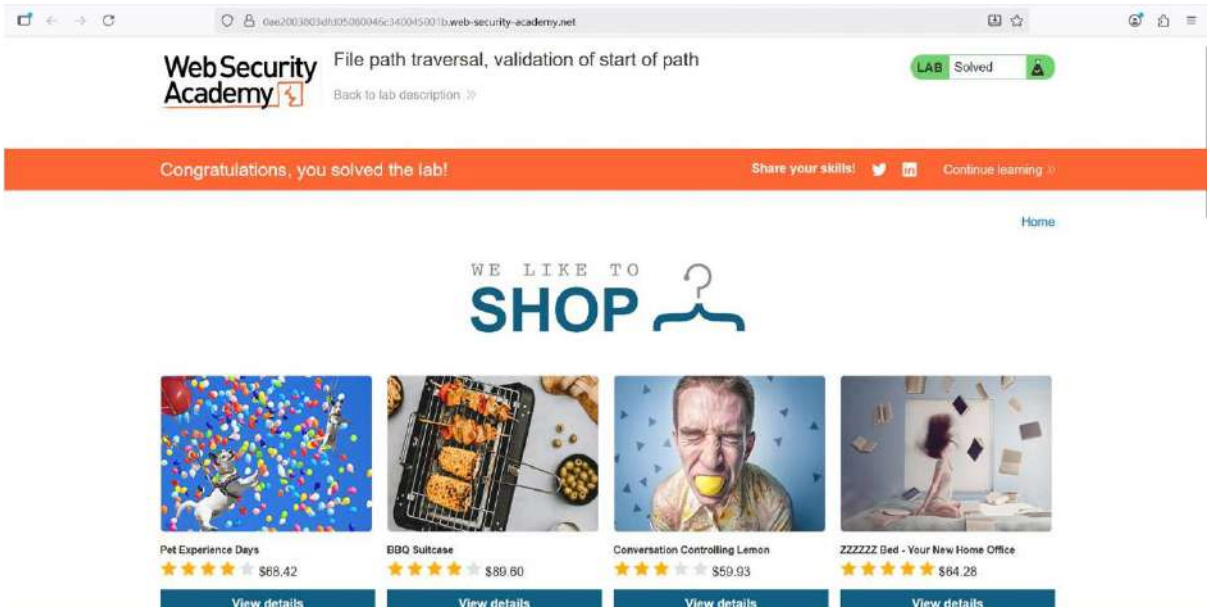
2. Saat melihat gambar, URL di *address bar* menunjukkan bahwa aplikasi menggunakan *absolute path* untuk mengambil file. Contoh: .../image?filename=/var/www/images/15.jpg. Ini mengindikasikan bahwa aplikasi kemungkinan memvalidasi awal dari path tersebut.



3. Untuk menguji *bypass*, *payload* dibuat agar tetap mematuhi aturan "starts-with" /var/www/images/, tetapi kemudian menambahkan sekuens *traversal* di belakangnya.
4. *Payload* yang digunakan adalah: /var/www/images/../../../etc/passwd.



5. Saat *payload* ini dikirim, server berhasil mengambil file /etc/passwd. Browser mencoba menampilkannya sebagai gambar, yang mengakibatkan pesan *error* "cannot be displayed because it contains errors". Ini mengonfirmasi bahwa file non-gambar (yaitu /etc/passwd) telah berhasil dibaca.
6. Keberhasilan eksploitasi ini secara otomatis menyelesaikan lab.



Mitigasi untuk kerentanan ini harus lebih kuat daripada sekadar pemeriksaan "starts-with":

1. Normalisasi dan Validasi Path Kanonikal: Setelah menggabungkan input pengguna dengan direktori dasar, aplikasi harus menormalisasi path tersebut (misalnya, mengubah /var/www/images/../../../etc/passwd menjadi path kanonikal-nya, yaitu /etc/passwd).
2. Validasi Path Final: Setelah normalisasi, aplikasi harus memvalidasi *sekali lagi* bahwa path yang sudah dinormalisasi tersebut masih berada di dalam direktori yang diizinkan (yaitu, masih dimulai dengan /var/www/images/). Dalam kasus ini, /etc/passwd akan gagal dalam pemeriksaan dan *request* akan diblokir.
3. Gunakan *Allow-List*: Cara teraman adalah dengan tidak menerima path dari pengguna, melainkan hanya nama file (misalnya 15.jpg). Validasi bahwa nama file hanya berisi karakter alfanumerik dan ekstensi yang diizinkan, lalu gabungkan dengan path direktori dasar yang aman di *backend*.

2.6.6 File path traversal, validation of file extension with null byte bypass

Aplikasi web ini mencoba mencegah *Path Traversal* dengan memberlakukan validasi ekstensi file. Aplikasi ini mengharuskan nilai dari parameter filename diakhiri dengan ekstensi yang diharapkan (misalnya, .jpg). Namun, aplikasi ini rentan terhadap injeksi *null byte* (%00).

Penyerang dapat memanfaatkan ini dengan membuat *payload* yang berisi sekuens *traversal*, file target, dan diakhiri dengan *null byte* yang di-URL-encode (%00) serta ekstensi file yang valid. Contoh: ../../../etc/passwd%00.jpg

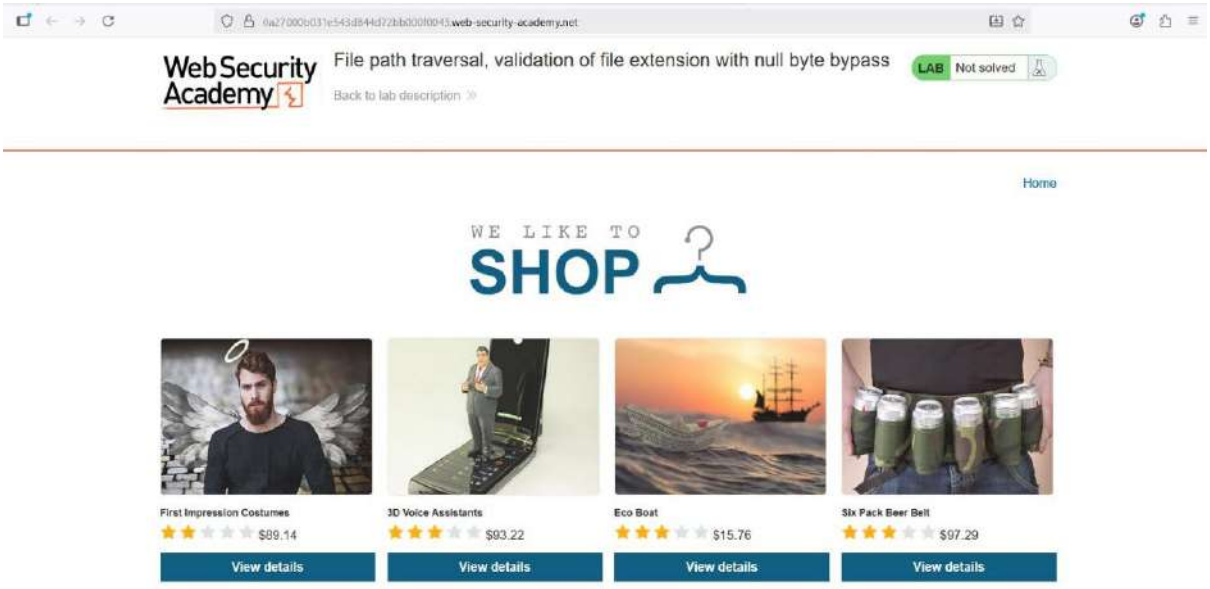
Filter keamanan aplikasi akan membaca seluruh *string* dan memvalidasinya karena diakhiri dengan .jpg. Namun, ketika *string* ini diteruskan ke *filesystem* di *backend* (seringkali berbasis C), *null byte* ditafsirkan sebagai terminator *string*. Akibatnya, *filesystem* hanya memproses bagian ../../../etc/passwd dan mengabaikan sisa *string* (.jpg), sehingga berhasil membaca file sistem yang sensitif.

Url Affected/Endpoint	https://[LAB_ID].web-security-academy.net/image?filename=[INJECTION_POINT]
Severity	**High**
CWE	CWE-22: *Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')* (Secara spesifik memanfaatkan CWE-158: *Improper Handling of Null Byte*)
OWASP	A01:2021-Broken Access Control
CVSS Score	7.5
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

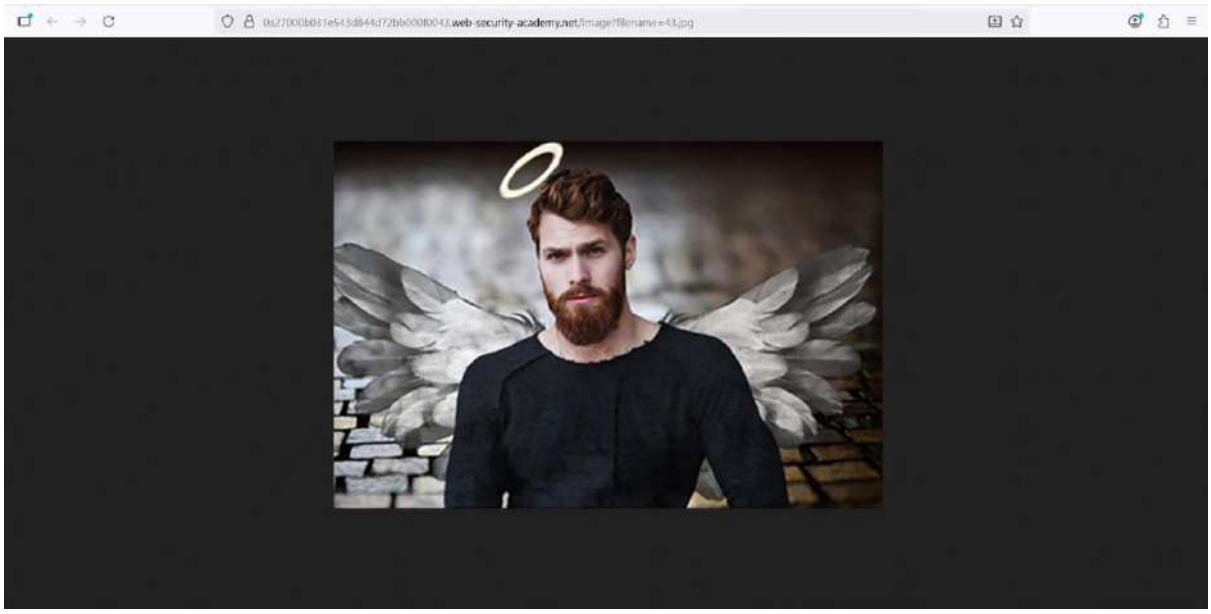
Meskipun ada validasi ekstensi, dampaknya tetap sama: penyerang dapat membaca file-file arbitrer di server, yang dapat mengarah pada kebocoran file konfigurasi, *credential*, kode sumber, atau informasi pengguna sistem.

Step to Reproduce with POC (Proof of Concept)

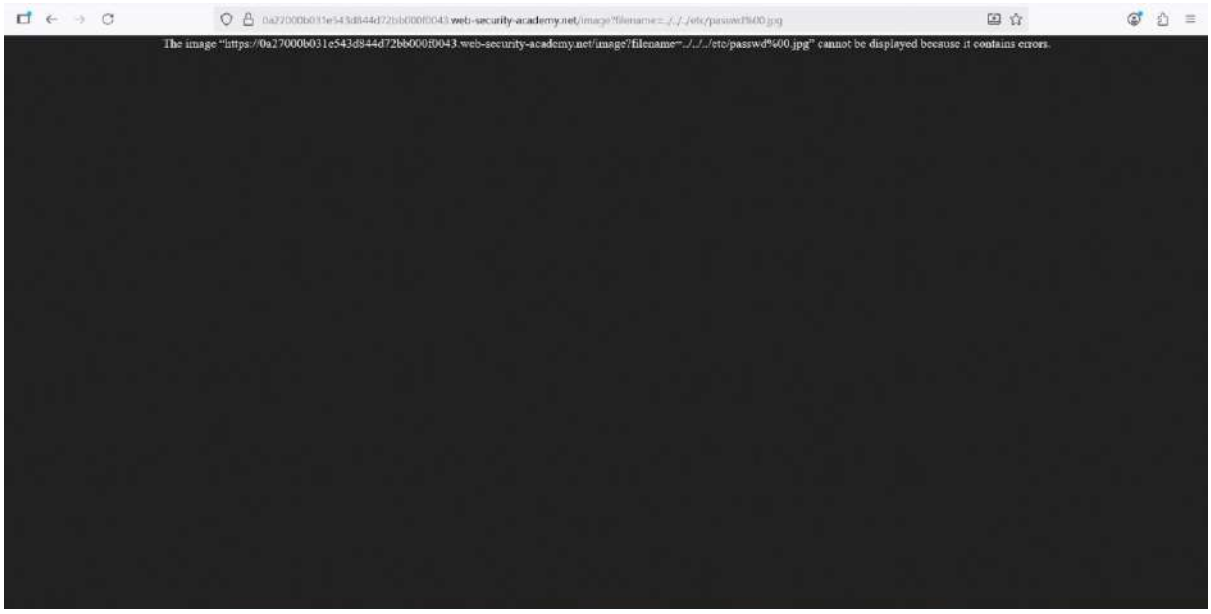
1. Buka halaman lab. Terlihat sebuah situs e-commerce.



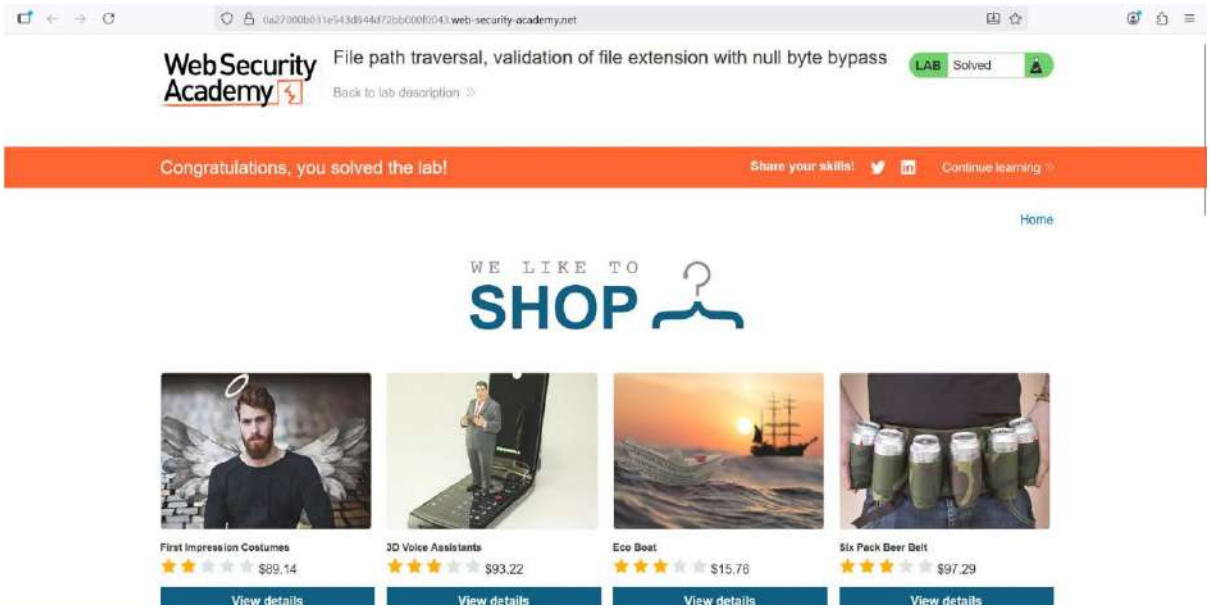
2. Observasi pada URL gambar (contoh: .../image?filename=43.jpg) menunjukkan bahwa aplikasi sepertinya mengambil file berdasarkan nama dan ekstensinya.



3. Percobaan *traversal* biasa (misalnya ../../../etc/passwd) akan gagal karena tidak diakhiri dengan .jpg.
4. Untuk melewati filter, *payload* yang dimanipulasi dengan *null byte* digunakan: ../../../etc/passwd%00.jpg.



5. Saat *payload* ini dikirim, filter ekstensi berhasil dilewati. *Filesystem* memproses path hingga ke *null byte* dan membaca file /etc/passwd. Browser mencoba merender file ini sebagai gambar, yang mengakibatkan *error* "cannot be displayed because it contains errors". Ini mengonfirmasi bahwa file non-gambar telah berhasil dibaca.
6. Keberhasilan eksploitasi ini secara otomatis menyelesaikan lab.



Untuk mencegah serangan *null byte* dan *Path Traversal*:

1. Tolak Input dengan *Null Byte*: Validasi input aplikasi harus secara eksplisit menolak *request* yang mengandung karakter *null byte* (%00 atau \x00).
2. Validasi Setelah Normalisasi Path: Lakukan validasi keamanan *setelah* path dinormalisasi dan periksa apakah ada karakter ilegal.
3. Gunakan *Allow-List*: Cara teraman adalah dengan memvalidasi input agar hanya berisi karakter alfanumerik dan satu titik untuk ekstensi, lalu gabungkan dengan path direktori dasar yang aman di *backend*.
4. Periksa Panjang *String*: Beberapa API *file* modern memungkinkan pemeriksaan panjang *string* secara eksplisit, yang tidak akan terpengaruh oleh *null byte* di tengah *string*.

BAB III

PEMBAHASAN TUGAS OPSIONAL

3.1 .GIT Vuln xsslabs - Exposing .git Directory Leads to Full Source Code Disclosure

Aplikasi web xsslabs memiliki kerentanan miskonfigurasi keamanan yang kritis. Direktori .git, yang digunakan oleh sistem *version control* Git untuk melacak perubahan kode, dibiarkan dapat diakses secara publik melalui web. Hal ini kemungkinan besar terjadi karena kesalahan saat *deployment*, di mana seluruh folder proyek, termasuk sub-direktori .git yang tersembunyi, diunggah ke *web root* server. Seorang penyerang dapat menemukan direktori ini dan menggunakan *tools* khusus untuk mengunduh (*dump*) seluruh isi repositori, yang pada akhirnya memungkinkan mereka untuk merekonstruksi dan mendapatkan kode sumber lengkap dari aplikasi.

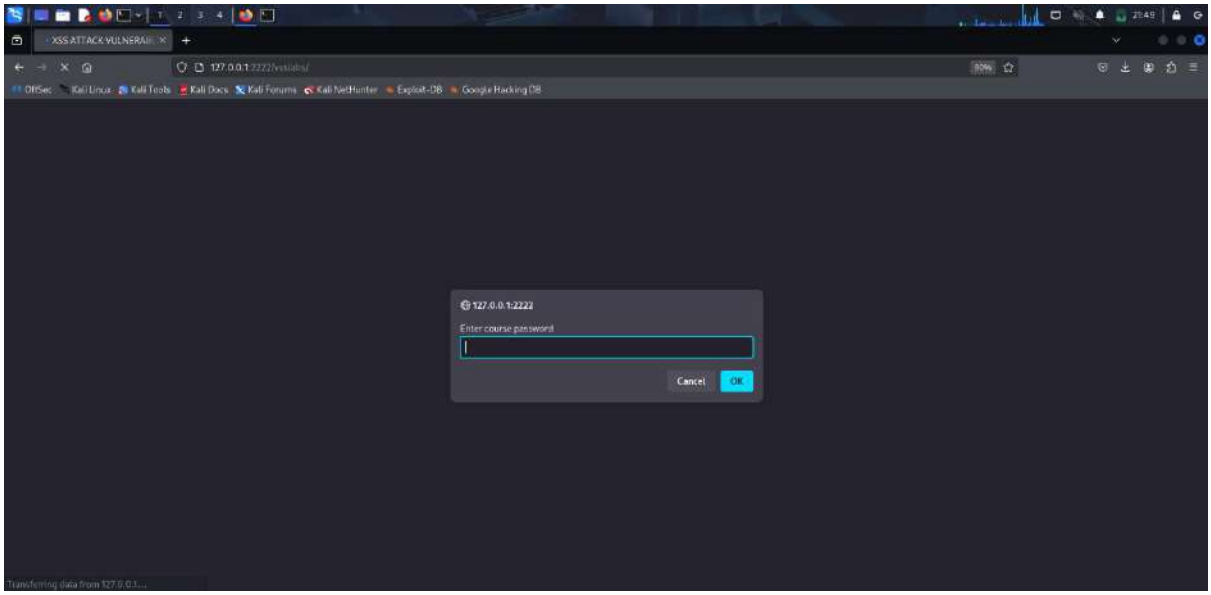
Url Affected/Endpoint	http://127.0.0.1:2222/xsslabs/.git/
Severity	**High**
CWE	CWE-200: *Exposure of Sensitive Information to an Unauthorized Actor*
OWASP	A05:2021-Security Misconfiguration
CVSS Score	7.5
CVSS String	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

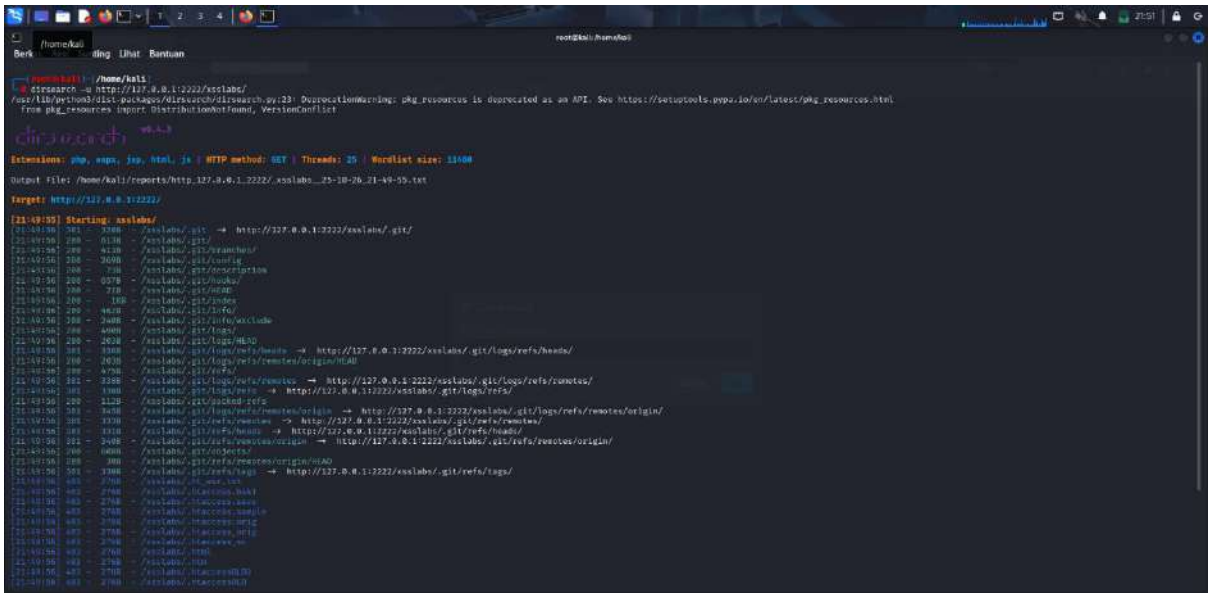
Dampak dari kebocoran kode sumber ini sangat signifikan, meliputi:

1. Penemuan Kerentanan Lain: Penyerang dapat menganalisis kode sumber secara *offline* (disebut juga *white-box testing*) untuk menemukan kerentanan lain yang lebih kompleks, seperti SQL Injection, RCE, atau *business logic flaws* yang sulit ditemukan dengan *black-box scanning*.
2. Kebocoran Kredensial dan API Keys: Kode sumber dan riwayat *commit* sering kali berisi *hardcoded credentials*, seperti password database, *API keys*, atau token layanan eksternal.
3. Pemahaman Infrastruktur: File konfigurasi yang bocor (seperti file .sql yang ditemukan) dapat memberikan informasi berharga tentang infrastruktur *backend*, versi *software*, dan skema database, yang sangat membantu penyerang dalam merencanakan serangan lebih lanjut.

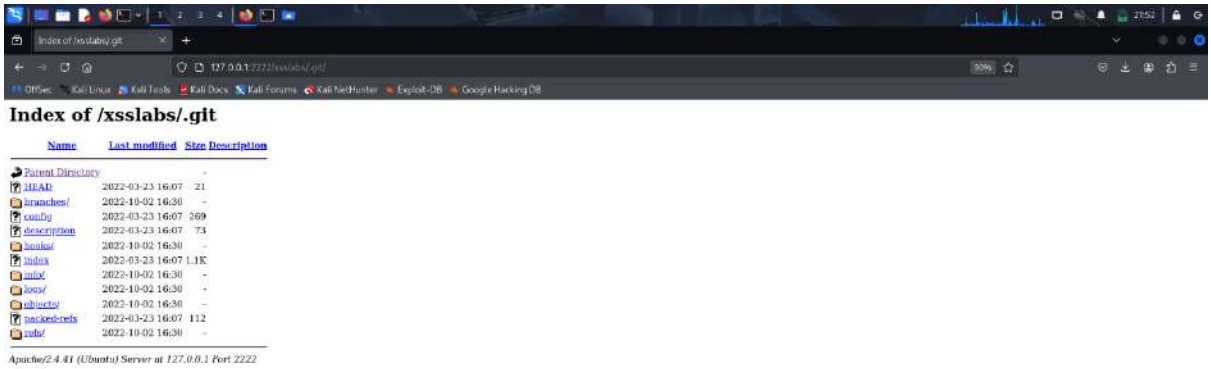
Step to Reproduce with POC (Proof of Concept)

1. Penemuan (Dirsearch): Langkah pertama adalah melakukan *reconnaissance* pada direktori target menggunakan dirsearch untuk menemukan *endpoint* yang tersembunyi. Perintah ini berhasil menemukan direktori .git yang merespons dengan 200 OK.

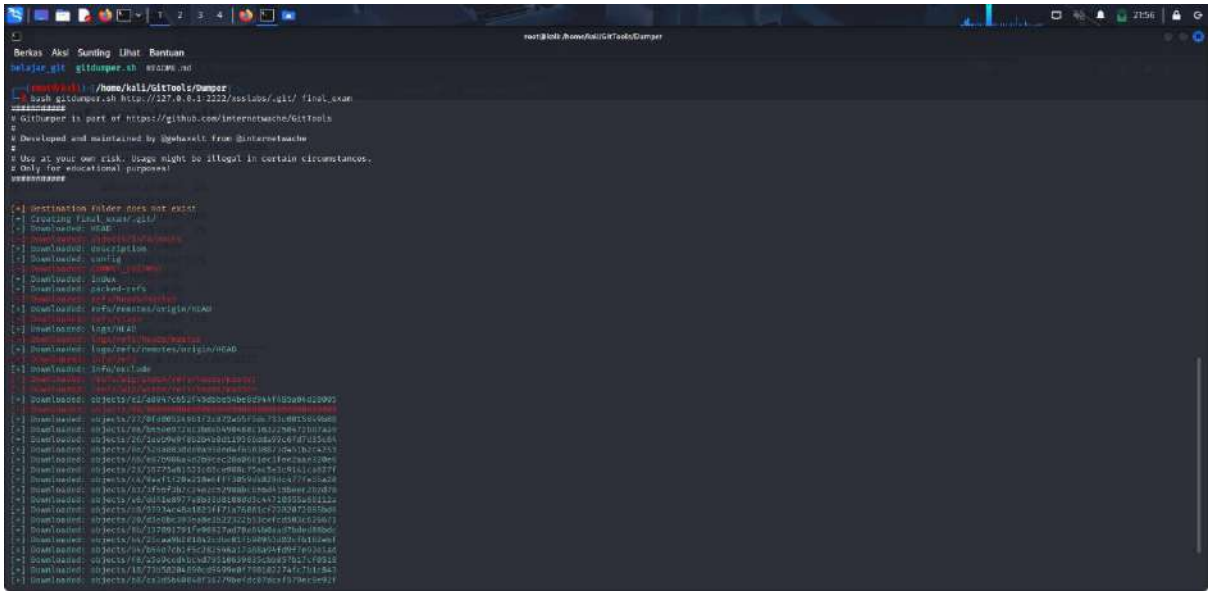




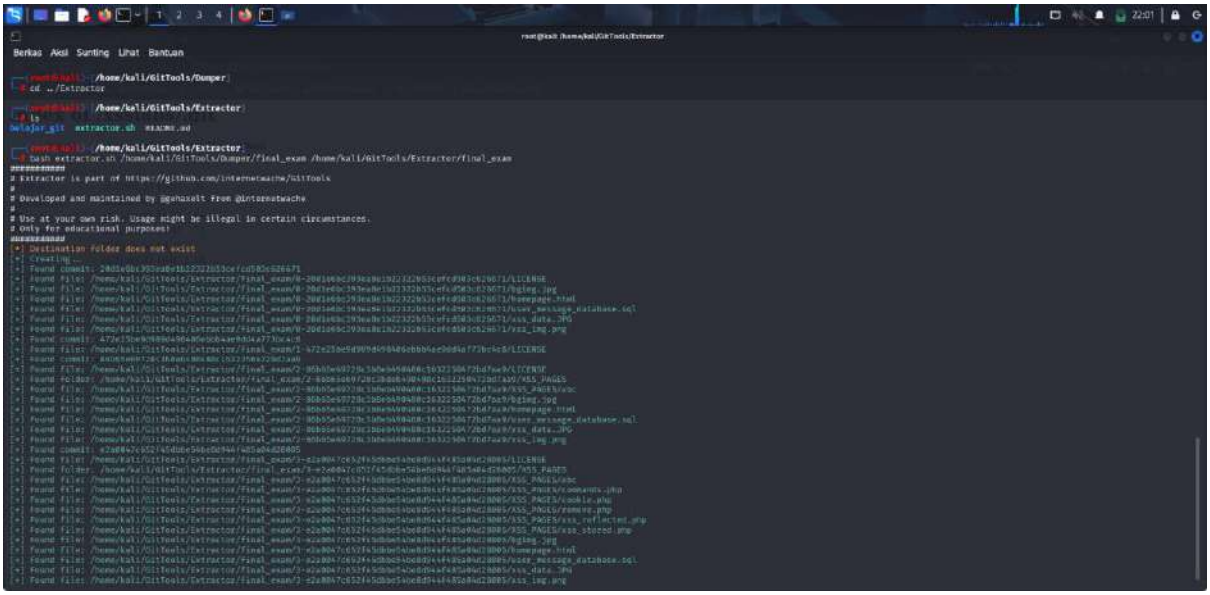
2. Verifikasi Manual: Membuka URL http://127.0.0.1:2222/xsslabs/.git/ di browser mengonfirmasi bahwa *directory listing* aktif, yang memperlihatkan struktur internal repositori Git.



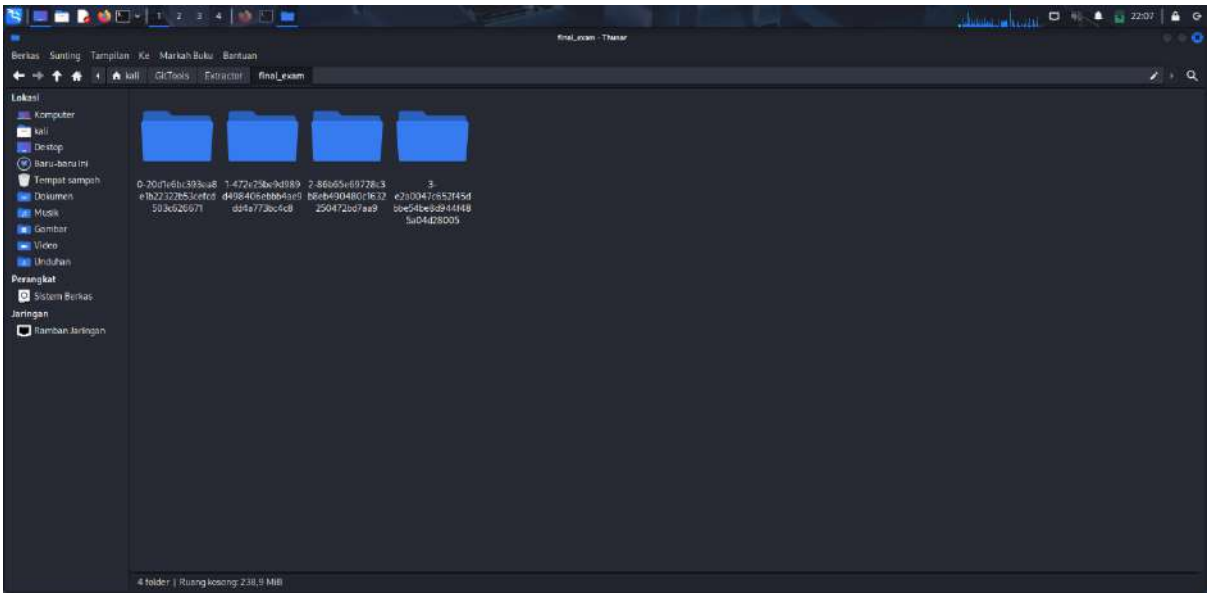
3. Dumping Repositori (GitTools - Dumper): Menggunakan *tool* gitdumper.sh dari GitTools, seluruh objek dari repositori .git yang terekspos diunduh ke direktori lokal (final_exam).



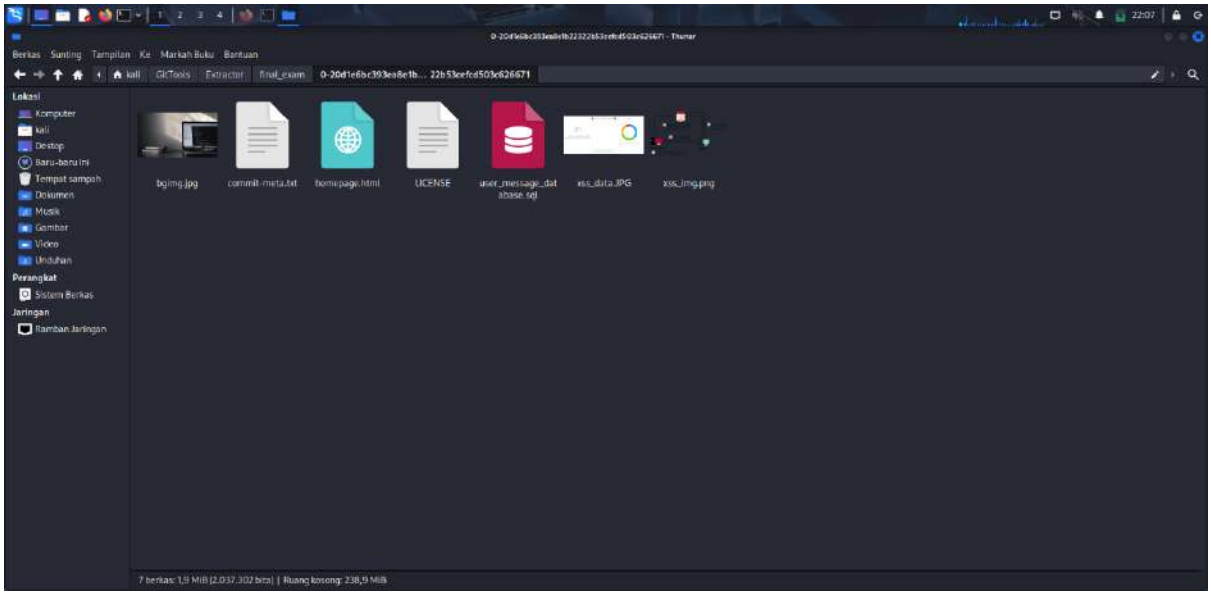
4. Ekstraksi Kode Sumber (GitTools - Extractor): Setelah semua objek diunduh, *tool* extractor.sh dari GitTools digunakan untuk merekonstruksi kode sumber dari objek-objek Git tersebut.

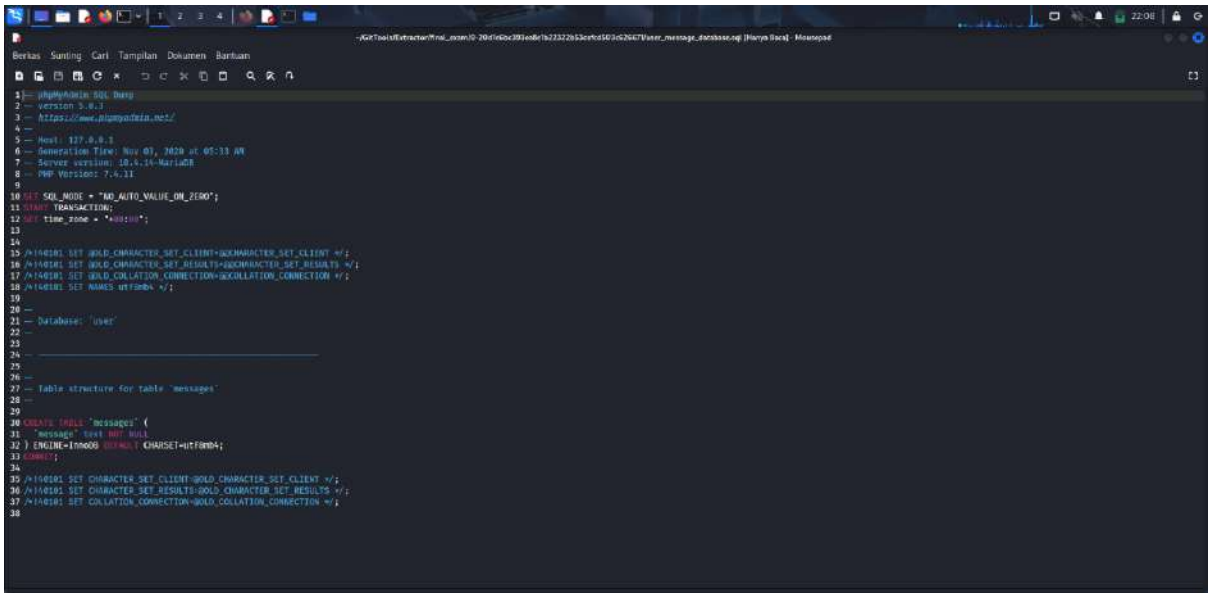


5. Analisis Hasil Ekstraksi: Proses ekstraksi berhasil membuat ulang struktur file proyek, termasuk semua kode sumber, gambar, dan file lainnya.



6. Temuan Informasi Sensitif: Sebagai bukti dampak, salah satu file yang berhasil diekstrak adalah user_message_database.sql. File ini berisi informasi sensitif mengenai skema database, versi server (MariaDB, PHP), dan nama host.





Untuk memperbaiki dan mencegah kerentanan ini, langkah-langkah berikut sangat penting:

1. Blokir Akses ke Direktori .git: Konfigurasikan server web (misalnya melalui .htaccess untuk Apache atau konfigurasi Nginx) untuk secara eksplisit menolak semua permintaan yang mencoba mengakses direktori .git dan mengembalikan respons 403 Forbidden.

2. Praktik Deployment yang Aman: Pastikan bahwa proses *deployment* ke server produksi tidak menyertakan direktori .git. Repositori hanya boleh ada di lingkungan pengembangan dan di *remote repository* (seperti GitHub/GitLab).

3. Audit Kredensial: Segera lakukan audit pada seluruh kode sumber dan riwayat *commit* untuk mencari *hardcoded credentials*. Jika ditemukan, segera rotasi (ganti) semua *keys* dan *password* yang bocor.

4. Gunakan .gitignore: Pastikan file-file sensitif (seperti file .sql, file konfigurasi .env, dll.) selalu dimasukkan ke dalam .gitignore agar tidak pernah ter-commit ke repositori.

BAB 4

KESIMPULAN

4.1 Rangkuman Hasil

Seluruh lab yang ditugaskan telah berhasil diselesaikan dan dieksploitasi. Temuan ini mencakup berbagai kelas kerentanan kritis yang sering ditemukan di aplikasi web modern. Rangkuman dari kerentanan yang berhasil dieksploitasi adalah sebagai berikut:

1. SQL Injection (A03:2021-Injection): Berhasil melakukan eksploitasi SQL Injection untuk *bypass* otentikasi login, baik secara manual maupun otomatis menggunakan SQLMap. Selain itu, berhasil melakukan enumerasi database lengkap untuk mengekstrak kredensial pengguna dari berbagai jenis *database backend* (MySQL, Non-Oracle, dan Oracle), membuktikan adaptabilitas serangan.
2. Broken Access Control (A01:2021-Broken Access Control): Serangkaian kerentanan *Insecure Direct Object Reference* (IDOR) berhasil diidentifikasi. Eksploitasi ini mengarah pada dampak kritis, termasuk:
    - Melihat faktur milik pengguna lain.
    - Mengambil alih akun pengguna lain dengan mengubah *password* mereka.
    - Mencuri dana dengan memanipulasi ID pengirim dalam transaksi transfer uang.
    - Mengalihkan pengiriman pesanan ke alamat pengguna lain.
3. Business Logic Vulnerability (A01:2021-Broken Access Control / A05:2021-Security Misconfiguration): Berhasil memanipulasi parameter di sisi klien untuk membeli tiket dengan harga yang jauh lebih rendah dari yang seharusnya, membuktikan adanya cacat logika bisnis yang dapat menyebabkan kerugian finansial.
4. Path Traversal (A01:2021-Broken Access Control): Berhasil membaca file sensitif server (/etc/passwd) dengan melewati berbagai jenis mekanisme pertahanan, meliputi:
    - *Bypass* filter *traversal sequence* (../).
    - *Bypass* menggunakan *absolute path*.
    - *Bypass* filter non-rekursif (..../).
    - *Bypass* filter menggunakan *double URL encoding* (%252f).
    - *Bypass* validasi "starts-with" path.
    - *Bypass* validasi ekstensi file menggunakan injeksi *null byte* (%00).
5. Security Misconfiguration (A05:2021-Security Misconfiguration): Pada tugas opsional, direktori .git yang terekspos berhasil diidentifikasi menggunakan dirsearch. Seluruh kode sumber aplikasi berhasil diunduh dan direkonstruksi menggunakan GitTools, yang mengarah pada penemuan informasi sensitif (skema database).

4.2. Pembelajaran dan Pengalaman

Pengerjaan tugas akhir ini memberikan pembelajaran dan pengalaman praktis yang sangat berharga:

1. Pentingnya Reconnaissance: Tugas opsional .git membuktikan bahwa fase *reconnaissance* menggunakan *tool* seperti dirsearch adalah krusial untuk menemukan titik masuk yang sering terlewatkan.
2. Kreativitas dalam Melewati Filter: Serangkaian lab *Path Traversal* dan SQL Injection mengajarkan bahwa sebuah aplikasi mungkin memiliki lapisan pertahanan, namun filter tersebut seringkali dapat dilewati dengan pemikiran *out-of-the-box* dan pemahaman mendalam tentang bagaimana *server* memproses data (misalnya, *double encoding*, *null byte*, sintaks dual Oracle).
3. Dampak Kritis dari Kerentanan Logika: Lab IDOR dan *Business Logic* menunjukkan bahwa kerentanan yang paling merusak tidak selalu yang paling teknis (seperti RCE), tetapi bisa berupa cacat logika sederhana yang berdampak langsung pada finansial atau privasi pengguna.
4. Penguasaan Tools: Mendapatkan pengalaman praktis yang mendalam dalam menggunakan *tool* standar industri seperti Burp Suite (khususnya Repeater) untuk memanipulasi *request* dan SQLMap untuk otomatisasi eksploitasi.
5. Pentingnya Dokumentasi: Proses penyusunan laporan ini memberikan pemahaman tentang betapa pentingnya mendokumentasikan setiap langkah secara rinci, jelas, dan profesional, sesuai dengan standar laporan *bug bounty* atau *penetration testing*.

INFORMASI PENYUSUN

Laporan ini disusun sebagai syarat kelulusan Bootcamp Web Penetration Testing & Bug Bounty.

Nama: Malvin Wijaya

Batch: Batch 22

Penyelenggara: JadiHacker