

# **Group Project**

## **Testing a Web Application for Security**

**DPI902 Penetration Testing and Software Security  
Assessment**

**Anson Tan Wui Kang, Gabriel Xu Au Tao, Jessica Zhao, Huiwen Huang**

<b>Introduction</b>	<b>3</b>
<b>Description of Web Application</b>	<b>3</b>
<b>Description of each round testing</b>	<b>5</b>
Broken authentication and session management	10
Cross Site Scripting XSS	12
Injection	13
Sensitive Data Exposure	15
Insufficient Attack Protection	16
Server Side Scripts	19
<b>Summary Comments</b>	<b>21</b>

# Introduction

The purposes of web application that developed in this group project are looking vulnerables of the web application and trying to exploit the vulnerables by using some hacking tools. Majority vulnerables are from top 10 OWASP web vulnerabilities such as Cross Site Scripting, SQL injection, broken authentication and session management. Several hacking tools are used in this penetration including Nmap, Arachni, Nessus and etc.

Our web application has a registration system that allow user to register an user account to log into main page to leave a comment on the sites. Registration system will ask user to input first name, last name, email, password and birth date with specific requirements to register an user account such as password has to be more than 8 characters or first name has to be alphabet instead of numbers or symbols. Logged user will be able to leave a comment on the textform which the system will store the comment into database.

## Description of Web Application

The web application contains two files which are login.cgi and registration.cgi. Web application is written in Python and HTML which saved into cgi file. Web application will be running on Ubuntu Linux with several web servers including MySQL and Apache 2. All passwords will be encrypted by using MD5 and stored in MySQL database, this creates hash number on database that unread by others. Furthermore, comment section will store comments that leave by user into MySQL database with logged email. User initially go to login page, if they don't register any account, they need to click Registration and go to registration.cgi page to enroll into the system, and then login to the web application to write common.

# Penetration Testing Login

Email

Password

[Registration](#)

# Penetration Testing Registration

First name

Last name

Email

Re-enter email

Password

Birthday

[Login](#)

User : axu10@myseneca.ca

# Welcome to Penetration Testing!

**Warning !!** Comment box is currently not fully functional

# Description of each round testing

Before each round of testing, we run couples scanning tools such as Nmap, Nikto, Arachni and Nessus to scan vulnerabilities of host to gather information before hacking so that we can focus on the specific vulnerability to exploit. Below screenshots are taken from Nessus which indicating vulnerabilities of the host

DPI902

CURRENT RESULTS: TODAY AT 1:20 PM

Configure

Audit Trail

Launch

Export

Filter Vulnerabilities

Hosts > 172.16.1.223 > Vulnerabilities 10

<input type="checkbox"/>	Severity	Plugin Name	Plugin Family	Count
<input type="checkbox"/>	HIGH	Apache 2.2.x < 2.2.33-d...	Web Servers	1
<input type="checkbox"/>	INFO	Nessus SYN scanner	Port scanners	2
<input type="checkbox"/>	INFO	External URLs	Web Servers	1
<input type="checkbox"/>	INFO	HTTP Methods Allowed ...	Web Servers	1
<input type="checkbox"/>	INFO	HTTP Server Type and ...	Web Servers	1
<input type="checkbox"/>	INFO	HyperText Transfer Prot...	Web Servers	1
<input type="checkbox"/>	INFO	Missing or Permissiv... <span>Plugin ID: 50345</span> Uses		1
<input type="checkbox"/>	INFO	Missing or Permissive ...	CGI abuses	1
<input type="checkbox"/>	INFO	Web Application Sitemap	Web Servers	1
<input type="checkbox"/>	INFO	Web Server Directory E...	Web Servers	1

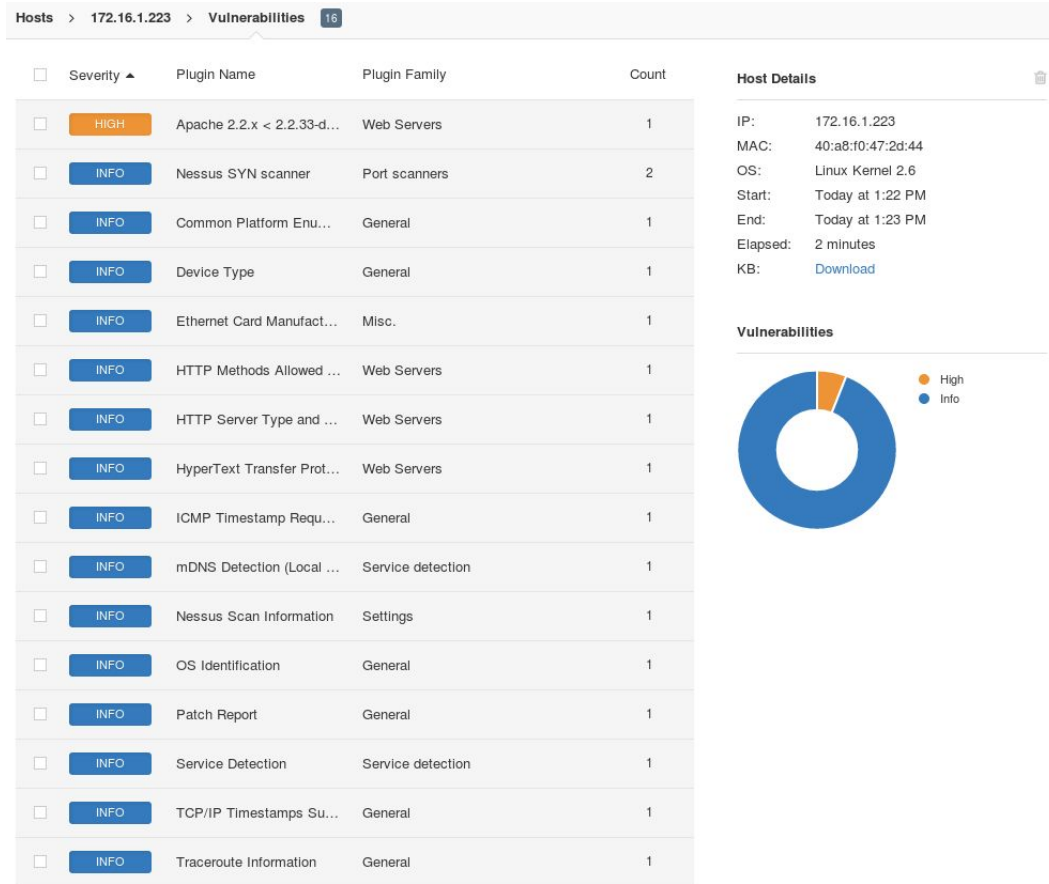
Host Details

IP: 172.16.1.223  
OS: Linux Kernel 2.6  
Start: Today at 1:18 PM  
End: Today at 1:20 PM  
Elapsed: 2 minutes  
KB: [Download](#)

Vulnerabilities

High

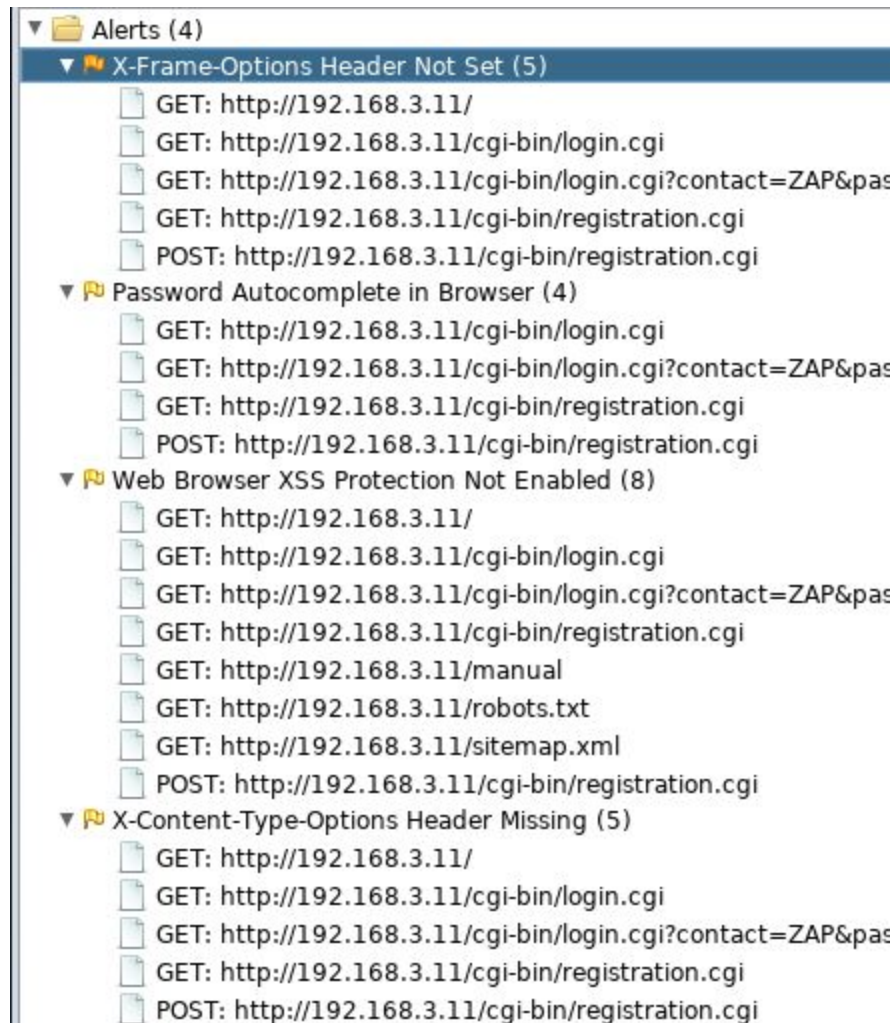
Info



Vulnerabilities report by Arachni

See attachment

Screenshots taken from OWASP ZAP, scanning web application vulnerabilities. Discovered 4 main vulnerabilities which are X-Frame options header, Password autocomplete, XSS protection not enabled and x-content-type-options header missing. OWASP ZAP provides explanation for each vulnerability and solution to fix it



## X-Frame-Options Header Not Set

### X-Frame-Options Header Not Set

URL: http://192.168.3.11/

Risk:  Medium

Confidence: Medium

Parameter: X-Frame-Options

Attack:

Evidence:

CWE ID: 16

WASC ID: 15

Source: Passive

#### Description:

X-Frame-Options header is not included in the HTTP response to protect against 'Clickjacking' attacks.

#### Other Info:


#### Solution:

Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).

## Password Autocomplete in Browser

### Password Autocomplete in Browser

URL: http://192.168.3.11/cgi-bin/login.cgi

Risk:  Low

Confidence: Medium

Parameter: password

Attack:

Evidence: <input type="password" placeholder="Enter Password" name="password" value="" required>

CWE ID: 525

WASC ID: 15

Source: Passive

#### Description:

The AUTOCOMPLETE attribute is not disabled on an HTML FORM/INPUT element containing password type input. Passwords may be stored in browsers and retrieved.

#### Other Info:

#### Solution:


Turn off the AUTOCOMPLETE attribute in forms or individual input elements containing password inputs by using AUTOCOMPLETE='OFF'.



## Web Browser XSS Protection Not Enabled

### Web Browser XSS Protection Not Enabled

URL: http://192.168.3.11/

Risk:  Low

Confidence: Medium

Parameter: X-XSS-Protection

Attack:

Evidence:

CWE ID: 933

WASC ID: 14

Source: Passive

#### Description:

Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server

#### Other Info:

The X-XSS-Protection HTTP response header allows the web server to enable or disable the web browser's XSS protection mechanism. The following values would attempt to enable it:

X-XSS-Protection: 1; mode=block

X-XSS-Protection: 1; report=http://www.example.com/xss


#### Solution:

Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.

## X-Content-Type-Options Header Missing

### X-Content-Type-Options Header Missing

URL: http://192.168.3.11/

Risk:  Low

Confidence: Medium

Parameter: X-Content-Type-Options

Attack:

Evidence:

CWE ID: 16

WASC ID: 15

Source: Passive

#### Description:

The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

#### Other Info:

This issue still applies to error type pages (401, 403, 500, etc) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type.

At "High" threshold this scanner will not alert on client or server error responses.

#### Solution:

Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.

If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

## Broken authentication and session management

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities (temporarily or permanently).

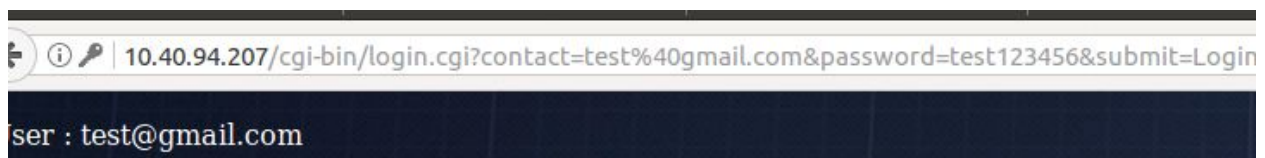
### Detailed description of each attack attempted

The easiest way to find authentication is HTML clues. Some HTML editor put their password in html page just for convenient. In this case, we just right click Login and find Inspect Element, and we can find some login username and password.

```
<td>
  <input name="submit" value="Login" type="submit">
  <!--TEST test@gmail.com:test123456-->
  <!--This is the test account-->
  <a href="/cgi-bin/registration.cgi" style="color:white;">Registration</a>
</td>
</tr>
</tbody>
```

On the other hands, Tamper Data will be used on login page to get input name of email and password which is needed later on for dictionary attack using crack\_web\_form.pl. Crack\_web\_form.pl was written by computersecuritystudent.com which is used to discover login password by launching dictionary attack from a password dictionary file.

Besides, once we successfully login the web, the login username and password will be shown on URL. If user just send the url to their friends, other guys will know their user credential.



The screenshot shows a web browser address bar with the URL: 10.40.94.207/cgi-bin/login.cgi?contact=test%40gmail.com&password=test123456&submit=Login. Below the address bar, a dark blue box displays the text: ser : test@gmail.com.

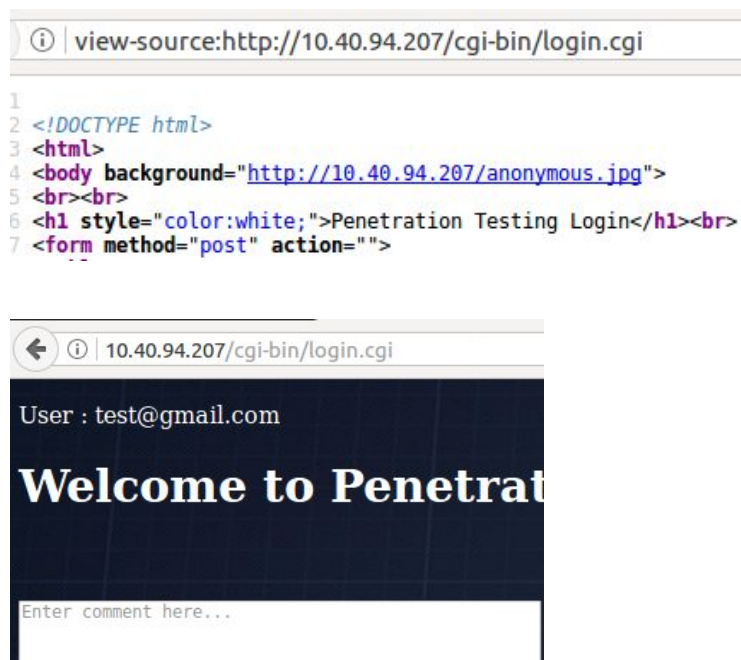
## Result

Crack\_web\_form.pl successfully found password that can be used to login to the test email account. Surprisingly, we discovered all registered email will be able to login by using any password that more than 8 character such as 000000000, abcd1234 and etc

## Remediation

The web application does not compare login password to the password that stored in database. Web application will show the main page directly with any password that more than 8. For remediation for this vulnerability, compare email and password to the database and then redirect to main page.

To prevent password showing on url, we need to change the html page. We use “GET” method on form, which will append the form variables to the URL. If we change to “POST” method, such sensitive data will be hidden from URL.



## Cross Site Scripting XSS

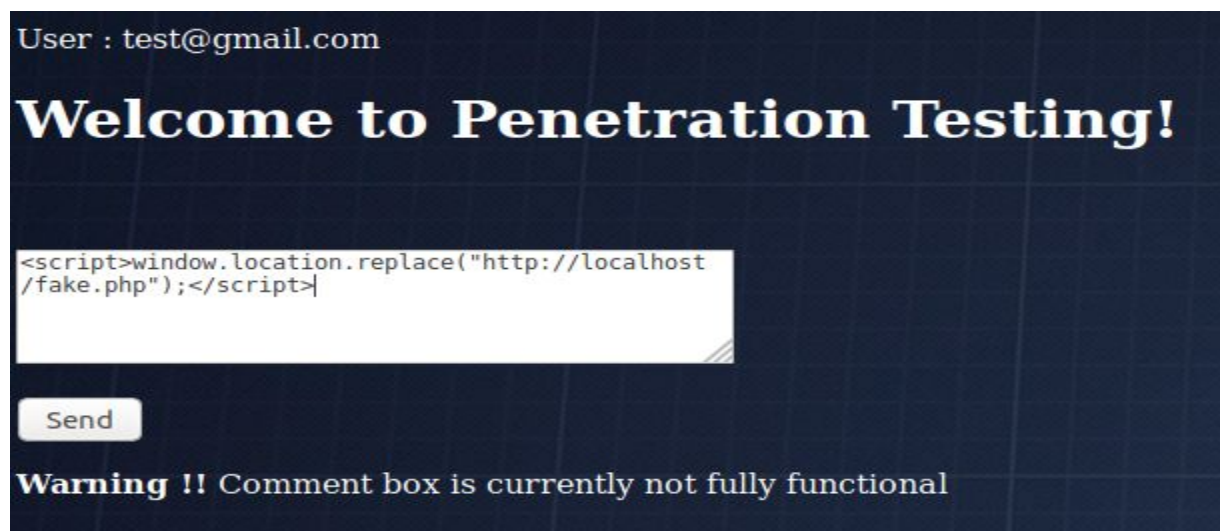
XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user supplied data using a browser API that can create JavaScript. XSS allows attackers to run some malicious scripts in the victim's browser. This scripts can be a easy alter windows for trick. But it can also get sensitive information or doing some malicious execution like hijacking user sessions, defacing web sites, or redirecting the user to malicious sites.

### Detailed description of each attack attempted

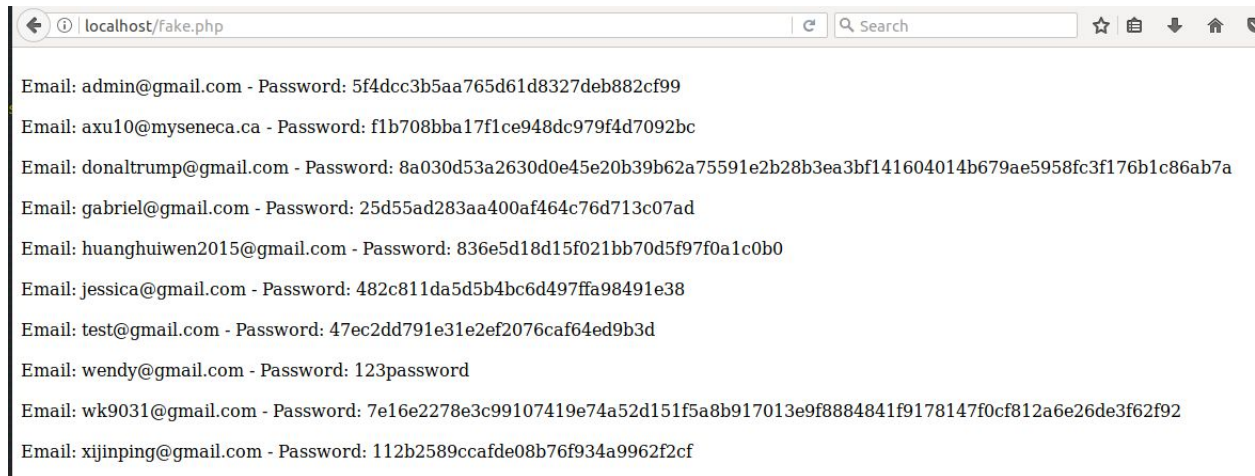
Create a Python file that to connect MySQL database and extract all email and password. Insert script that redirect to fake python file for retrieve usernames and passwords. However, this vulnerability test is inserting localhost file instead of sending a fake file to the server and execute it

### Result

Successfully reveal all email and password from database by injecting fake webpage to the server after clicking "Send"



Inject script on web application to redirect to fake.php which located in the server



Fake.php will connect to the server and retrieve all emails and passwords on the screen. Also, Fake.php is able to retrieve any data from database by editing code in the file

## Remediation

Create regular expression may remediate the vulnerable by rejecting any comment that consist "<script>"

## Injection

Injection flaws, such as SQL, OS, XXE, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization

## Detailed description of each attack attempted

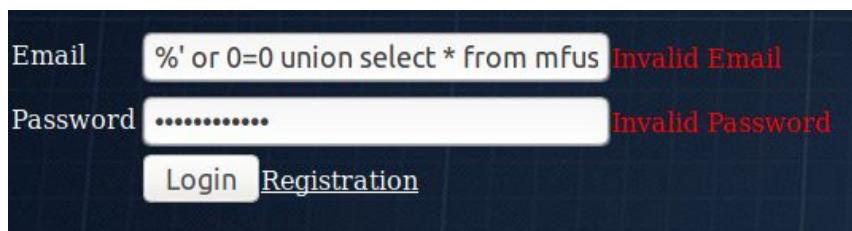
In this case, attacks will inject malicious SQL code into login page to reveal sensitive data when logging into user account

## Result

However, the attack attempt is not success. because we have already applied regular expression into our login page and registration page. The regular expression would check user input and only when the input meet its requirement and the system can pass authentication.



The screenshot shows a login form with the title "Penetration Testing Login". The "Email" field contains "test@gmail.com". The "Password" field contains "test12345' ; or 1=1 #". To the right of the password field, the text "Invalid Password" is displayed in red. Below the password field are two buttons: "Login" and "Registration".



The screenshot shows the same login form. The "Email" field contains the SQL injection payload "%' or 0=0 union select \* from mfus". To the right of the email field, the text "Invalid Email" is displayed in red. The "Password" field contains ".....". To the right of the password field, the text "Invalid Password" is displayed in red. The "Login" and "Registration" buttons are still present.

## Remediation

SQL Injection can be avoided by using regular expression which filtered input data before sending to the web server. Regular expression helps validate input data format such as phone number only allow number and no other characters or first name only allows alphabets and no any number



# Penetration Testing Registration

First name	<input type="text" value="123123123"/>	Invalid first name
Last name	<input type="text" value="anonymous"/>	
Email	<input type="text"/>	Invalid email or mobile phone
Re-enter email	<input type="text"/>	Does not match with email
Password	<input type="password"/>	Invalid password
Birthday	<input type="text" value="Month"/> <input type="text" value="Day"/> <input type="text" value="Year"/>	Invalid birthday
	<input type="button" value="Send"/> <input type="button" value="Reset"/> <a href="#">Login</a>	

## Sensitive Data Exposure

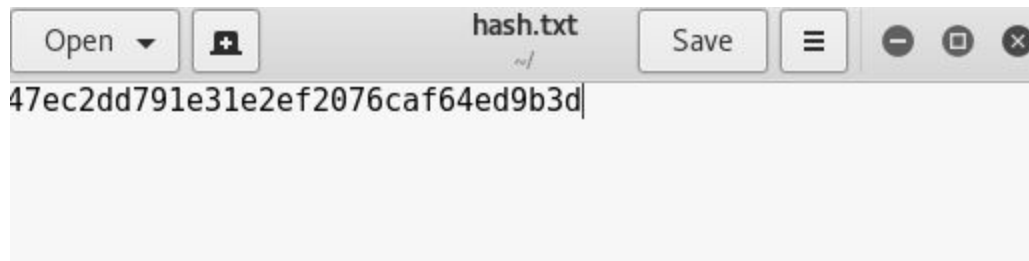
Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser

### Detailed description of each attack attempted

In our web application, we only use md5 hash to encrypt user's password. Actually, it is a really dangerous action. As we get the hash output from server database. We can use john the ripple tools to crack the hashed md5 password.

## Result

Just take test user as example. As we known in previous part, the hash value of test user is **47ec2dd791e31e2ef2076caf64ed9b3d**, we type it in a text file hash.txt



## Remediation

Although password are stored in hash value. But the weak hash algorithm cannot 100% protect the sensitive data. In order to secure the password, we can use more advanced hash algorithm such as sha 256, which make it harder to cracked. Meanwhile, we can apply another encryption based on hashed password. In addition, we can also add salt on hash algorithm.

## Insufficient Attack Protection

The majority of applications and APIs lack the basic ability to detect, prevent, and respond to both manual and automated attacks. Attack protection goes far beyond basic input validation and involves automatically detecting, logging, responding, and even blocking exploit attempts. Application owners also need to be able to deploy patches quickly to protect against attacks.

### Detailed description of each attack attempted

User's password can also be known by some simple action. If we open Inspect Element on password window and change the type from **password** to **text**, and the password will be shown in plaintext. Actually, we also trying this action on both Seneca Blackboard and Seneca Moodle, and getting the same output.



## Result

# Penetration Testing Login

Email

Password



This connection is not secure.  
Logins entered here could be compromised. [Learn More](#)

```
<tr></tr>
<tr></tr>
<tr>
  <td style="color:white;">Password</td>
  <td>
    <input placeholder="Enter Password" name="password" value="" required="" type="password">
    <font color="red"></font>
  </td>
</tr>
<tr></tr>
</tbody>
```

Password

[Registration](#)

**Login Here**

 [Change Text Size](#)  [High Contrast Setting](#)

You are not logged in

Seneca Students, Applicants, and Employees -  
Welcome to My.Seneca, your one-stop portal  
to Seneca College information, resources, and  
services.

[Reset/Forgot your password](#)

USERNAME

PASSWORD

**Returning to this web site?**

Login here using your username and password  
(Cookies must be enabled in your browser) ?

Username

Password

Some courses may allow guest access

Forgotten your username or password?

(This is random input, not the real password)

Just like previous picture, the normal password inputting authentication can be easily exploited by provisionally changing the password type. For convenience, many users allow the browser to remember the username and password. But it is possible for

hackers, perhaps just as their classmates or friends, to physically close victim machine to try malicious attempt.

## **Remediation**

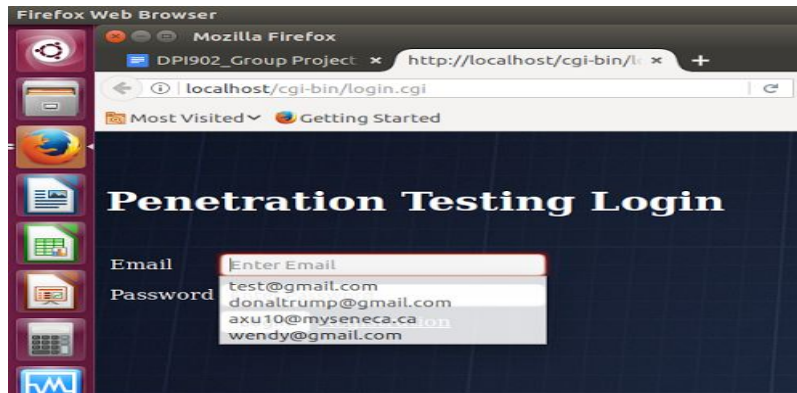
Actually, this vulnerability is the common issue for current web login. So I notice that many social media promote QR code scanning rather than username password login to avoid such attack.

## **Server Side Scripts**

By not providing autoComplete=off to the fields in the form, values that can be sensitive in their nature, for example credit card numbers, password, etc may be cached and saved by the browser accessing the site. This could lead to its compromise or re-usage without the user's consent or approval

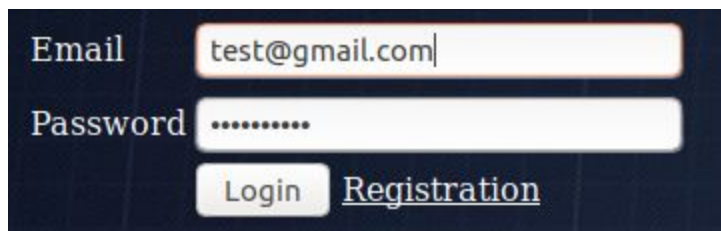
### **Detailed description of each attack attempted**

Visit login page and click on email textbox which will automatically show a drop down menu with entered emails.



## Result

When email and password is entered in a form and the form is sent, web browser asks if the password should be saved. Thereafter when the form is displayed, the email and password are filled in automatically or are completed as the name is entered. An attacker with local access could obtain the cleartext password from the browser cache.



## Remediation

Turn off autocomplete in web page which used for sensitive information such as passwords. This accomplishes for a single field by modifying the HTML source and adding the the following line `<input name="password" type="password" autocomplete="off">`

## Summary Comments

Our group developed a web application which contains a registration system that allow user to register an account and able to login with the account to leave comment message. We uses several tools to scan vulnerabilities of web application and web server including OWASP ZAP, Nessus, Nmap and Arachni.

We discovered web application may exploit by 5 web vulnerabilities from OWASP Top 10 such as cross site scripting, server side scripting, broken authentication and session management and etc. We applied some remediation to the web application to avoid attacker after discovered the vulnerabilities. Furthermore, our web application is written in Python and inserted some methods to improve security. The methods are password encryption which encrypt all registered password before storing to MySQL database and regular express to avoid SQL injection which only accept in required format like characters or numbers.

In conclusion, we have learnt web security and web hacking in this group project about common web vulnerabilities and how to improve web security. This project caught our attention to penetration testing and would like to go further in this field.