LAPORAN PRAKTIKUM KEAMANAN INFORMASI 1
WEB XSS INJECTION & SQL *Injection*

DI SUSUN OLEH

Nama	: M Abdul Aziz
NIM	: 21/474516/SV/18951
Hari, Tanggal	: Selasa, 16 Mei 2023
Kelas	: RI4AA

LABORATORIUM PERANGKAT KERAS DAN LUNAK
PROGRAM SARJANA TERAPAN (DIV) TEKNOLOGI REKAYASA INTERNET
DEPARTEMEN TEKNIK ELEKTRO DAN INFORMATIKA
SEKOLAH VOKASI
UNIVERSITAS GADJAH MADA

2023

Praktikum Keamanan Informasi 1
Web XSS *Injection* & SQL *Injection*

I. Tujuan

- Melakukan pengujian *Cross Site Scripting Injection* atau XSS.
- Melakukan pengujian SQL *Injection*.

II. Landasan Teori

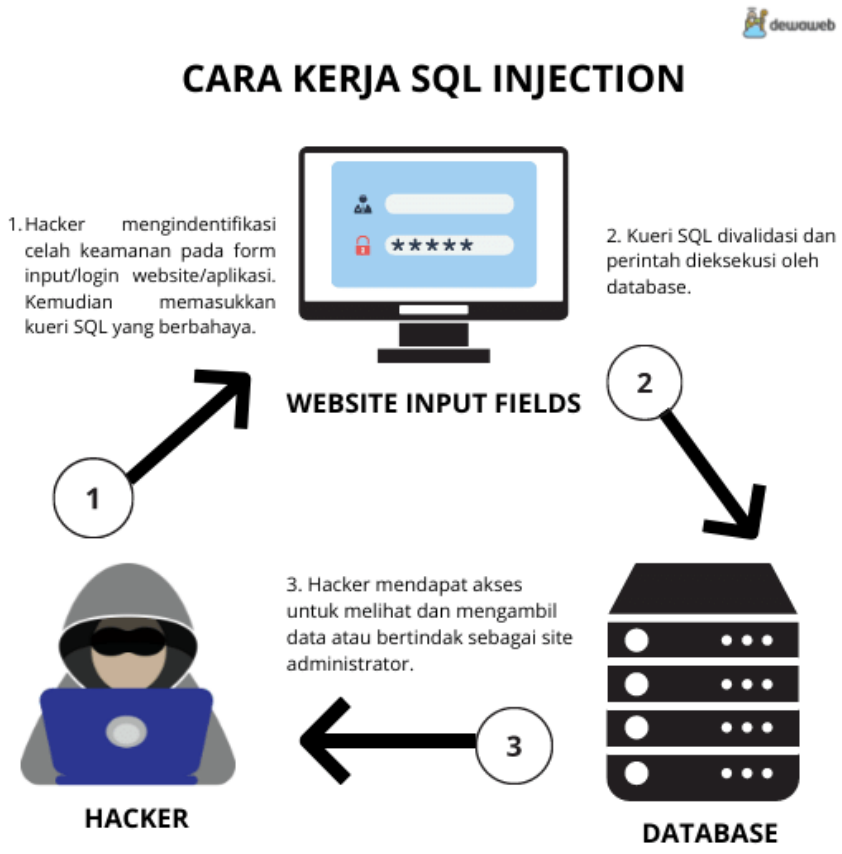
Cross site scripting (XSS) adalah serangan injeksi kode pada sisi klien dengan menggunakan sarana halaman website atau web aplikasi. Peretas akan mengeksekusi skrip berbahaya di browser korban dengan cara memasukkan kode berbahaya ke halaman web atau web aplikasi yang sah. Serangan ini dapat dilakukan menggunakan *JavaScript, VBScript, ActiveX, Flash,* dan bahasa sisi klien lainnya.

Forum, kolom komentar, dan *message boards* biasanya digunakan oleh penyerang untuk memposting *link* untuk membuat skrip berbahaya. Skrip tersebut kemudian akan menyerang ketika korban mengklik tautan tersebut. Cross site scripting ini sering digunakan untuk mencuri session cookies, yang memungkinkan penyerang untuk menyamar sebagai korban. Dengan cara inilah, peretas bisa mengetahui data-data sensitif milik korban.

Serangan *Cross Site Scripting* sendiri terdapat beberapa jenis yang memiliki karakter yang berbeda, diantaranya adalah *Stored* XSS (*Persistent* XSS), *Reflected* XSS *(Non-persistent* XSS*), Blind* XSS, *Self* XSS dan *DOM Based* XSS. *Stored* XSS merupakan serangan yang bersifat permanen dan bisa berakibat pada seluruh penggunanya. *Reflected* XSS *(Non-persistent* XSS*)* merupakan jenis *cross site scripting* yang tidak permanen. *Cross site scripting* tipe ini akan hilang apabila Anda melakukan refresh. *Hacker* akan menggunakan teknik *social engineering* agar pengguna mengakses situs yang telah terinfeksi dengan kode berbahaya, sehingga *hacker* dapat memperoleh data penting dari pengguna untuk melakukan kejahatan lain. *Blind* XSS merupakan kerentanan dari *cross site scripting* yang *hacker* sendiri sebenarnya tidak mengetahui kemana dan siapa *payload* tersebut akan diterima. *Self* XSS merupakan jenis *cross site scripting* yang memerlukan

proses urut dan hanya akan berdampak pada pribadi seseorang itu sendiri. Tipe ini biasanya akan dipadukan dengan *clickjacking*. *DOM Based XSS* dapat terjadi jika web aplikasi menulis data ke Document Object Model (DOM) tanpa sanitization yang tepat. Penyerang dapat memanipulasi data ini untuk memasukkan konten XSS pada halaman web seperti kode Javascript yang berbahaya.

SQL *Injection* adalah salah satu teknik peretasan dengan cara menyalahgunakan celah keamanan yang ada di lapisan SQL berbasis data suatu aplikasi. Terbentuknya celah tersebut akibat *input* yang tidak difilter dengan benar dalam pembuatannya, sehingga terciptalah celah yang bisa disalahgunakan. Umumnya, hacker menggunakan perintah atau *query* SQL dengan *tools* tertentu untuk mengakses *database*. Injeksi kode yang dilakukan membuat mereka dapat masuk tanpa proses otentikasi. Setelah berhasil, hacker bebas untuk menambahkan, menghapus, serta mengubah data-data pada website.



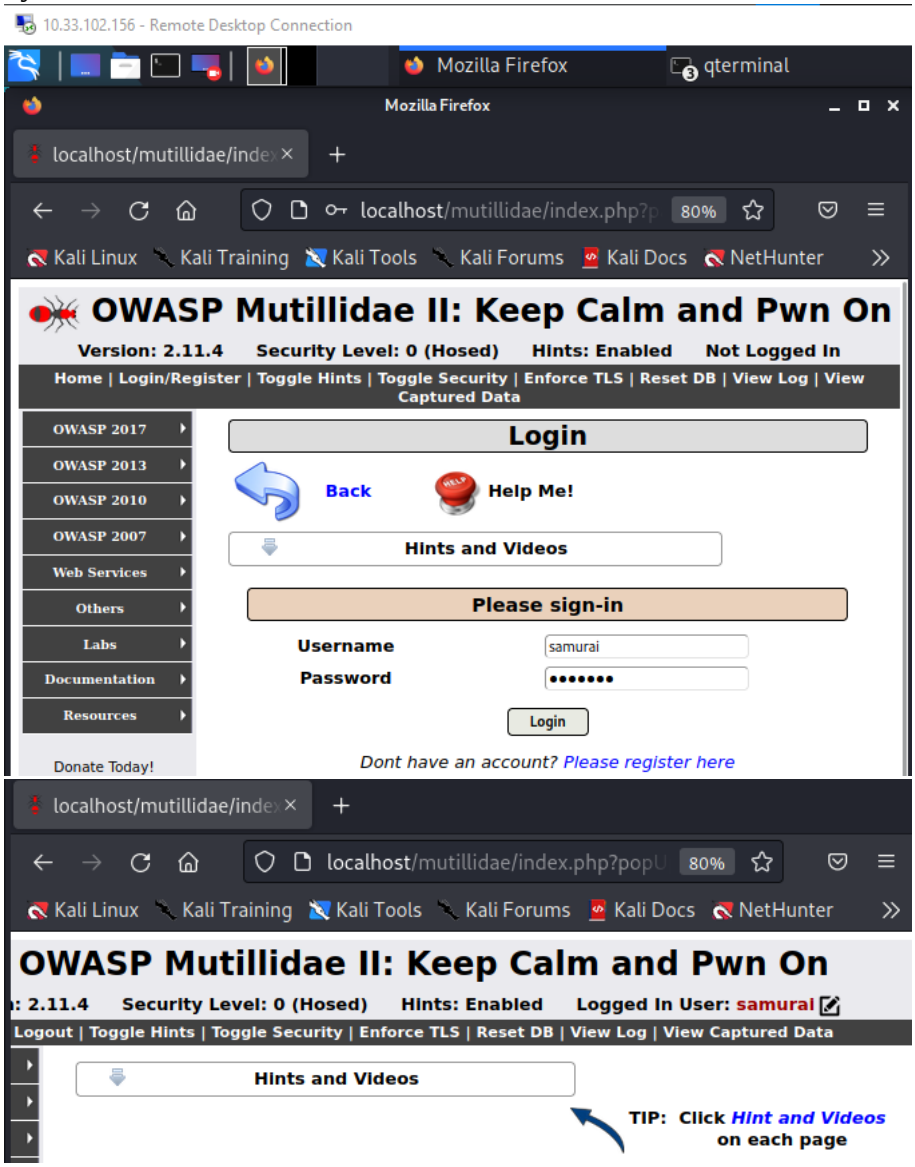
III. Alat & Bahan

- *Software Remote Desktop Connection*
- OS Kali Linux
- Laptop/PC
- Koneksi Internet

IV. Instruksi Kerja

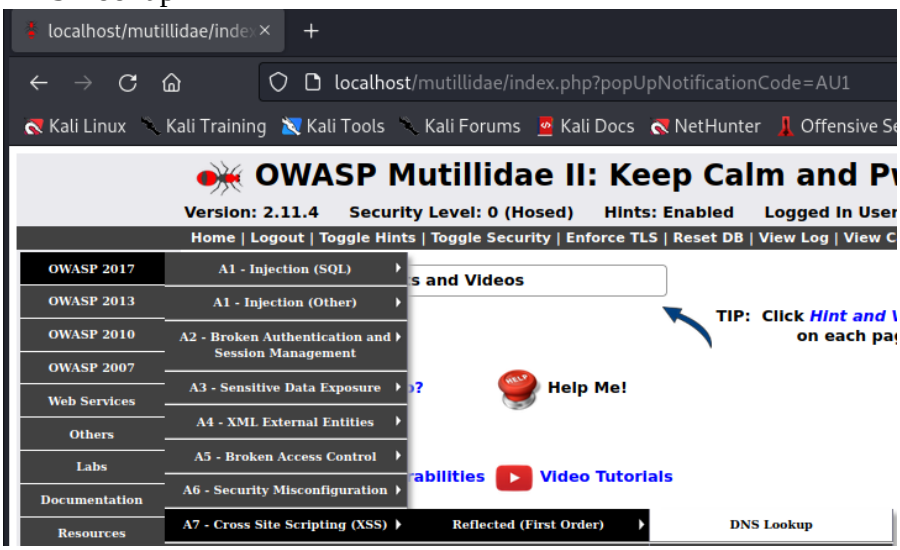
A. Langkah 1 : *Login*

Masuk ke Mutillidae untuk mensimulasikan pengguna yang masuk ke aplikasi nyata dan diberikan ID Sesi.

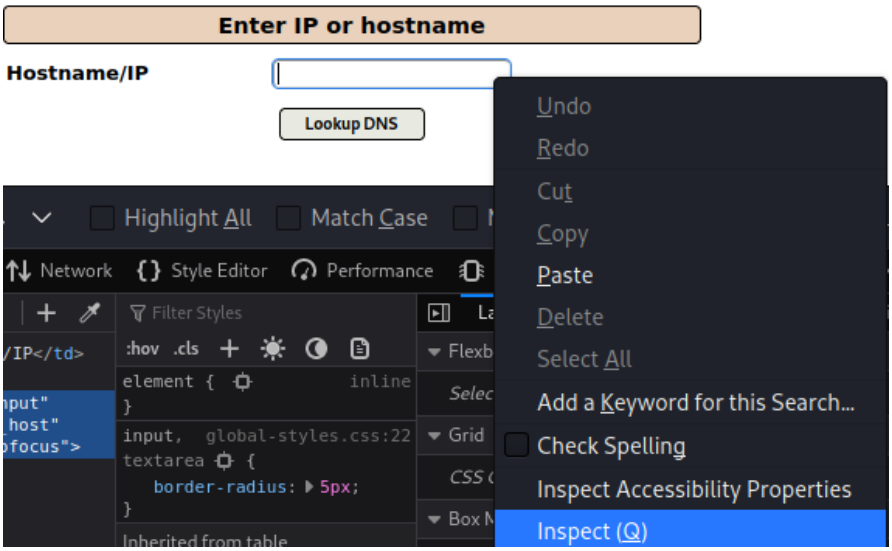


B. Langkah 2: *Reflected Cross Site Scripting* (XSS) *Injection* #1 - *Popup Window*

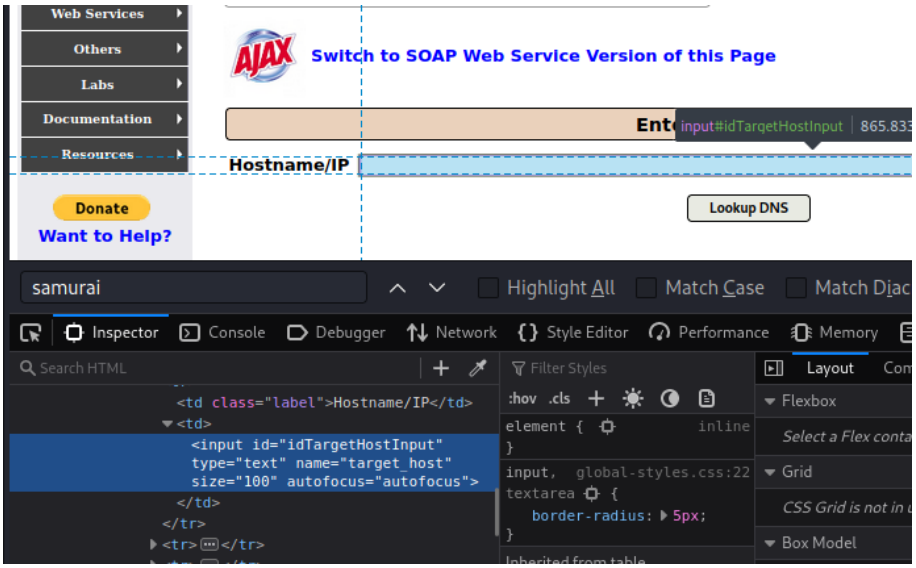
1. DNS Lookup



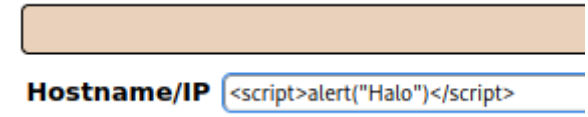
2. *Inspect Textbox Element*

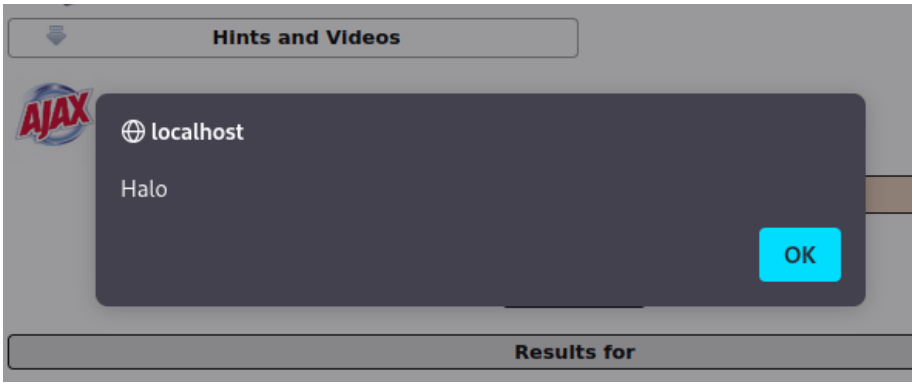


3. Ubah ukuran *Textbox*



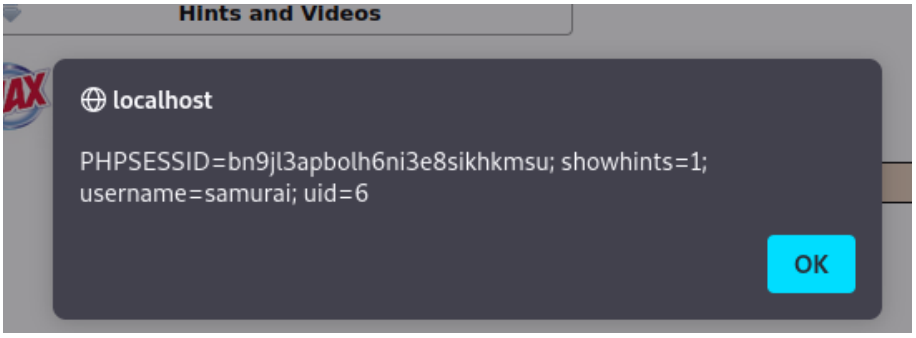
4. Uji Injeksi (XSS)



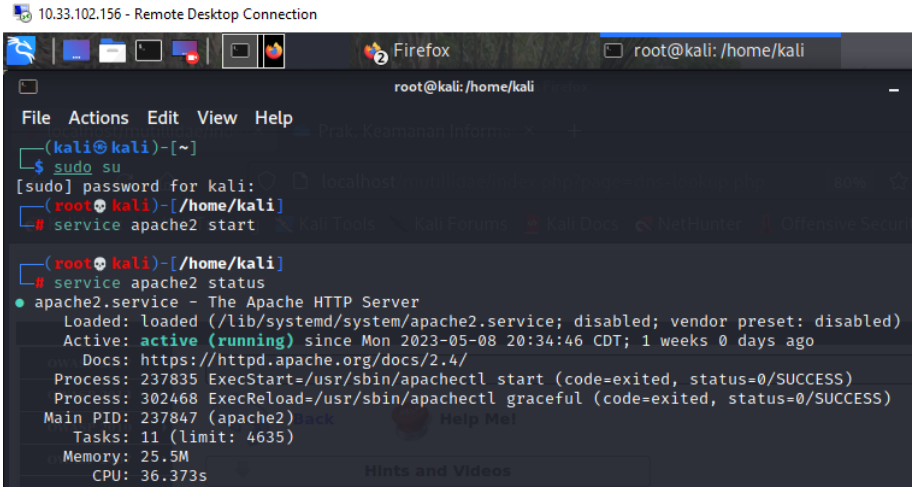


C. Langkah 3 : *Reflected Cross Site Scripting* (XSS) *Injection #2 - Popup Cookie*

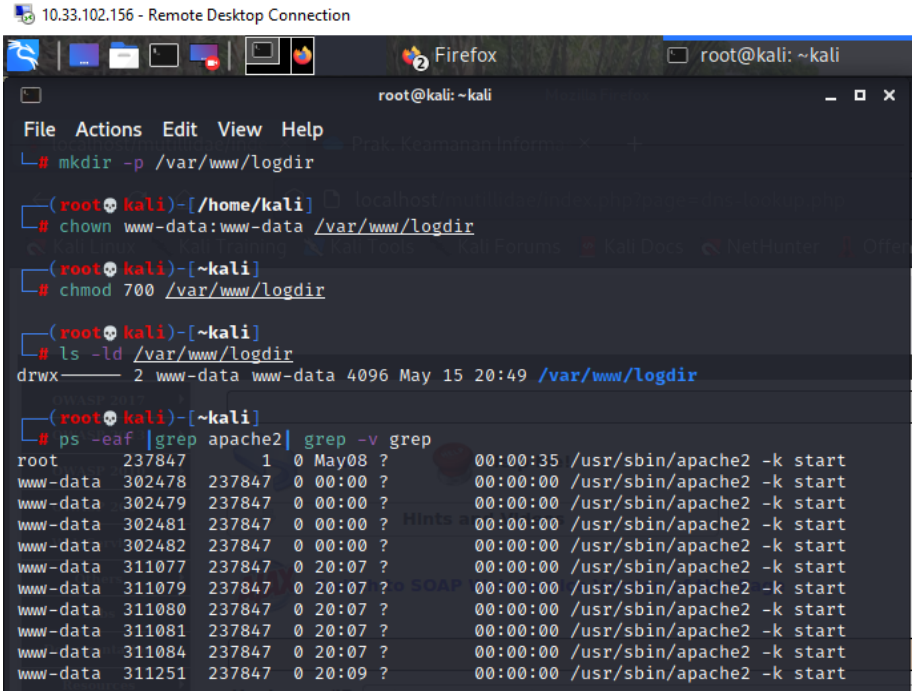
1. DNS *Lookup*
2. *Inspect Textbox*
3. Ubah ukuran *Textbox*
4. Uji Injeksi XSS



5. *Start* server Apache2



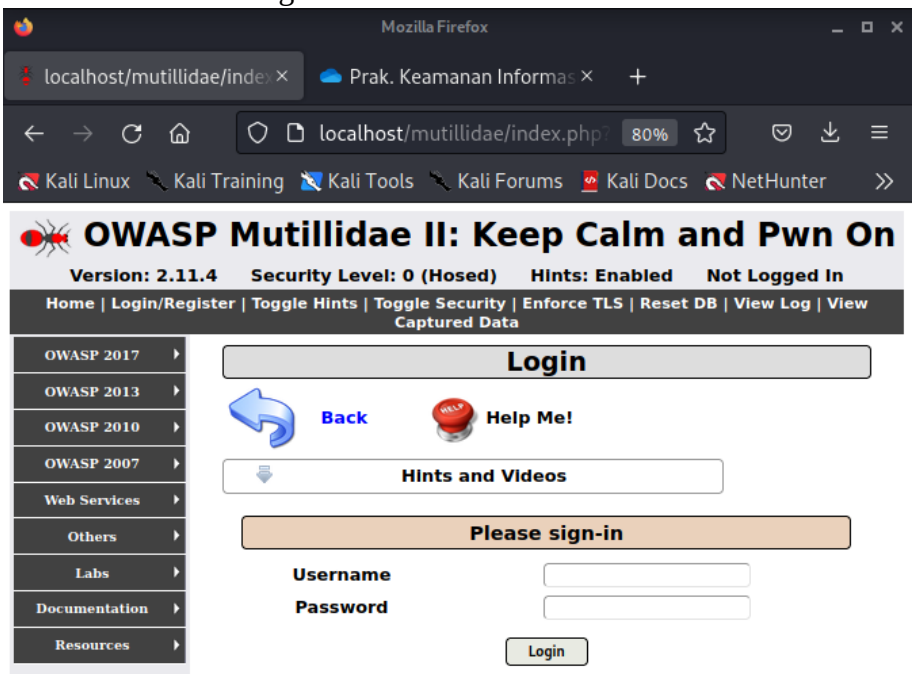
6. Buat direktori Log Apache



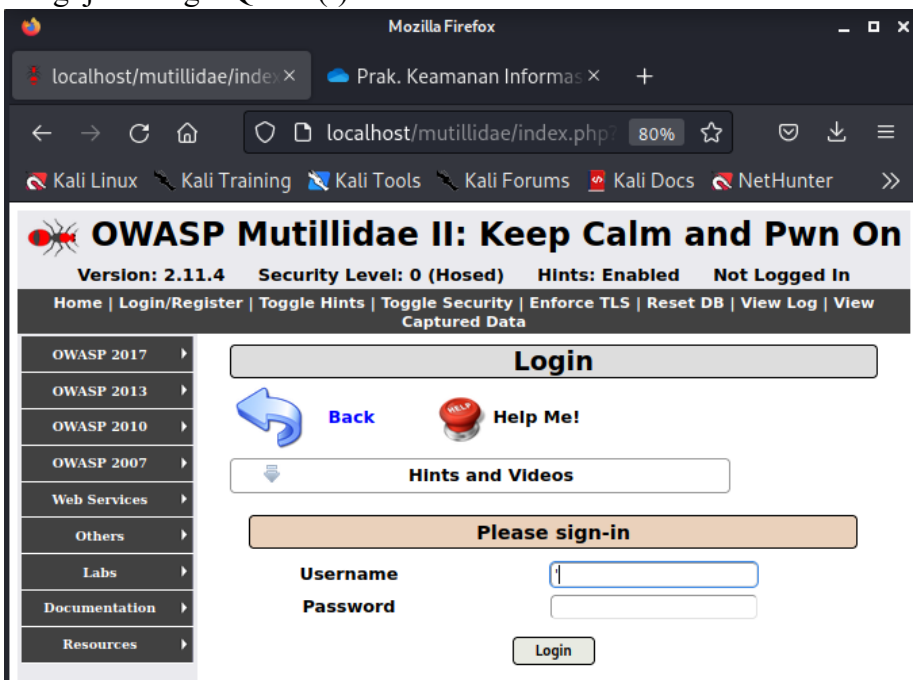
7. Konfigurasi CGI Cookie Script

D. Langkah 1: SQL Injection: : Single Quote Test pada form Username

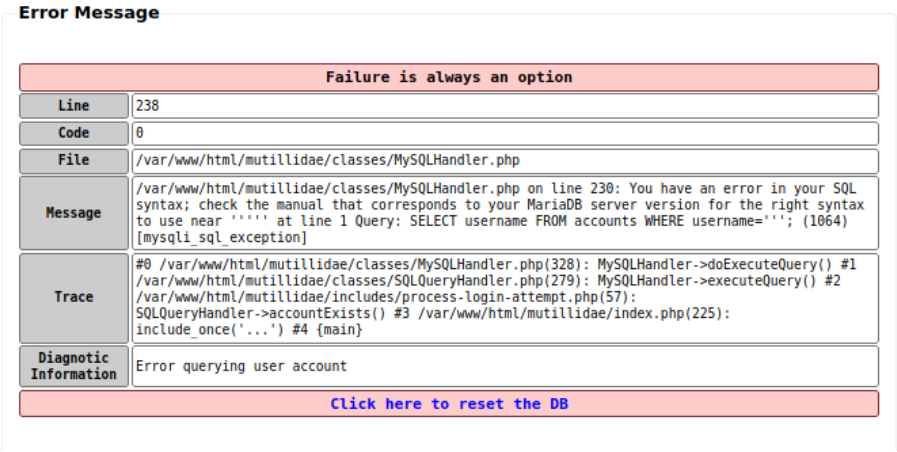
1. Masuk ke halaman login



2. Pengujian Single Quote (‘)



3. Hasil kutipan tunggal

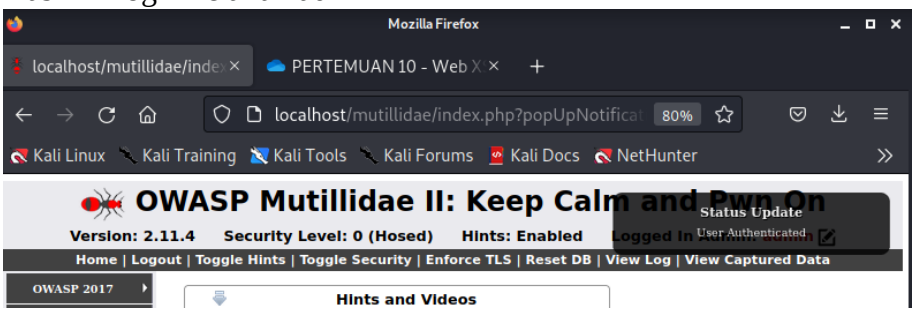


E. Langkah 2: SQL Injection: By-Pass Password tanpa Username

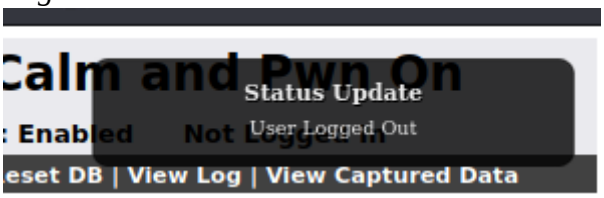
1. Login tanpa kata sandi

   Ketikkan ‘ or 1 = 1-- . Pastikan memberikan smasi setelah “--”. Selanjutnya klik tombol login

2. Hasil = Login ke akun admin

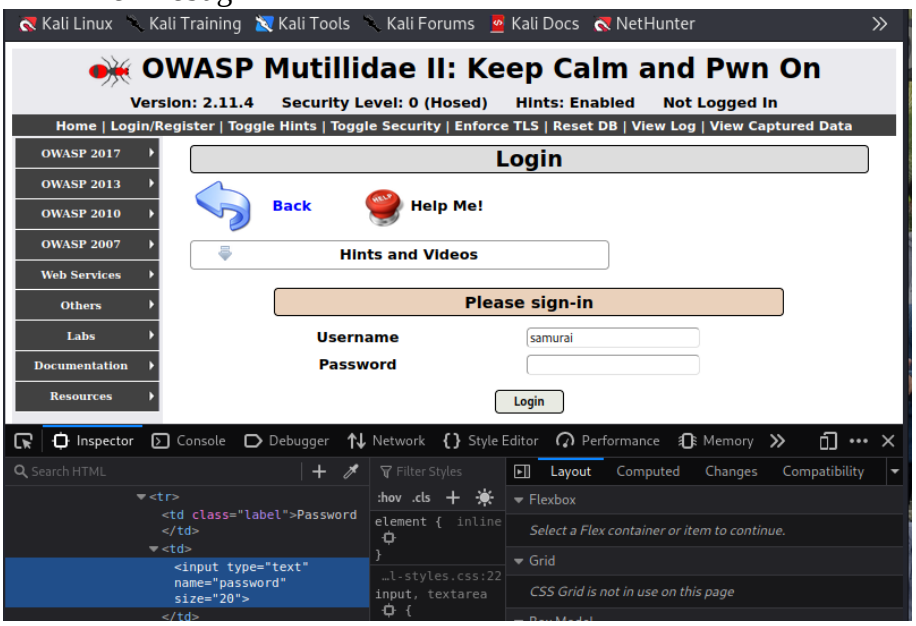


3. *Logout*

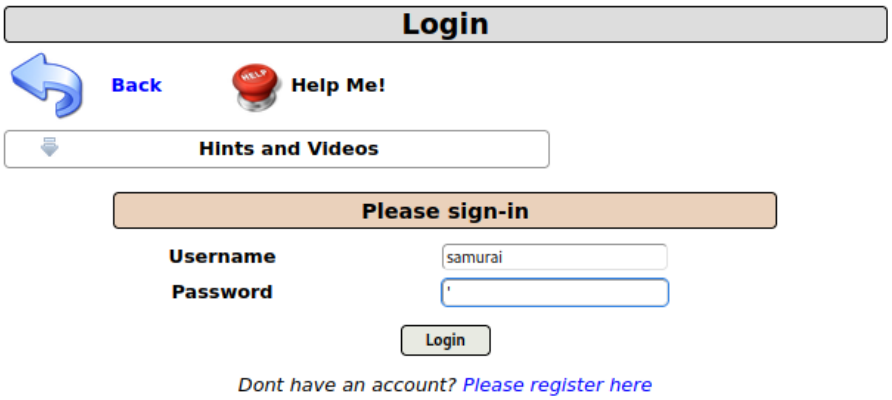


F. Langkah 3: SQL Injection: Single Quote Test On Password Field

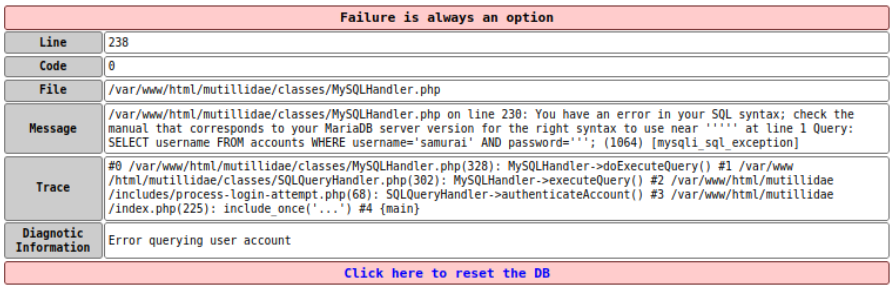
1. Klik Login/Register
2. Login dengan user : samurai
3. Klik kanan pada *Textbox Password*, inspect. Ubah type menjadi text, minimize Firebug



4. Masukkan *password* : ‘. Kemudian klik *Login*

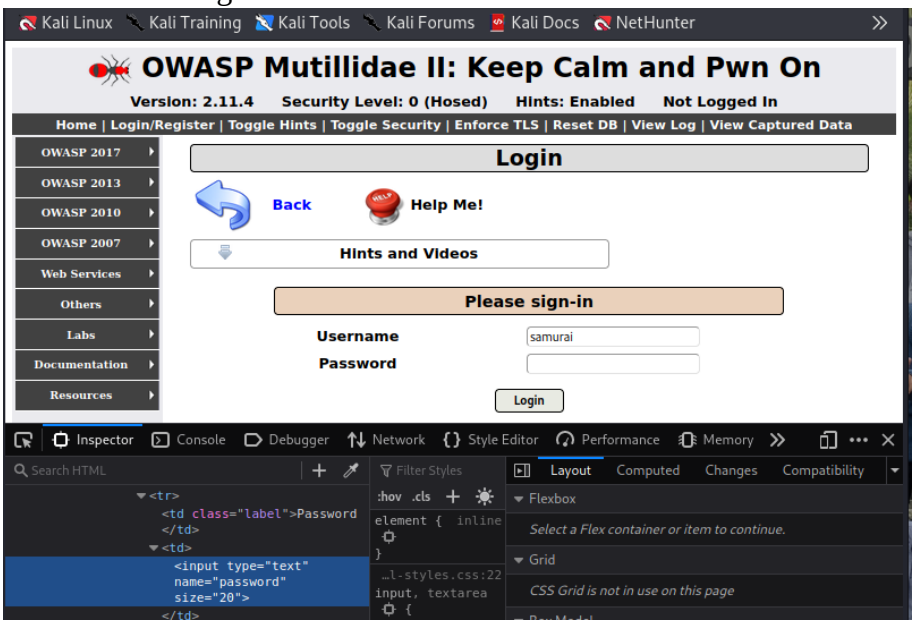


5. Hasil

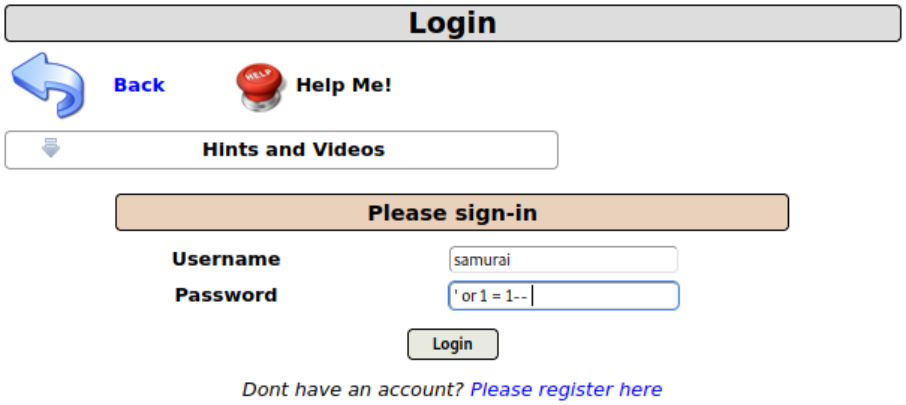


G. Langkah 4: SQL Injection: Single Quote Test On Password Field

1. Klik Login/Register
2. Login dengan user : samurai
3. Klik kanan pada *Textbox Password*, inspect. Ubah type menjadi text, minimize Firebug



4. Masukan password ‘ or 1 = 1-- .



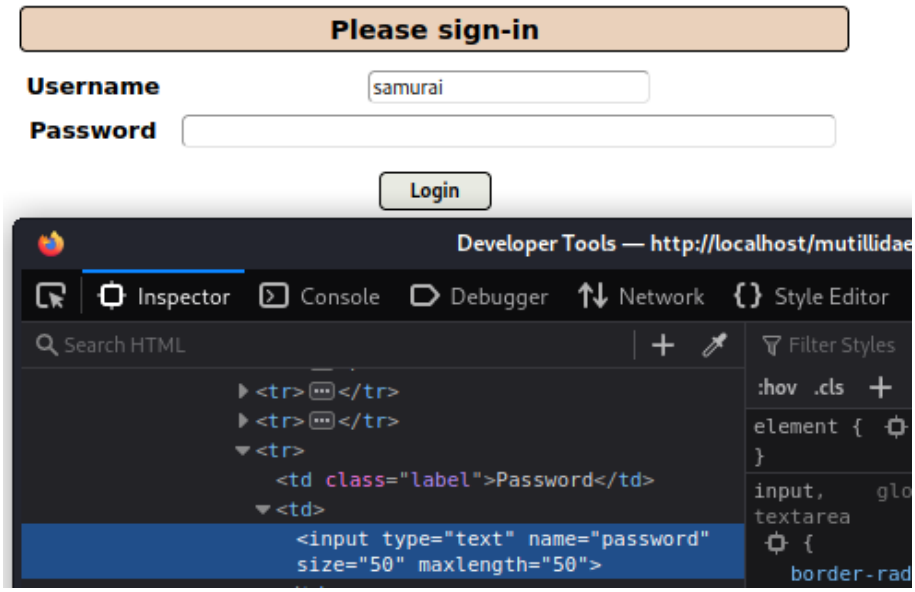
5. Hasil

Terdapat signature g0t r00t?

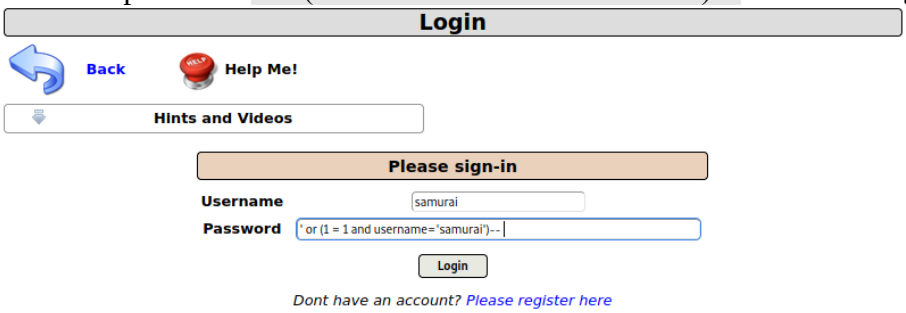
H. Langkah 5: SQL Injection: Single Quote Test On Password Field

1. Klik Login/Register
2. Login dengan user : samurai

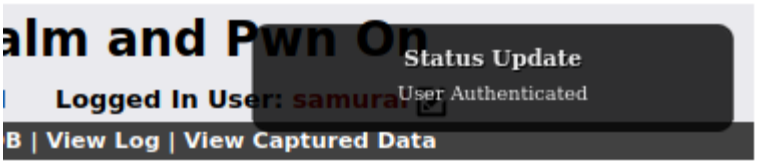
3. Klik kanan pada *Textbox Password*, inspect. Ubah type menjadi text, Ganti *size* menjadi 50, dan *maxlength* menjadi 50. Minimize Firebug.



4. Masukan password ‘ or (1 = 1 and username=’samurai’)-- . Lalu klik login



5. Hasil



V. Pembahasan

Pada praktikum ini, mahasiswa diminta untuk menjalankan praktikum serangan *Cross Site Scripting Injection* serta SQL *Injection*. Mahasiswa diminta untuk melakukan pengujian XSS Refleksi yang terjadi ketika data yang dikirim oleh pengguna langsung disertakan dalam tanggapan server dan dijalankan pada browser pengguna yang melihat tanggapan tersebut.

Sebelum melakukan pengujian serangan, jalankan terlebih dahulu Mutillidae, dimana Mutillidae merupakan web yang akan diserang sebagai simulasi pengguna yang masuk ke aplikasi nyata dan diberikan ID sesi. Serangan pertama yang dilakukan adalah serangan XSS Refleksi dengan *pop-up* window. Sebelumnya identifikasi terlebih dahulu parameter atau *input* yang dapat dimanipulasi untuk menyisipkan script, dalam hal ini adalah menu DNS *Lookup*. Sebagai penyerang juga mempersiapkan *payload* yang berisi kode JavaScript untuk membuka jendela pop-up, pada praktikum ini kodenya adalah *<script>alert("Halo")<\script>*. Kemudian *input* pada menu DNS *Lookup* dimanipulasi untuk menyisipkan *payload*. Sehingga ketika *payload* dieksekusi, akan muncul jendela *pop-up* yang tidak diinginkan, dalam hal ini jendela pop-up bertuliskan "Halo". Dampak dari serangan ini dapat menyebabkan gangguan pada pengalaman pengguna. Namun, untuk kasus yang lebih besar, jendela *pop-up* dapat digunakan untuk melakukan tindakan yang tidak diinginkan seperti mencuri informasi pengguna atau mengarahkan pengguna ke situs *phishing*.

Pengujian kedua serangan XSS Refleksi yaitu dengan *pop-up cookie* yang mana ini hampir sama dengan serangan yang sebelumnya, perbedaannya serangan ini akan memanipulasi *cookie* pengguna. Langkah awalnya pun sama di mana penyerang mengidentifikasi aplikasi web yang rentan dan menentukan *input* yang dapat dimanipulasi (dalam hal ini Mutillidae dengan menu DNS *Lookup*). Selanjutnya, masukkan *string* *<script>alert(document.cookie)</script>* pada *input* DNS *Lookup* dan cari DNS. Hal ini dilakukan untuk mengecek apakah halaman web berisi *cookie* dan apakah dapat menampilkan *cookie* di kotak peringatan JavaScript. Dari hasil yang muncul, dapat dilihat bahwa *cookie* menampilkan *username* dan PHP *Session* ID.

Kemudian sebagai penyerang masuk atau memulai Apache2. Hal ini berkaitan dengan serangan XSS Refleksi dengan pop-up *cookie* terjadi pada lapisan aplikasi web, dimana celah keamanan terjadi pada kode atau logika aplikasi yang dijalankan di atas server web seperti Apache. Cek pula status saat ini dari Apache2 dan informasi tentang proses-proses Apache HTTP Server yang sedang berjalan di sistem. Lalu dibuat Apache *Log Directory* yang akan menyimpan berbagai jenis log

yang berkaitan dengan operasi server dan aktivitasnya. Untuk mempermudah serangan, ubah kepemilikan direktori menjadi pengguna 'www-data' dan grup 'www-data' di mana ini akan memberikan pengguna dan grup tersebut kontrol penuh atas direktori log. Atur juga izin akses pada direktori log menggunakan mode '700' yang memberikan izin baca, tulis, dan eksekusi hanya kepada pemilik direktori yaitu pengguna 'www-data' dan grup 'www-data'. Karena izin akses sudah diatur, coba lihat informasi mengenai direktori log, yang mana dari hasil dapat dilihat bahwa ini adalah direktori ('d') dengan izin 'rwx' (baca, tulis, eksekusi) yang hanya untuk pemilik direktori, memiliki 2 jumlah entri (sub-direktori dan file), ukuran direktori sebesar 4096 bytes, direktori terakhir dimodifikasi pada 15 Mei pukul 20:57, serta *path* dari direktori adalah '/var/www/logdir'.

Selanjutnya untuk mengkonfigurasi CGI *Cookie Script*, pindah ke direktori 'usr/lib/cgi-bin' terlebih dahulu. Pada direktori ini, kita dapat mengakses dan bekerja dengan file-file CGI yang ada di dalamnya, seperti mengedit, menjalankan, atau melakukan tindakan lain yang diperlukan. Pada kasus ini, unduh *Cookie Script* CGI dari GitHub dan ubah namanya menjadi logit.pl. Lalu ubah kepemilikan file tersebut menjadi pengguna dan grup 'www-data', serta izin akses menjadi mode '700' agar pengguna dan grup tersebut dapat membaca, menulis, dan mengeksekusi file tersebut. Terakhir, periksa sintaksis dan kesalahan pada file Perl logit.pl. Hasil dari pemeriksaan ini yaitu file tidak memiliki kesalahan sintak. Pemeriksaan ini berguna untuk memeriksa dan menemukan kesalahan yang mungkin terjadi sebelum menjalankan file Perl secara efektif.

Selain melakukan pengujian terhadap serangan *Cross Site Scripting Injection*, pada praktikum ini juga dilakukan pengujian terhadap serangan SQL *Injection*. Salah satu jenis serangan SQL *Injection* yang umum adalah menggunakan tanda kutip tunggal ('), yang memungkinkan penyerang untuk menyisipkan kode SQL setelah tanda kutip tunggal tersebut. Dalam kasus ini, penyerang akan menguji apakah aplikasi rentan terhadap serangan SQL *Injection* dengan memasukkan tanda kutip tunggal sebagai *input* pada bidang yang menerima *string*, *seperti form username*. Penyerang dapat memasukkan *input* tersebut apabila *form username* tidak melakukan validasi atau perlindungan yang memadai.

Jenis SQL *Injection* yang kedua adalah *bypass password* tanpa *username*, jenis ini dapat terjadi jika aplikasi web tidak memvalidasi *input* dengan benar yang mana hal ini sudah terbukti di pengujian pertama. Payload yang digunakan untuk pengujian ini adalah frasa ‘' or 1 = 1-- ’. Tanda kutip tunggal (') pada frasa menutupi tanda kutip pada input dan membantu mengakhiri tanda kutip yang dibuka oleh *query* asli. OR digunakan untuk menghubungkan kondisi yang akan dievaluasi sebagai benar jika salah satunya benar. Angka 1 adalah angka yang benar dalam logika SQL. Tanda sama dengan (=) adalah operator perbandingan yang digunakan untuk membandingkan nilai. Tanda dua minus (--) adalah komentar dalam SQL yang mengakibatkan database mengabaikan sisa baris komentar. Jadi, jika frasa ‘' or 1 = 1-- ’ disisipkan ke dalam query SQL, kondisi 1 = 1 akan dievaluasi sebagai benar, yang berarti kondisi ini selalu terpenuhi. Dengan demikian, frasa tersebut dapat digunakan untuk mengubah arti *query* asli dan memungkinkan akses yang tidak sah atau mempengaruhi eksekusi *query* secara keseluruhan.

Pengujian SQL *Injection* selanjutnya adalah dengan *Single Quote Test On Password Field* yang mana dilakukan pada *input field password* dalam sebuah aplikasi web. Teknik "*Single Quote Test*" pada *password field* dilakukan dengan menyisipkan karakter tanda kutip tunggal (') pada *input password* yang dikirimkan ke aplikasi. Tujuan dari ini adalah untuk menguji bagaimana aplikasi menangani karakter khusus seperti tanda kutip dalam *query* SQL yang digunakan untuk memeriksa kecocokan *password*. Pada praktikum ini, pengujian dilakukan dengan tiga *payload* yang berbeda, pertama yaitu *single quote* itu sendiri ('), '‘ or 1 = 1-- ', dan '‘ or (1 = 1 and username=’samurai’)-- '.

VI. Kesimpulan

Pada praktikum kali ini dapat disimpulkan bahwa :

1. XXS *injection* dapat digunakan untuk menyisipkan *script* berbahaya pada suatu web.
2. XXS jenis *Reflected* biasa dijalankan dengan teknik *social engineering* yang mana pengguna akan menekan suatu tombol yang berisi *phising*, sehingga data pengguna dapat dimanfaatkan oleh penyerang untuk melakukan kejahatan lain.
3. SQL *Injection* dapat digunakan untuk memperoleh hak akses *database* dari suatu web

VII. Daftar Pustaka

Feradhita. (October 16, 2019). Cross site scripting (serangan XSS) : Pengertian dan Jenis-jenisnya. Retrieved May 24, 2023, from https://www.logique.co.id/blog/2019/10/16/serangan-cross-site-scripting/

Unknown. (Novemer 02, 2022). Apa itu XSS? Pengertian, Cara Kerja, dan Cara Mengatasinya (Lengkap). Retrieved May 24, 2023, from https://makinrajin.com/blog/xss-adalah/#Apa_Itu_XSS

Rizal. (November 29, 2022). Apa Itu SQL Injection? Kenali Pengertian & Contohnya. Retrieved May 24, 2023, from https://dqlab.id/apa-itu-sql-injection-kenali-pengertian-and-contohnya