	CSE 482L: Internet and Web Technology Lab	Lab Manual
	Faculty: naqib.hussain@northsouth.edu Instructor: shabbir.ahmed01@northsouth.edu Updated: 2023-Oct-25	10

Goal:

- SQL Injection, Cross Site Scripting, MD5, Sha
- Private/Public key, TLS and http2 on XAMPP
- Virtual Host enabling on XAMPP
- Set up a custom domain name on your localhost
- Set up SSL/HTTPS in your local server

SQL Injection

SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

You can detect SQL injection manually using a systematic set of tests against every entry point in the application. To do this, you would typically submit:

- The single quote character ' and look for errors or other anomalies.
- Some SQL-specific syntax that evaluates to the base (original) value of the entry point, and to a different value, and look for systematic differences in the application responses.
- Boolean conditions such as `OR 1=1` and `OR 1=2`, and look for differences in the application's responses.
- Payloads designed to trigger time delays when executed within a SQL query, and look for differences in the time taken to respond.

SQL injection examples

There are lots of SQL injection vulnerabilities, attacks, and techniques that occur in different situations. Some common SQL injection examples include:

• Retrieving hidden data, where you can modify a SQL query to return additional results.

• Subverting application logic, where you can change a query to interfere with the application's logic.

• UNION attacks, where you can retrieve data from different database tables.

• Blind SQL injection, where the results of a query you control are not returned in the application's responses.

Retrieving hidden data

Imagine a shopping application that displays products in different categories. When the user clicks on the Gifts category, their browser requests the URL:

```
https://insecure-website.com/products?category=Gifts
```

This causes the application to make a SQL query to retrieve details of the relevant products from the database:

```
SELECT * FROM products WHERE category = 'Gifts' AND released = 1
```

This SQL query asks the database to return: all details (*) from the products table, where the category is `Gifts` and released is 1.

The restriction `released = 1` is being used to hide products that are not released. We could assume for unreleased products, `released = 0`.

The application doesn't implement any defenses against SQL injection attacks. This means an attacker can construct the following attack, for example:

```
https://insecure-website.com/products?category=Gifts'--
```

This results in the SQL query:

```
SELECT * FROM products WHERE category = 'Gifts'--' AND released = 1
```

Note that, `--` is a comment indicator in SQL. This means that the rest of the query is interpreted as a comment, effectively removing it. In this example, this means the query no longer includes AND `released = 1`. As a result, all products are displayed, including those that are not yet released.

You can use a similar attack to cause the application to display all the products in any category, including categories that they don't know about:

```
https://insecure-website.com/products?category=Gifts'+OR+1=1--
```

This results in the SQL query:

```
SELECT * FROM products WHERE category = 'Gifts' OR 1=1--' AND released=1
```

The modified query returns all items where either the category is `Gifts`, or `1 is equal to 1`. As `1=1` is always true, the query returns all items.

Subverting application logic

Imagine an application that lets users log in with a username and password. If a user submits the username `wiener` and the password `bluecheese`, the application checks the credentials by performing the following SQL query:

```
SELECT * FROM users WHERE username = 'wiener' AND password = 'bluecheese'
```

If the query returns the details of a user, then the login is successful. Otherwise, it is rejected.

In this case, an attacker can log in as any user without the need for a password. They can do this using the SQL comment sequence -- to remove the password check from the WHERE clause of the query. For example, submitting the username `administrator'--` and a blank password results in the following query:

```
SELECT * FROM users WHERE username = 'administrator'--' AND password = ''
```

This query returns the user whose username is administrator and successfully logs the attacker in as that user.

Retrieving data from other database tables

In cases where the application responds with the results of a SQL query, an attacker can use a SQL injection vulnerability to retrieve data from other tables within the database. You can use the `UNION` keyword to execute an additional SELECT query and append the results to the original query.

For example, if an application executes the following query containing the user input `Gifts`:

```
SELECT name, description FROM products WHERE category = 'Gifts'
```

An attacker can submit the input:

```
'UNION SELECT username, password FROM users--
```

This causes the application to return all usernames and passwords along with the names and descriptions of products.

How to prevent SQL injection

You can prevent most instances of SQL injection using parameterized queries instead of string concatenation within the query. These parameterized queries are also known as "prepared statements". The following code is vulnerable to SQL injection because the user input is concatenated directly into the query:

```
String query = "SELECT * FROM products WHERE category = '"+ input + "'";

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(query);
```

You can rewrite this code in a way that prevents the user input from interfering with the query structure:

```
PreparedStatement statement = connection.prepareStatement("SELECT * FROM
products WHERE category = ?");

statement.setString(1, input);

ResultSet resultSet = statement.executeQuery();
```

You can use parameterized queries for any situation where untrusted input appears as data within the query, including the WHERE clause and values in an INSERT or UPDATE statement. They can't be used to handle untrusted input in other parts of the query, such as table or column names, or the ORDER BY clause. Application functionality that places untrusted data into these parts of the query needs to take a different approach, such as:

- Whitelisting permitted input values.
- Using different logic to deliver the required behavior.

Cross Site Scripting

Cross Site Scripting (XSS) is a vulnerability in a web application that allows a third party to execute a script in the user's browser on behalf of the web application. Cross-site Scripting is one of the most prevalent vulnerabilities present on the web today. The exploitation of XSS against a user can lead to various consequences such as account compromise, account deletion, privilege escalation, malware infection and many more.

Depending on the context, there are two types of XSS –

Reflected XSS: If the input has to be provided each time to execute, such XSS is called reflected. These attacks are mostly carried out by delivering a payload directly to the victim. Victim requests a page with a request containing the payload and the payload comes embedded in the response as a script. An example of reflected XSS is XSS in the search field.

Stored XSS: When the response containing the payload is stored on the server in such a way that the script gets executed on every visit without submission of payload, then it is identified as stored XSS. An example of stored XSS is XSS in the comment thread.

DOM based XSS: and its instances are either reflected or stored. DOM-based XSS arises when user-supplied data is provided to the DOM objects without proper sanitizing. An example of code vulnerable to XSS is below, notice the variables `firstname` and `lastname`:

```
<?php
   if(isset($_GET["firstname"]) && isset($_GET["lastname"]))
   {
        $firstname = $_GET["firstname"];
         $lastname = $_GET["lastname"];
        if($firstname == "" or $lastname == "")
        {
            echo "<font color=\"red\">Please enter both fields...</font>";
        }
        else
        {
            echo "Welcome " . $firstname. " " . $lastname;
        }
   } ?>
```

User-supplied input is directly added in the response without any sanity check. Attacker an input something like:

```
<script> alert(1) </script>
```

And it will be rendered as JavaScript.

Cross-site request forgery (CSRF):

CSRF (sometimes also called XSRF) is a related class of attack. The attacker causes the user's browser to perform a request to the website's backend without the user's consent or knowledge. An attacker can use an XSS payload to launch a CSRF attack. Wikipedia mentions a good example for CSRF. In this situation, someone includes an image that isn't really an image (for example in an unfiltered chat or forum), instead it really is a request to your bank's server to withdraw money:

```
<img
src="https://bank.example.com/withdraw?account=bob&amount=1000000&for=mall
ory" />
```

Now, if you are logged into your bank account and your cookies are still valid (and there is no other validation), you will transfer money as soon as you load the HTML that contains this image. For endpoints that require a POST request, it's possible to programmatically trigger a `<form>` submit (perhaps in an invisible `<iframe>`) when the page is loaded:

```
<form action="https://bank.example.com/withdraw" method="POST">

  <input type="hidden" name="account" value="bob" />

  <input type="hidden" name="amount" value="1000000" />

  <input type="hidden" name="for" value="mallory" />

</form>

<script>

  window.addEventListener("DOMContentLoaded", () => {

    document.querySelector("form").submit();

  });

</script>
```

Set up a custom domain

1. Open the `Notepad` as an administrator
2. Open the directory in file explorer: `C:\Windows\System32\drivers\etc.`
3. There you will find a file called `hosts`. Double click on the file to open it with `Notepad`
4. Scroll to the end of the file and add the following line: `127.0.0.1 mylab.test`.
5. Click Save to overwrite the changes to the file.

6. You should replace `mylab.test` with **your preferred domain name**

Set up ssl/https in your local server

Generate a CSR code:

Method 1:

The Certificate Signing Request, or simply CSR, is a small, encoded text file containing information about your domain and/or company. All commercial CAs require SSL applicants to submit a CSR code, as part of the SSL validation process.

You can use the following steps to generate a private key with puttygen:

1. Download and Open puttygen
2. select RSA as the type of key to generate.
3. Enter 2048 as the number of bits in the generated key.
4. Click the Generate button and move your mouse randomly over the blank area to create some randomness.
5. Once the key is generated, you can optionally enter a passphrase to protect it.
6. Click the Save private key button and choose a location and name for your private key file. For example: private.key.

You can use the following steps to generate a CSR file with the openssl command:

7. Open a command prompt and navigate to the Apache directory in XAMPP.
   `C:\xampp\apache.`
8. Run the following command to create a CSR file using your private key and the configuration file for openssl. Replace private.key with the name and location of your private key file, and request .csr with the name and location of your desired CSR file.
   `C:\xampp\apache\ssl\request.csr.`
   ```
   openssl req -new -key private.key -config
   "C:\xampp\php\extras\openssl" -out request.csr
   ```
9. You will be prompted to enter some information about your domain and company, such as country name, state or province name, locality name, organization name, organizational unit name, common name, and email address. You can also leave some fields blank by pressing

Enter.

10. Once you have entered all the required information, your CSR file will be created and ready to use.

Method 2:

1. Navigate to Apache directory in xampp
   `C:\xampp\apache`
2. Create a new crt folder, here we will store our ssl files
   `C:\xampp\apache\crt`
3. Create these two files from below *(you may download these files from end notes c)*
   `cert.txt`
   `make-cert.txt`
4. Edit the `cert.txt` file with your domain name and change both of the file's extensions as prescribed

File name: `cert.txt`	**New file name:** `cert.conf`

```
[ req ]
default_bits        = 2048
default_keyfile     = server-key.pem
distinguished_name  = subject
req_extensions      = req_ext
x509_extensions     = x509_ext
string_mask         = utf8only

[ subject ]
countryName                 = Country Name (2 letter code)
countryName_default         = US

stateOrProvinceName         = State or Province Name (full name)
stateOrProvinceName_default = NY

localityName                = Locality Name (eg, city)
localityName_default        = New York
```

```
organizationName            = Organization Name (eg, company)
organizationName_default    = Example, LLC

commonName                  = Common Name (e.g. server FQDN or YOUR name)
commonName_default          = 482lab.test

emailAddress                = Email Address
emailAddress_default        = test@example.com

[ x509_ext ]
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid,issuer

basicConstraints       = CA:FALSE
keyUsage               = digitalSignature, keyEncipherment
subjectAltName         = @alternate_names
nsComment              = "OpenSSL Generated Certificate"

[ req_ext ]
subjectKeyIdentifier = hash
basicConstraints     = CA:FALSE
keyUsage             = digitalSignature, keyEncipherment
subjectAltName       = @alternate_names
nsComment            = "OpenSSL Generated Certificate"

[ alternate_names ]
DNS.1       = 482lab.test
```

File name: make-cert.txt	**New file name:** make-cert.bat

```
@echo off
set /p domain="Enter Domain: "
set OPENSSL_CONF=../conf/openssl.cnf
```

```
if not exist .\%domain% mkdir .\%domain%
..\bin\openssl req -config cert.conf -new -sha256 -newkey rsa:2048 -nodes -
keyout %domain%\server.key -x509 -days 365 -out %domain%\server.crt
echo.
echo -----
echo The certificate was provided.
echo
pause
```

5. Double click the `make-cert.bat` and input your domain when prompted. And just do `enter` in other question since we already set the default from `cert.conf`
6. After that, you will see `482lab.test` folder created. In that folder we will have `server.crt` and `server.key`. This is our SSL certificate.
7. Double click on the server.crt to install it on Windows so Windows can trust it.
8. select `Local Machine` as Store Location.
9. And then Select `Place all certificate in the following store` and click `browse` and select `Trusted Root Certification Authorities`.
10. Click `Next` and `Finish`.

Add the site in xampp conf

We need to enable SSL for this domain and let XAMPP know where we store the SSL Cert. So we need to edit `C:\xampp\apache\conf\extra\httpd-xampp.conf`

add this code at the bottom:

```
## site.test
 <VirtualHost *:80>
     DocumentRoot "C:/xampp/htdocs"
     ServerName 482lab.test
     ServerAlias *.482lab.test
 </VirtualHost>
 <VirtualHost *:443>
     DocumentRoot "C:/xampp/htdocs"
     ServerName 482lab.test
     ServerAlias *.482lab.test
     SSLEngine on
```

```
    SSLCertificateFile "crt/site.test/server.crt"
    SSLCertificateKeyFile "crt/site.test/server.key"
</VirtualHost>
```

After that, you will need to restart Apache in XAMPP. Now you can browse `https://482lab.test` on your browser with secure connection.

End Notes:

[a] You should replace c:/xampp/htdocs/example/ with the location of your project. For example, if your project is in a subfolder, then add the subfolder c:/xampp/htdocs/folder/subfolder/.

[b] It will not work with mozila firefox since firefox doesn't accept self-signed certificates

[c] How to Create Valid SSL in localhost for XAMPP – Shellcreeper.com

[d] How to Install an SSL certificate for Localhost XAMPP? | by Dinu Gitlan | Medium

[e] How to generate secure SSL certificate for localhost in XAMPP and Enable HTTPS for localhost - YouTube