SENG 4220

Software Security

Web Security and Code Injection Attacks and Defenses

Due - Mar 24, 2023 (11:59 pm)

Objectives

This assignment aims to provide students with practical knowledge in the field of software security by exploring web security threats and SQL code injection attacks. By the end of the assignment, students will have a solid understanding of web security fundamentals and the vulnerabilities that exist in web applications, as well as the vulnerabilities present in SQL databases. Furthermore, they will gain hands-on experience in identifying and mitigating these vulnerabilities to improve the security of software systems.
Please complete the sections labeled "# Your answer".

Section 1: A Simple Cross-Site Scripting (XSS) Attack

In this section, we will learn how a cross-site scripting attack (XSS) works. In particular, we will see a variety of attack scripts and learn how to defend against them. XSS attack is a powerful attack that can subvert the same-origin policy. To do so, the attacker injects malicious JavaScript onto a webpage. When the victim loads the webpage, the user’s browser will run the malicious JavaScript automatically.

Setup

To begin with, open this file using Jupyter notebook. The assignment is divided into two parts - attacking a website and attacking a database. The main objective of this assignment is to learn about software security, and as a part of this assignment, we will be using Jupyter notebook as the running environment, Flask as the web server, and SQLite for database management. The source code to build these systems is provided, and you do not need any prior web programming knowledge to complete this assignment. However, if you have any questions, the instructor is available to help you. To run this assignment, you need to install the Flask package, which is a Python framework for building web servers. You can install this package by running the following command.

```
!pip3 install flask
```

Run the following code to import the package.

```
import logging
from flask import Flask, request
from multiprocessing import Process

log = logging.getLogger('Werkzeug')
log.setLevel(logging.ERROR)

p = None
```

Below we provide a template website that simulates a simple social networking site.

```
web_template = """
<html>

<head>
    <style type="text/css">
        body {
            margin-top: 20px;
        }

        /*==================================================
              Post Contents CSS
              ==================================================*/

        .post-content {
```

```css
    background: #f8f8f8;
    border-radius: 4px;
    width: 100%;
    border: 1px solid #f1f2f2;
    margin-bottom: 20px;
    overflow: hidden;
    position: relative;
}

.post-content img.post-image,
video.post-video,
.google-maps {
    width: 100%;
    height: auto;
}

.post-content .google-maps .map {
    height: 300px;
}

.post-content .post-container {
    padding: 20px;
}

.post-content .post-container .post-detail {
    margin-left: 65px;
    position: relative;
}

.post-content .post-container .post-detail .post-text {
    line-height: 24px;
    margin: 0;
}

.post-content .post-container .post-detail .reaction {
    position: absolute;
    right: 0;
    top: 0;
}

.post-content .post-container .post-detail .post-comment {
    display: inline-flex;
    margin: 10px auto;
    width: 100%;
}

.post-content .post-container .post-detail .post-comment img.profile-photo-sm {
    margin-right: 10px;
}

.post-content .post-container .post-detail .post-comment .form-control {
    height: 30px;
    border: 1px solid #ccc;
    box-shadow: inset 0 1px 1px rgba(0, 0, 0, .075);
    margin: 7px 0;
    min-width: 0;
}

img.profile-photo-md {
    height: 50px;
    width: 50px;
    border-radius: 50%;
}

img.profile-photo-sm {
    height: 40px;
    width: 40px;
    border-radius: 50%;
}

.text-green {
    color: #8dc63f;
}

.text-red {
    color: #ef4136;
}
```

```
        .following {
            color: #8dc63f;
            font-size: 12px;
            margin-left: 20px;
        }
    </style>
</head>

<body>
    <div class="container">
        <div class="row">
            <div class="col-md-8">
                <div class="post-content">
                    <div class="post-container">
                        <img src="https://bootdey.com/img/Content/avatar/avatar6.png" alt="user"
                            class="profile-photo-md pull-left">
                        <div class="post-detail">
                            <div class="user-info">
                                <h5><a href="timeline.html" class="profile-link">Domenick Behnke</a> <span
                                        class="following">following</span></h5>
                                <p class="text-muted">Published a post about 3 mins ago</p>
                            </div>
                            <div class="reaction">
                                <a class="btn text-green"><i class="fa fa-thumbs-up"></i> 13</a>
                                <a class="btn text-red"><i class="fa fa-thumbs-down"></i> 0</a>
                            </div>
                            <div class="line-divider"></div>
                            <div class="post-text">
                                <p> I got an A in SENG 4220! <i class="em em-anguished"></i> <i class="em em-anguished"></i>
                                    <i class="em em-anguished"></i></p>
                            </div>
                            <div class="line-divider"></div>
                            <div class="post-comment">
                                <img src="https://bootdey.com/img/Content/avatar/avatar7.png" alt=""
                                    class="profile-photo-sm">
                                <p><a href="timeline.html" class="profile-link">Brendan</a><i class="em em-laughing"></i>
                                    Congratulations! </p>
                            </div>
                            <div class="post-comment">
                                <img src="https://bootdey.com/img/Content/avatar/avatar5.png" alt=""
                                    class="profile-photo-sm">
                                <p><a href="timeline.html" class="profile-link">Trevor</a> Congratulations! </p>
                            </div>
                            <div class="post-comment">
                                <img src="https://bootdey.com/img/Content/avatar/avatar3.png" alt=""
                                    class="profile-photo-sm">
                                <input type="text" class="form-control" placeholder="Post a comment">
                            </div>
                        </div>
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>

</html>
"""
```

Below is some code that will spawn an HTTP server on your local computer and dump the request data. Run the code block and navigate to (http://localhost:5000) in your browser of choice to see our website. Note that this is a static website, so you cannot interact with it directly.

```
app = Flask('SENG 4220 Social Network Post Demo')
web = web_template

@app.route('/', defaults={'path': ''})
@app.route('/<path:path>')
def dump_response(path):
    print("%s %s %s\n%s" % (
        request.method,
        request.environ['RAW_URI'],
        request.environ['SERVER_PROTOCOL'],
        request.headers
    ), end='')
    data = request.get_data()
    if data:
        print("%s" % data.decode('utf-8'))
    return web

if p:
    p.terminate()
    p.join()
p = Process(target=app.run(host="0.0.0.0", port=5000))
p.start()
```

Below is a provided function to post comments in the above website. Take a moment and read the code.

```
def post(comment):
    (w1, w2, w3) = web.partition("<input type=\"text\" class=\"form-control\" placeholder=\"Post a comment\">");
    tmp = w1 + "\n<p><a href=\"timeline.html\" class=\"profile-link\">Mikhayla</a> " + comment + " </p>\n" + """
    </div>
    <div class="post-comment">
        <img src="https://bootdey.com/img/Content/avatar/avatar3.png" alt="" class="profile-photo-sm">
    """ + w2 + w3;
    return tmp;
```

The `partition()` method splits a string into three parts based on the given separator. It returns a tuple containing the part before the separator, the separator itself, and the part after the separator.

For example, `"hello world".partition(" ")` will return `("hello", " ", "world")`.

Warmup

1. Provide a comment below and see how it shows up on the website. Run the following two blocks and refresh the website (http://localhost:5000).

```
comment = "" # Your answer (1. 2 marks)
```

```
# Post your comment and refresh the website (http://localhost:5000).
web = web_template;
web = post(comment);

if p:
    p.terminate()
    p.join()
p = Process(target=app.run(host="0.0.0.0", port=5000))
p.start()
```

2. Construct a malicious input to comment that will inject an Javascript script of your choice.

```
comment = # Your answer (2 - 8 marks) (Hint: Use the script tag and the alert() method in JavaScript.)
```

```
# Post your comment and refresh the website (http://localhost:5000).
web = web_template;
web = post(comment);

if p:
    p.terminate()
    p.join()
p = Process(target=app.run(host="0.0.0.0", port=5000))
p.start()
```

Practice

An important defense against XSS attacks is input sanitization. In this lab, we provide several attack scripts, and you will need to implement an sanitizer against them. Note that your sanitizer should allow normal comments to pass.

Implement a sanitizer

This part is the most important part of your assignment. Implement a sanitizer against the following XSS attack scripts. Note that your sanitizer should make all the malicious input fail but allow the honest input to be posted normally (which means you should not just output an empty string). You should think of the scenario where your sanitizer is used in a real-world website.

```
def sanitizer(comment):
    # Your answer (3 - Avoid each attack 15 marks (60 marks in total))
    return comment;
```

1. Normal input

```
normal_comment = "<b>Here is my script.</b><br><span> Nobody told me I needed to bring my Web programming skills to SENG 4220! <
```

```
# Post your comment and refresh the website (http://localhost:5000).
web = web_template;
comment = sanitizer(normal_comment);
web = post(comment);

if p:
    p.terminate()
    p.join()
p = Process(target=app.run(host="0.0.0.0", port=5000))
p.start()
```

2. XSS attack script A.
   Note. Your sanitizer function must prevent this type of attack from occurring.

```
attack_comment = "<script>alert(\"XSS attack!\")</script>" # attack comment
```

```
# Post your comment and refresh the website (http://localhost:5000).
web = web_template;
comment = sanitizer(attack_comment);
web = post(comment);

if p:
    p.terminate()
    p.join()
p = Process(target=app.run(host="0.0.0.0", port=5000))
p.start()
```

3. XSS attack script B.
   Note. Your sanitizer function also must prevent this type of attack from occurring.

```
attack_comment = '<scr<script>ipt>alert("XSS attack!")</scr<script>ipt>' # attack comment
```

```
# Post your comment and refresh the website (http://localhost:5000).
web = web_template;
comment = sanitizer(attack_comment);
web = post(comment);

if p:
    p.terminate()
    p.join()
p = Process(target=app.run(host="0.0.0.0", port=5000))
p.start()
```

4. XSS attack script C.
   Note. Your sanitizer function also must prevent this type of attack from occurring.

```
attack_comment = '<b onmouseover=alert(\'Attack!\')>click me!</b>' # attack comment
```

```
# Post your comment and refresh the website (http://localhost:5000).
# To trigger the attack, you need to click the button.
web = web_template;
comment = sanitizer(attack_comment);
web = post(comment);
```

```
if p:
    p.terminate()
    p.join()
p = Process(target=app.run(host="0.0.0.0", port=5000))
p.start()
```

5. XSS attack script D.

```
attack_comment = '\x3cscript>alert("XSS attack!")\x3c/script>' # attack comment
```

```
# Post your comment and refresh the website (http://localhost:5000).
web = web_template;
comment = sanitizer(attack_comment);
web = post(comment);

if p:
    p.terminate()
    p.join()
p = Process(target=app.run(host="0.0.0.0", port=5000))
p.start()
```

Finish

Run the following block of code to stop the server once you are finished.

```
p.terminate()
p.join()
```

Section 2: A Simple SQL Injection Attack

In this section, we will learn how to perform a SQL injection attack. SQL injection is a subset of code injection attacks, a class of attacks whose core idea is to convince the server to interpret an input as part of a SQL query. If the application is creating SQL strings by integrating user's input on the fly and then running them, we will see that it's straightforward to inject malicious SQL code. SQL injection attacks allow attackers to spoof one's identity or tamper with existing data.

Setup

We first introduce how to build a database in python. **Sqlite3** is a python module that provides a SQL interface to interact with a lightweight disk-based database. We will use this module to create the database and execute our SQL command.

```
import sqlite3
```

To use the module, we must first create a *Connection* object that represents the database. Usually, the data will be stored in the local file (for example, example.db). In this section, we use a special name :memory: to create a database in the memory. Once we have establish a *Connection* for the database, we can create a *Cursor* object and use it to execute SQL commands to insert data.

```
# Create database
# con = sqlite3.connect('example.db')
con = sqlite3.connect(':memory:')
# Create the Cursor
cur = con.cursor()
```

Now we call the execute() function to create two tables: the **transcripts** table that stores the grades of all the students, and the **users** table that stores passwords of all users.

```
# Create tables
cur.execute('''CREATE TABLE transcripts
               (studentID integer, name text, course text, grade text)''')
cur.execute('''CREATE TABLE users
               (username text, password text)''')
```

Below, we provide a function to execute multiple SQL commands in a single script. Using this function, we can insert data to both tables.

```
def exe_sql_script(query):
    (q1, q2, q3) = query.partition(';');
    res = [];
    while q2 != '':
        print(q1);
        try:
            for row in cur.execute(q1):
                res.append(row)
        except:
            print("Failed to execute the sql script!")
            return []
        (q1, q2, q3) = q3.partition(';');
    print(q1);
    try:
        for row in cur.execute(q1):
                res.append(row)
    except:
        print("Failed to execute the sql script!")
        return []
    return res


exe_sql_script("""
    INSERT INTO transcripts VALUES (1, 'Alfredo', 'SENG4220', 'A+');
    INSERT INTO transcripts VALUES (1, 'Alfredo', 'SENG4640', 'A-');
    INSERT INTO transcripts VALUES (2, 'Kyle', 'SENG4220', 'A-');
    INSERT INTO transcripts VALUES (2, 'Kyle', 'SENG3130', 'A');
    INSERT INTO transcripts VALUES (3, 'Brendan', 'SENG3130', 'B-');
    INSERT INTO transcripts VALUES (4, 'Brendan', 'SENG4220', 'B');
    INSERT INTO transcripts VALUES (4, 'Brendan', 'SENG3130', 'A');
    INSERT INTO transcripts VALUES (5, 'Adithya', 'SENG2110', 'A');
    INSERT INTO transcripts VALUES (6, 'Adithya', 'SENG3130', 'A');
    INSERT INTO transcripts VALUES (7, 'Ahana', 'SENG3130', 'C-');
    INSERT INTO transcripts VALUES (8, 'Domenick', 'SENG4220', 'A');
    INSERT INTO transcripts VALUES (8, 'Domenick', 'SENG3130', 'B');
    INSERT INTO transcripts VALUES (9, 'Trevor', 'SENG4220', 'A-');
    INSERT INTO transcripts VALUES (10, 'Kyle', 'SENG4640', 'B');
    INSERT INTO transcripts VALUES (11, 'Ahana', 'SENG4220', 'A');
    INSERT INTO transcripts VALUES (11, 'Ahana', 'SENG4640', 'A-');
    INSERT INTO transcripts VALUES (12, 'Trevor', 'SENG4640', 'B');

    INSERT INTO users VALUES ('Sina', '83@Ln^Yck_R@z2x#');
    INSERT INTO users VALUES ('manager', 'LbJHaRRm+98tF5P$');
    INSERT INTO users VALUES ('admin', 'cLzvCw9kCxYg?AdE');
    """);
```

Now we can collect data from the tables using the Select SQL command. Below, we select the records from the transcript table in the order of grades.

```
res = exe_sql_script('SELECT * FROM transcripts ORDER BY grade');
print(res)
```

Warmup:

1. Please provide a query that selects all records from the transcript table, ordered by **course**.

```
query = ''; # Your answer (1 - 4 marks)
```

```
# Test your answer
res = exe_sql_script(query);
print(res)
```

2. Please provide a query that selects all **grades** from the transcript table where the course is **SENG4220**.

```
query = ""; # Your answer (2 - 4 marks)
```

```
# Test your answer
res = exe_sql_script(query);
print(res)
```

Practice

In the above examples, users can provide the full query string for accessing the database. In the most of real-world applications, however, users usually can only provide a small part of query string, which will be concatenated with other strings provided by the application. However, this still allows users to perform an SQL injection attack.

In the following questions, you are asked to fill in the `input` variable, which will be used to construct the `query` string later.

Question 1.

Please fill in the `input` variable so that the SQL command will select all the records in the transcripts table where the course is **SENG4220**. You can execute the following code block to see if your answer is correct.

```
input = '' # Your answer (3 - 2 marks)
```

```
# Test your answer
query = "SELECT * FROM transcripts WHERE course = '" + input + "'";
res = exe_sql_script(query);
print(res)
```

Question 2.

Now we can see how SQL injection works. It seems users can only specify the course name; however, some input strings may result in undesirable SQL commands.

Please fill in the `input` variable so that the SQL command will select the **password** of **admin** from the **users** table. You can execute the following codes to see if your answer is correct.

```
input = '' # Your answer (4 - 10 marks)
```

```
# Test your answer
query = "SELECT * FROM transcripts WHERE course = '" + input + "'";
res = exe_sql_script(query);
print(res)
```

Question 3.

Consider the following login function. Given the username and password, it searches the database for any matching records. If (`len(select_res) > 0`), the login attempt will succeed; otherwise, the login attempt will fail.

```
def login(username, password):
    try:
        select_res = exe_sql_script("SELECT username FROM users WHERE username = '" + username + "' AND password = '" + passworc
    except:
        print("Failed to execute the sql script!")
    if len(select_res) > 0:
        print("Login Successfully!")
        print("user: " + username)
    else:
        print("Login Failed!")
```

Please provide values for the `username` and `password` variables to login as **admin**. You can find out the password of admin in the setup script or question 2. Execute the following `login` call to see the result.

```
username = "" # Your answer (5 - 1 marks)
password = "" # Your answer (6 - 1 marks)
```

```
# Test your answer
login(username, password)
```

Question 4.

Suppose the attacker is not a user of this database and does not know the password of any users in the system. Please fill in the `username` and `password` variables to login without using passwords of any existing users in the setup script.

```
username = "" # Your answer (7 - 4 marks)
password = "" # Your answer (8 - 4 marks)


# Test your answer
login(username, password)
```

Question 5.

Suppose the attacker wants to add their username and password to the database by leveraging the vulnerability in the `login` function. Please provide values for the `username` and `password` variables such that the attacker's credentials: (username) "attacker" and (password) "password123" are inserted into the users table. You can use the following `login` call to perform the injection and the second `login` call to see if the injection succeeds. Hint: the first login attempt might fail.

```
username = "" # Your answer (9 - 5 marks)
password = "" # Your answer (10 - 5 marks)


# Perform the injection
login(username, password)


# Test your anwser
login("attacker", "password123")
```

Question 6.

The application developer is trying to limit the number of SQL commands used in the login function as a potential defense. The new login function is given below. Please fill in the `username` and `password` variables to bypass this defense (Suppose the attacker does not know password of any existing user). Hint: think about how to start a comment in SQL.

```
def login_defense_attempt1(username, password):
    select_res = []
    try:
        print("SELECT username FROM users WHERE username = '" + username + "' AND password = '" + password +"'");
        for row in cur.execute("SELECT username FROM users WHERE username = '" + username + "' AND password = '" + password +"''
            select_res.append(row)
    except:
        print("Failed to execute the sql script!")
    if len(select_res) > 0:
        print("Login Successfully!")
        print("user: " + username)
    else:
        print("Login Failed!")


username = "" # Your answer (11 - 5 marks)
password = "" # Your answer (12 - 5 marks)


# Test your answer
login_defense_attempt1(username, password)
```

Question 7.

One way to defend against the attack in the question 6 is to sanitize the query string. For example, we may escape any potential input that could be used in an attack. Escaping a character means will treat characters as part of the string, not actual SQL syntax. Please implement a sanitizer in the following function to escape any potential symbol in the query string. You can test it using your answer in question 6.

```
def login_defense_attempt2(username, password):
    query = "SELECT username FROM users WHERE username = '" + username + "' AND password = '" + password +"'";
    # -----------------------
    # Your answer here (13 - 30 marks)
    # ------------------------
    select_res = []
    try:
        print(query);
        for row in cur.execute("SELECT username FROM users WHERE username = '" + username + "' AND password = '" + password +"''
            select_res.append(row)
    except:
        print("Failed to execute the sql script!")
    if len(select_res) > 0:
        print("Login Successfully!")
        print("user: " + username)
    else:
        print("Login Failed!")


username = "" # Your answer in question 6
password = "" # Your answer in question 6


# Test your answer; it is supposed to fail
login_defense_attempt2(username, password)
```

Finish

Run the following block of code to stop the database once you are finished.

```
cur.execute("DROP TABLE transcripts")
cur.execute("DROP TABLE users")
con.commit()
con.close()
```