LAPORAN PRAKTIKUM KEAMANAN INFORMASI 1

Pertemuan 10 – *Web XXS Injection*

DISUSUN OLEH

Nama	: Sofiyanatul Munawaroh
NIM	: 21/474781/SV/19035
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

Pertemuan 10 – *Web XSS Injection*

I. Tujuan

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

II. Latar Belakang

Perkembangan teknologi membuat *cybercrime* juga semakin marak terjadi, salah satunya adalah keberadaan XSS atau *Cross Site Scripting*. Tanpa disadari, XSS dapat menyerang beberapa *platform* besar, seperti Facebook, Google, hingga PayPal. Kemudian, mereka akan akan mencuri data dari platform-platform tersebut, mengendalikan sesi pengguna, menjalankan kode jahat, atau menggunakannya sebagai bagian dari serangan phishing. Itu sebabnya, *cross site scripting* masuk ke dalam daftar OWASP (*Open Web Application Security Project*).

XSS adalah eksploitasi keamanan di mana penyerang menempatkan *malicious client-end code* ke laman *web*. Tujuan utama dari penggunaan XSS yakni untuk mengambil data penting, mengambil *cookie* dari *user* atau mengirimkan suatu program yang dapat merusak *user*, namun seakan-akan penyebabnya adalah dari *web* itu sendiri.

Secara sederhana, XSS bekerja dengan mengeksekusi skrip berbahaya di *browser* korban dengan cara memasukkan kode berbahaya ke halaman *web* atau *web* aplikasi yang sah. Umumnya serangan ini dilakukan menggunakan JavaScript, VBScript, ActiveX, Flash, dan bahasa sisi klien lainnya. Nantinya, penyerang akan menghubungi para korban melalui forum, kolom komentar, hingga *message boards* dengan mengunggah *link* untuk membuat skrip yang berbahaya. Ketika korban mengklik *link* tersebut, skrip mulai menyerang dan menyamar sebagai korban. Melalui cara ini, para *hacker* dapat mengetahui data-data milik korban.

Jenis ancaman kejahatan dalam dunia digital lainnya adalah SQL *Injection.* 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 tercipta 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*.

Sederhananya, teknik ini dijalankan melalui *form username*. *Form username* seharusnya diisi menggunakan karakter saja, tapi *form* tersebut bisa diisi dengan karakter lain. Dengan begitu, *hacker* bisa menyematkan karakter kontrol SQL seperti (:;-,=') dan kata kunci perintah yang dapat merusak tatanan *database*. Alhasil, *hacker* mampu memasukkan kueri SQL *Injection* dan *website* telah berhasil diretas.

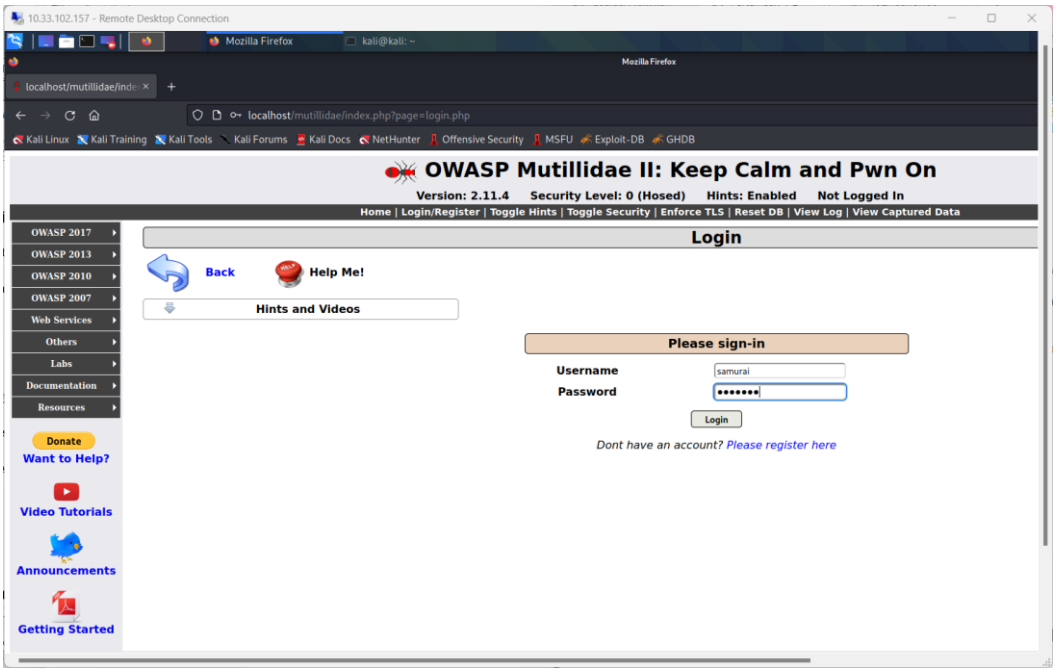
III. Alat dan Bahan

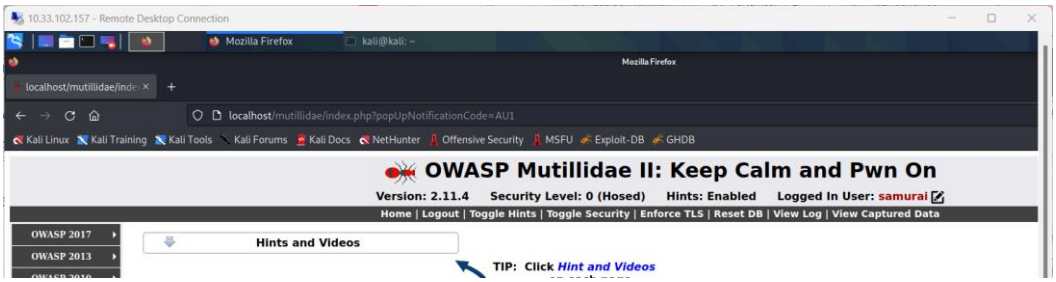
- *Software Remote Desktop Connection*
- Kali Linux
- Laptop/PC
- Koneksi Internet

IV. Instruksi Kerja

A. Langkah 1: Login

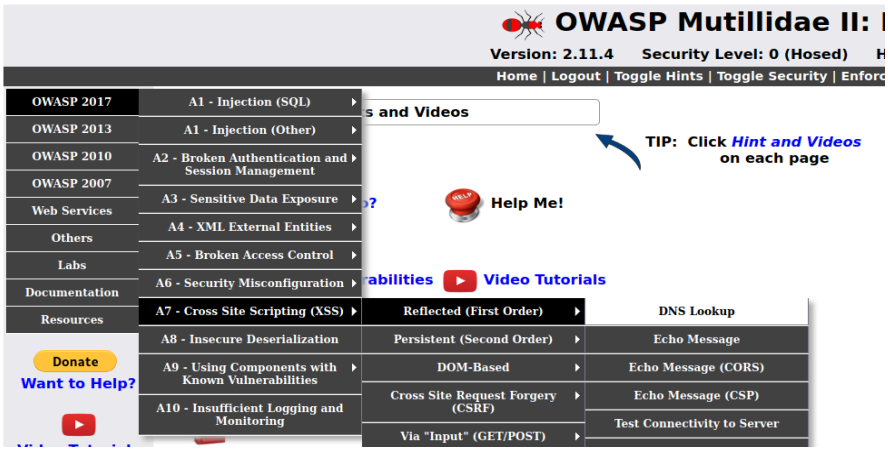
Masuk ke Mutillidae untuk mensimulasikan pengguna yang masuk ke aplikasi nyata dan diberikan ID Sesi.



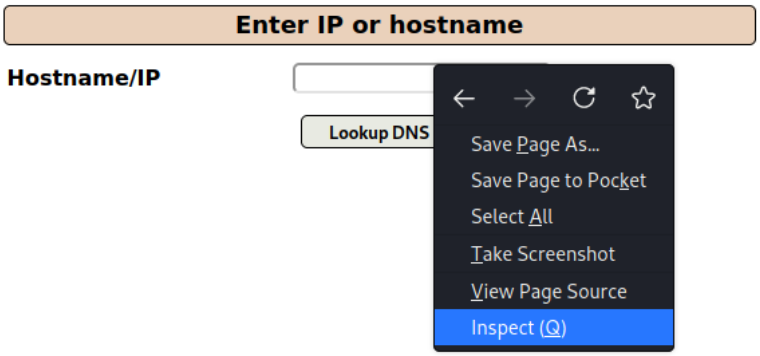


B. Langkah 2: *Reflected Cross Site Scripting* (XSS) *Injection* #1 – *Popup Window*

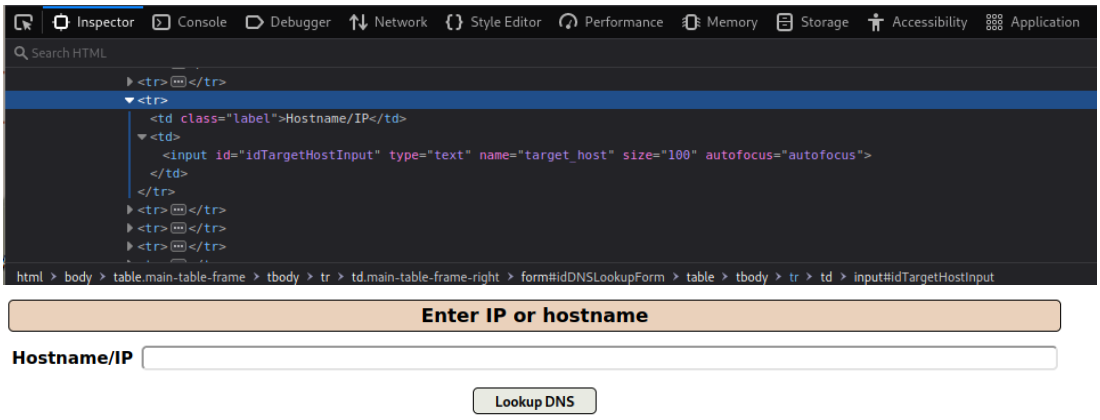
1. *DNS Lookup*



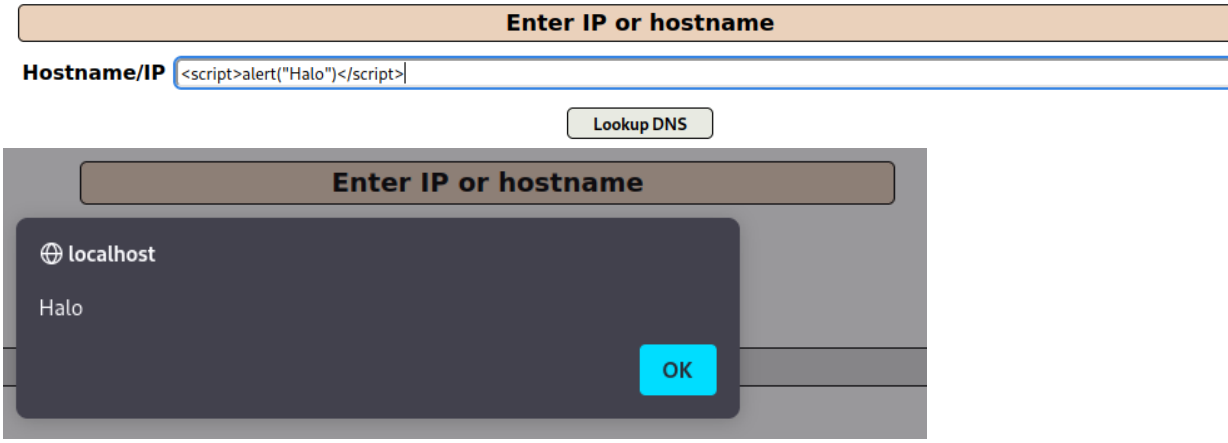
2. *Inspect Textbox Element*



3. Ubah ukuran *Text Box*

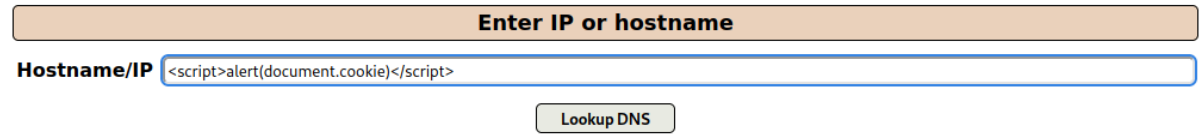


4. Uji Injeksi (XSS)

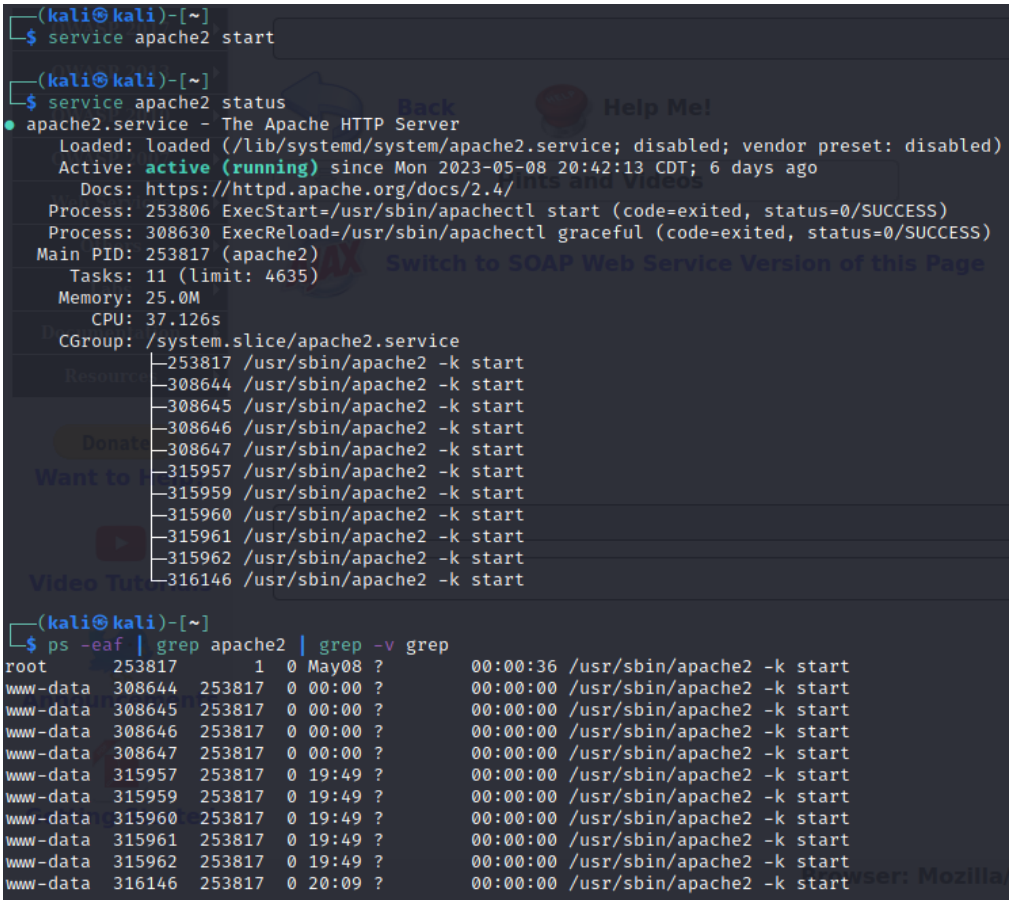


C. Langkah 3: *Reflected Cross Site Scripting* (XSS) *Injection* #2 – *Popup Cookie*

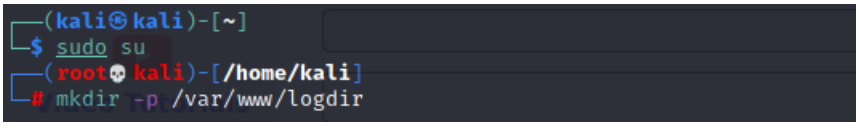
1. *DNS Lookup*
2. *Inspect Textbox*
3. Ubah ukuran *Textbox*
4. Uji Injeksi (XSS)



5. Memulai server Apache2



6. Buatlah direktori Apache Log Directory.



```
┌──(root💀kali)-[/home/kali]
└─# chown www-data:www-data /var/www/logdir

┌──(root💀kali)-[/home/kali]
└─# chmod 700 /var/www/logdir

┌──(root💀kali)-[/home/kali]
└─# ls -ld /var/www/logdir
drwx------ 2 www-data www-data 4096 May 15 20:38 /var/www/logdir
```

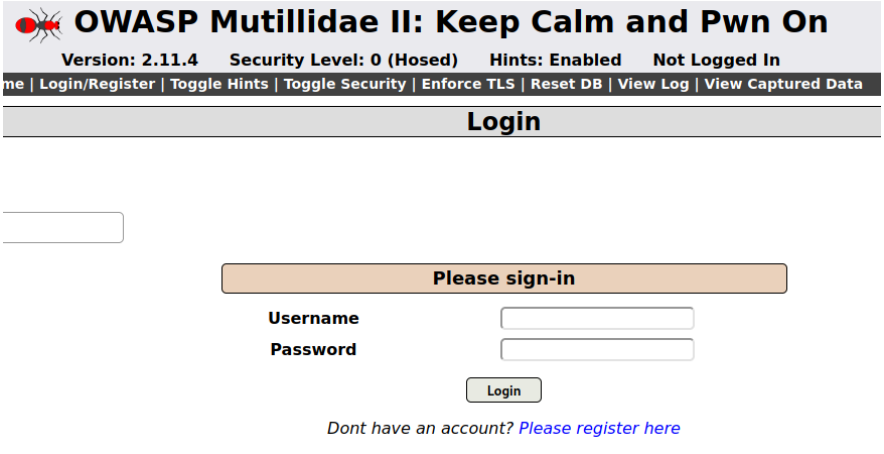
7. Konfigurasi CGI *Cookie Script*

```
┌──(root💀kali)-[/home/kali]
└─# cd /usr/lib/cgi-bin

┌──(root💀kali)-[/usr/lib/cgi-bin]
└─# wget https://github.com/cianni20/logit.git mv logit.pl.TXT logit.pl
--2023-05-15 20:44:59--  https://github.com/cianni20/logit.git
Resolving github.com (github.com)... 20.205.243.166
Connecting to github.com (github.com)|20.205.243.166|:443... connected.
GnuTLS: Error in the pull function.
Unable to establish SSL connection.
--2023-05-15 20:45:06--  http://mv/
Resolving mv (mv)... failed: No address associated with hostname.
wget: unable to resolve host address 'mv'
--2023-05-15 20:45:06--  http://logit.pl.txt/
Resolving logit.pl.txt (logit.pl.txt)... failed: Name or service not known.
wget: unable to resolve host address 'logit.pl.txt'
--2023-05-15 20:45:06--  http://logit.pl/
Resolving logit.pl (logit.pl)... 213.186.33.5
Connecting to logit.pl (logit.pl)|213.186.33.5|:80... connected.
HTTP request sent, awaiting response... 302 Moved Temporarily
Location: http://10.13.254.233:80/slogin/appoint.html?_URL_=http://logit.pl%2f&appoint=https://internet.ugm.ac.id/en/ [following]
--2023-05-15 20:45:07--  http://10.13.254.233/slogin/appoint.html?_URL_=http://logit.pl%2f&appoint=https://internet.ugm.ac.id/en/
Connecting to 10.13.254.233:80... connected.
HTTP request sent, awaiting response... 302 Moved Temporarily
Location: https://internet.ugm.ac.id/en/ [following]
--2023-05-15 20:45:07--  https://internet.ugm.ac.id/en/
Resolving internet.ugm.ac.id (internet.ugm.ac.id)... 10.13.243.12
Connecting to internet.ugm.ac.id (internet.ugm.ac.id)|10.13.243.12|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10062 (9.8K) [text/html]
Saving to: 'index.html'

index.html                                          100%[=================================================>]

2023-05-15 20:45:07 (57.8 MB/s) - 'index.html' saved [10062/10062]

FINISHED --2023-05-15 20:45:07--
Total wall clock time: 7.6s
Downloaded: 1 files, 9.8K in 0s (57.8 MB/s)
```

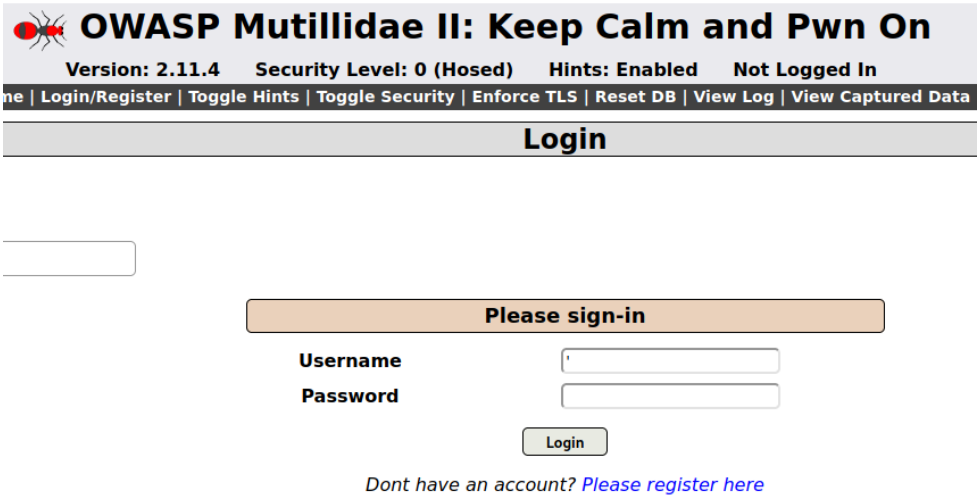
```
┌──(root💀kali)-[/usr/lib/cgi-bin]
└─# chown www-data:www-data logit.pl

┌──(root💀kali)-[/usr/lib/cgi-bin]
└─# chmod 700 logit.pl

┌──(root💀kali)-[/usr/lib/cgi-bin]
└─# perl -c logit.pl
logit.pl syntax OK
```

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

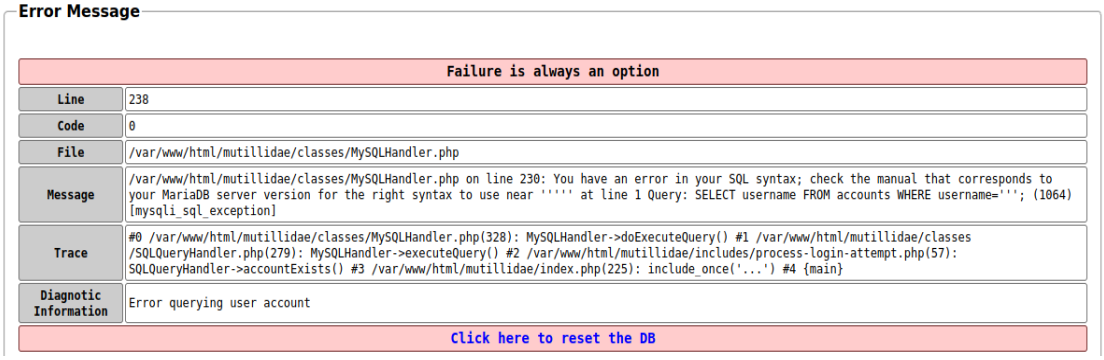
1. Masuk ke halaman *Login*.



2. Pengujian *Single Quote* (‘)



3. Hasil *Single Quote*



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

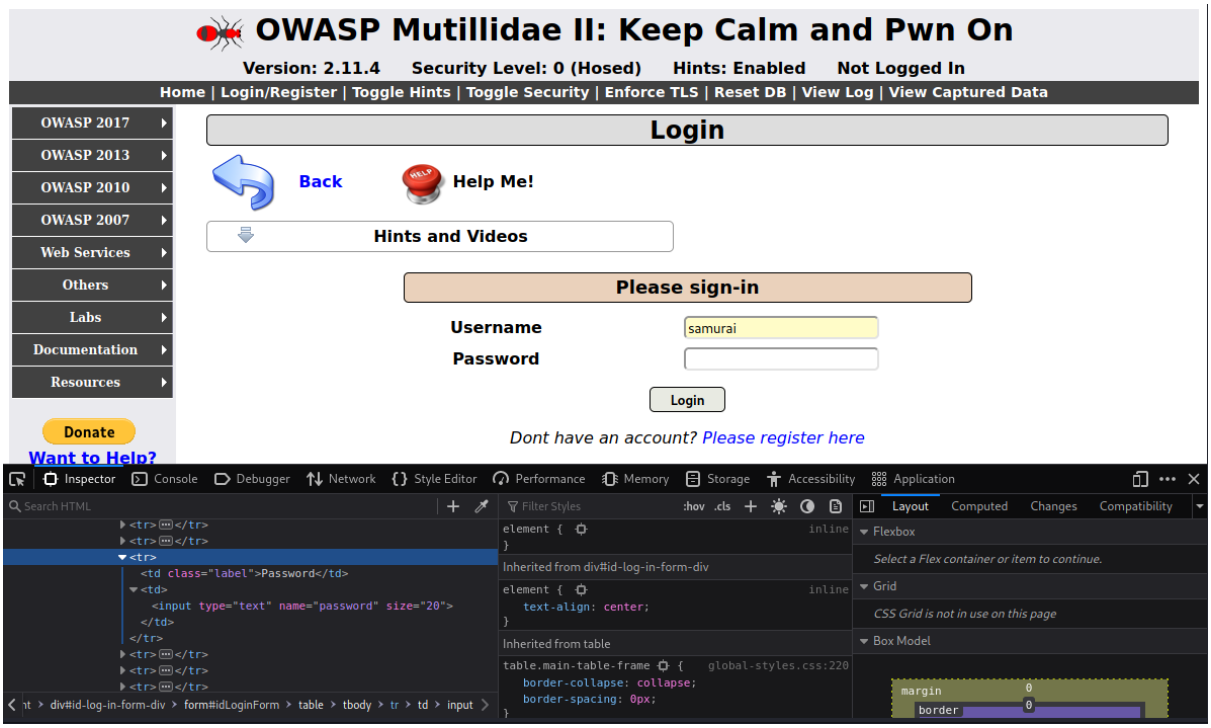
1. *Login* tanpa kata sandi
   Ketikkan ‘ or 1 = 1-- . Pastikan memberikan spasi setelah “--”. Selanjutnya klik tombol *login.*

2. Hasil = *Login* ke akun admin

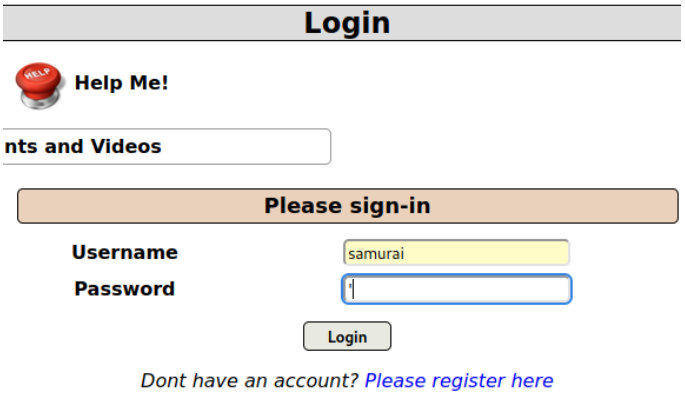
3. *Logout*

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*, pilih *inspect*. Ubah *type* menjadi *text*, lalu *minimize Firebug.*



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



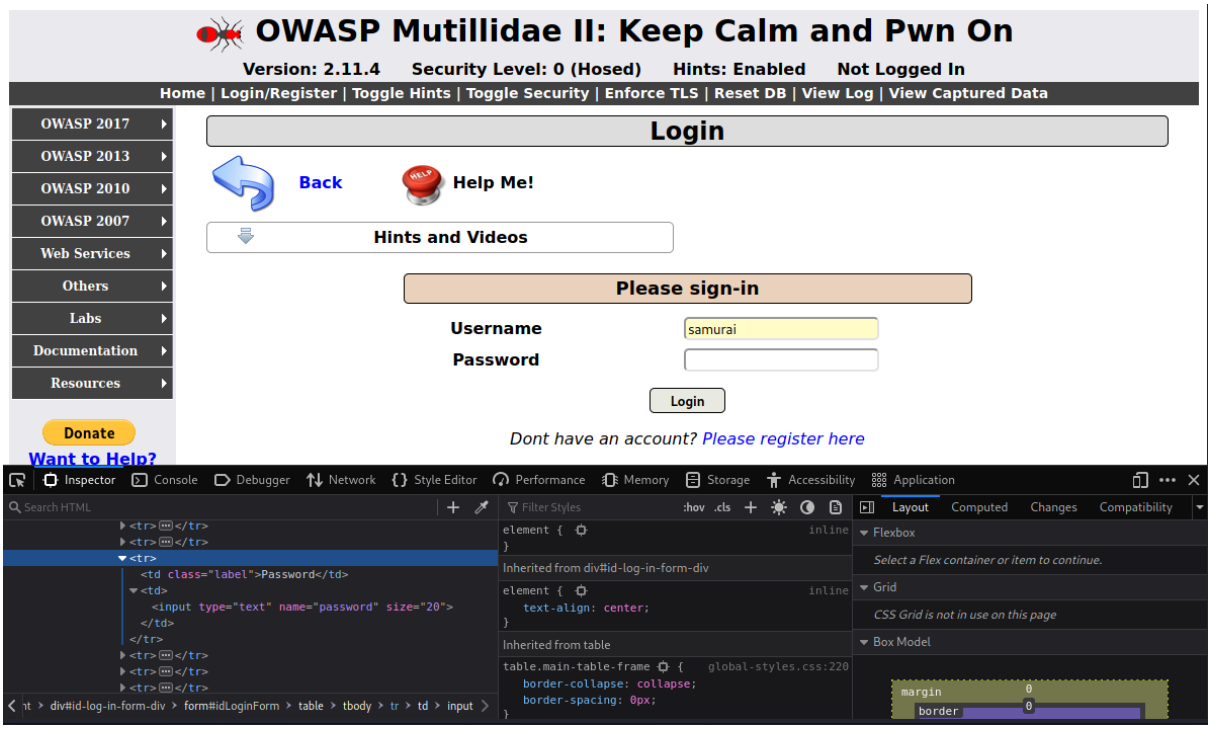
5. Hasil

Error Message

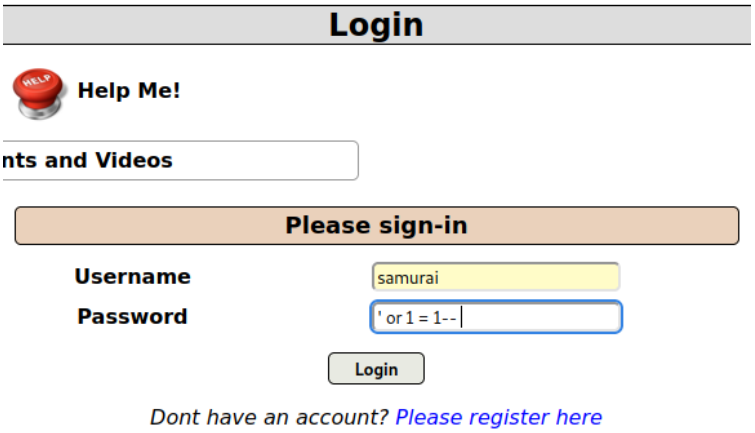
Failure is always an option	
Line	238
Code	0
File	/var/www/html/mutillidae/classes/MySQLHandler.php
Message	/var/www/html/mutillidae/classes/MySQLHandler.php on line 230: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near ''''' at line 1 Query: SELECT username FROM accounts WHERE username='samurai' AND password='''; (1064) [mysqli_sql_exception]
Trace	#0 /var/www/html/mutillidae/classes/MySQLHandler.php(328): MySQLHandler->doExecuteQuery() #1 /var/www/html/mutillidae/classes/SQLQueryHandler.php(302): MySQLHandler->executeQuery() #2 /var/www/html/mutillidae/includes/process-login-attempt.php(68): SQLQueryHandler->authenticateAccount() #3 /var/www/html/mutillidae/index.php(225): include_once('...') #4 {main}
Diagnotic Information	Error querying user account
Click here to reset the DB	

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

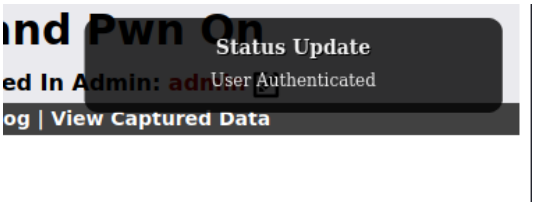
1. Klik *Login/Register*
2. *Login* dengan *user*: samurai
3. Klik kanan pada *Textbox Password*, pilih *inspect*. Ubah *type* menjadi *text*, lalu *minimize Firebug.*



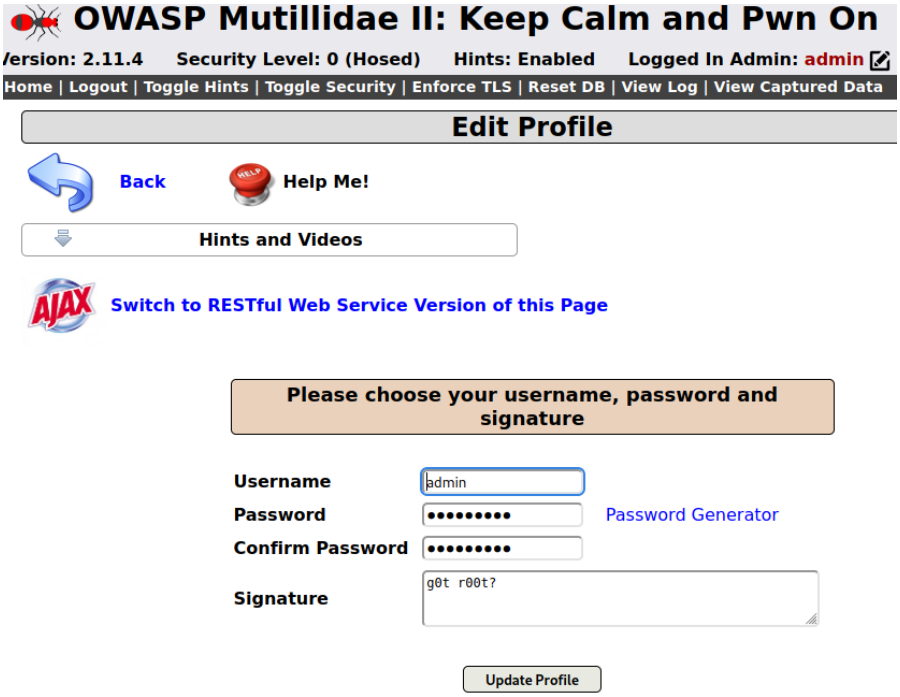
4. Masukan *password*: ‘ or 1 = 1-- .



5. Hasil

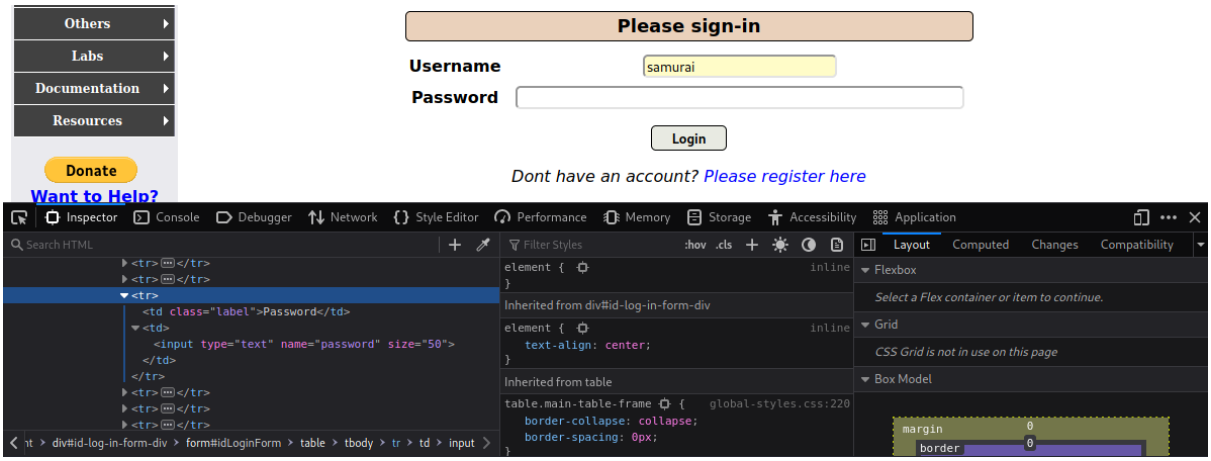


Cek apakah terdapat *signature* g0t r00t.

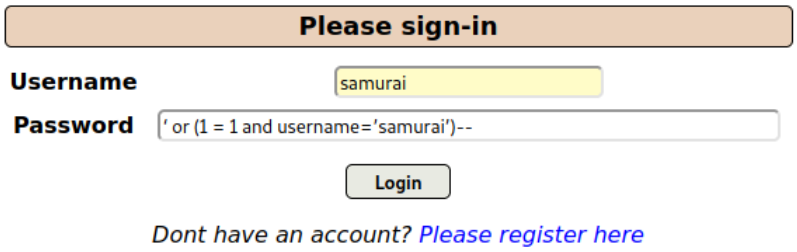


H. Langkah 5: SQL *Injection*: *Single Quote Test On Password Field*

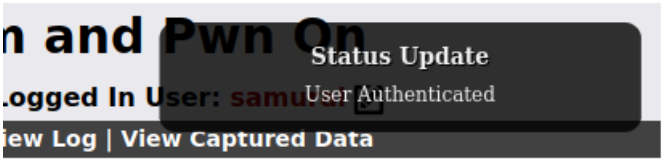
1. Klik *Login/Register*
2. *Login* dengan *user*: samurai
3. Klik kanan pada *Textbox Password*, pilih *inspect*. Ubah *type* menjadi *text*, ganti *size* menjadi 50, dan *max length* menjadi 50. Kemudian *Minimize Firebug*.



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



5. Hasil



V. Pembahasan

Pada praktikum ini, mahasiswa diminta untuk melakukan pengujian terhadap serangan *Cross Site Scripting Injection* dan *SQL Injection. Cross-Site Scripting* (XSS) adalah serangan keamanan yang memungkinkan penyerang menyisipkan *script* berbahaya ke halaman *web* yang dilihat oleh pengguna lain. Serangan ini dapat memiliki berbagai bentuk, seperti XSS Persisten dan XSS Refleksi. Namun, pada praktikum ini yang diuji adalah 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, masuk ke Mutillidae (*web* yang akan diserang) untuk mensimulasikan 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, di mana 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 20:38, 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 sintaksis. 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.* SQL *Injection* adalah serangan keamanan yang dilakukan pada aplikasi atau situs web yang menggunakan *input* pengguna untuk membangun atau mengeksekusi pernyataan SQL. 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, seringkali 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 *by-pass 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

1. Terdapat berbagai hal yang perlu dipersiapkan sebelum melakukan serangan, tergantung dari serangan yang ingin diluncurkan.
2. Pada XSS Refleksi, penyerang dapat mengubah kepemilikan dan akses sesuai dengan kehendaknya.
3. SQL *Injection* biasanya menggunakan *Single Quote* sebagai metodenya.

VII. Daftar Pustaka

Johanna. (2022). *Apa itu XSS? Cara Kerja, Jenis, dan Cara Mencegahnya*. Diakses pada 17 Mei 2023 dari https://www.dewaweb.com/blog/apa-itu-xss/

MR, Salsabila. (2022). *Apa Itu SQL Injection? Kenali Pengertian & Contohnya*. Diakses pada 17 Mei 2023 dari https://dqlab.id/apa-itu-sql-injection-kenali-pengertian-and-contohnya