****

**BACHELOR OF SCIENCE IN COMPUTER SCIENCE**

**BSE 3105**

**ADVANCED PROGRAMMING**

**BY**

**MR. LWOMWA JOSEPH**

**COURSEWORK**

**REPORT ON A CHAT ENGINE APPLICATION**

**GROUP DETAILS**

|  |  |  |  |
| --- | --- | --- | --- |
| **SN o.** | **Names** | **Registration Number** | **Signature** |
| 1 | **OKOT** Emmanuel |  |  |
| 2 | **ABILA** Raphael | **16/U/2673/PS** |  |
| 3 | **NAKAFEERO** Peninah | **16/U/8463/PS** |  |
| 4 | **MULINDWA** Five |  |  |

j)

Managing and prioritizing the security code review of source code components for the chat application’s threat modeling process was decomposed into 3 high level steps :

**Step 1: Decompose the Application:** The application is broken down and the composition of the application is known. This involved creating of use-cases to understand how the application is used, identifying entry points to see where a potential attacker could interact with the application for example, identifying assets i.e. items/areas that the attacker would be interested in, and identifying trust levels which represent the access rights that the application will grant to external entities. The components or assets (called threat targets) are investigated and how data flows between the components. Data Flow Diagrams (DFDs) and parts of the Unified Modeling Language (UML) are used to decompose the application into its key components. The goal of this step was concerned with gaining an understanding of the application and how it interacts with external entities.

**Step 2: Determine the threats to the system:** Categorize the threats by applying the following strategies. Using STRIDE (Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service and Elevation of privilege). This acronym aids asking questions in order to consider threats. The goal of the threat categorization was to help identify threats both from the attacker (STRIDE) and the defensive perspective (ASF). Using Threat Trees. They describe the decision-making process an attacker would go through to compromise the component. Using Attack Trees: They are conceptual diagrams showing how an asset can be attacked. These threats can be identified further as the roots for threat trees; there is one tree for threat goal. From the defensive perspective, ASF categorization helps to identify the threats as weaknesses of security controls for such threats. Common threat-lists with examples can help in the identification of such threats. Use and abuse cases can illustrate how existing protective measures could be bypassed, or where a lack of such protection exists.

**Step 3: Rank the threats by decreasing the risk/Determine countermeasures and mitigation:** After capturing the threats, determine the most important threats. The threats can be ranked by, multiplying damage potential by likelihood of vulnerability and DREAD (Damage potential, Reproducibility, Exploitability, Affected Users, Discoverability). Such countermeasures can be identified using threat-countermeasure mapping lists. Once a risk ranking is assigned to the threats, it is possible to sort threats from the highest to the lowest risk, and prioritize the mitigation effort, such as by responding to such threats by applying the identified countermeasures. The risk mitigation strategy might involve evaluating these threats from the business impact that they pose and reducing the risk. Other options might include taking the risk, assuming the business impact is acceptable because of compensating controls, informing the user of the threat, removing the risk posed by the threat completely, or the least preferable option, that is, to do nothing. Each of the above steps are documented as they are carried out. The resulting document is the threat model for the application. This guide will use an example to help explain the concepts behind threat modeling. The same example will be used throughout each of the 3 steps as a learning aid. The example that will be used is a college library website. At the end of the guide we will have produced the threat model for the college library website. Each of the steps in the threat modeling process are described in detail below.

k)

**SQL Injection**

SQL Injection is an attack where users can inject SQL commands via user input form and have them executed on the server. This SQL commands could do everything: read sensitive data, modify the database data, perform administrative tasks against the database server. Your application can be exposed to this attack when you dynamically create SQL statements (concatenating data based on user’s input).

This was implemented on lummox chat engine log in page application by entering the SQLite

commands as below.

**Cross-site request forgery**

Cross-site Scripting (XSS) refers to client-side code injection attack wherein an attacker can execute malicious scripts (also commonly referred to as a malicious payload) into a legitimate website or web application. XSS is amongst the most rampant of web application vulnerabilities and occurs when a web application makes use of unvalidated or unencoded user input within the output it generates.

By leveraging XSS, this implementation wasn’t intended target a victim directly. Instead, it will exploit a vulnerability within chat application that the victim will visit.

***Implementation***

**Cross-Site Request Forgery**

An XSRF attack is functionally the opposite of a cross-site scripting (XSS) attack, in which the hacker inserts malicious coding into a link on a Web site that appears to be from a trustworthy source. When an end user clicks on the link, the embedded programming is submitted as part of the client's Web request and can execute on the user's computer.

An XSRF attack also differs from cross-site tracing (XST), a sophisticated form of XSS that allows an intruder to obtain [cookie](https://searchsoftwarequality.techtarget.com/definition/cookie)s and other authentication data using simple client-side [script](https://whatis.techtarget.com/definition/script). In XSS and XST, the end user is the primary target of the attack. In XSRF, the Web server is the primary target although collateral harm is often done to individual end users.

***Implementattion***

i)

**Safely validating untrusted HTML input** to solve the problem of cross-site scripting.

When accepting HTML input from users (say, <b>very</b> large), output encoding (such as &lt;b&gt;very&lt;/b&gt; large) will not suffice since the user input needs to be rendered as HTML by the browser (so it shows as "very large", instead of "<b>very</b> large"). Stopping an XSS attack when accepting HTML input from users is much more complex in this situation. Untrusted HTML input must be run through an HTML sanitization engine to ensure that it does not contain XSS code.

It should also be noted that many validations rely on parsing out (blacklisting) specific "at risk" html tags such as the following

<script> <link> <iframe>

There are several issues with this approach, for example sometimes seemingly harmless tags can be left out which when utilized correctly can still result in an XSS

(see the below example)

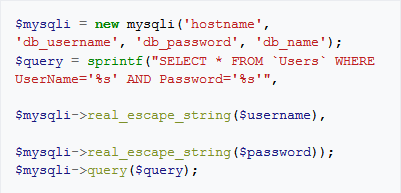
<img src="javascript:alert(1)">

Another popular method is to strip user input of " and ' however this can also be bypassed as the payload can be concealed with Obfuscation.

**Escaping to mitigate SQL injection** to solve SQL injection

A straightforward, though error-prone way to prevent injections is to escape characters that have a special meaning in SQL. The manual for an SQL DBMS explains which characters have a special meaning, which allows creating a comprehensive blacklist of characters that need translation. For instance, every occurrence of a single quote (') in a parameter must be replaced by two single quotes ('') to form a valid SQL string literal.

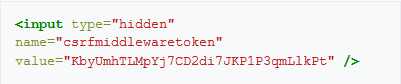
For example, in Python it is usual to escape parameters using the function mysqli\_real\_escape\_string(); before sending the SQL query:



**Synchronizer token pattern** to mitigate Cross-site forgery

Synchronizer token pattern (STP) is a technique where a token, secret and unique value for each request, is embedded by the web application in all HTML forms and verified on the server side. The token may be generated by any method that ensures unpredictability and uniqueness (e.g. using a hash chain of random seed). The attacker is thus unable to place a correct token in their requests to authenticate them.

Example of STP set by Django in a HTML form:



STP is the most compatible as it only relies on HTML, but introduces some complexity on the server side, due to the burden associated with checking validity of the token on each request. As the token is unique and unpredictable, it also enforces proper sequence of events (e.g. screen 1, then 2, then 3) which raises usability problem (e.g. user opens multiple tabs). It can be relaxed by using per session CSRF token instead of per request CSRF token

m)

1. **Exception/ Error handling:** An exception is an error that handles during execution of a program, when that error occurs, Python generates an exception that can be handled, which avoids your program to crash.

Exceptions are convenient in many ways for handling errors and special conditions in a program. When you have a code that can produce an error then you can use exception handling. A programmer implements a handling mechanism as a practice for secure programming to achieve the fail secure principle of programming. This can be used to handle errors that occur during data submissions and database queries

**Exception handling in python queries.**

try:

print 1/0

except ZeroDivisionError:

print "You can't divide by zero."

**2. Regular expressions:** Regular expressions are a powerful language for matching text patterns. The Python "re" module provides regular expression support.

In Python a regular expression search is typically written as:

  match = re.search(pat, str)

The re.search() method takes a regular expression pattern and a string and searches for that pattern within the string. If the search is successful, search() returns a match object or None otherwise. Therefore, the search is usually immediately followed by an if-statement to test if the search succeeded, as shown in the following example which searches for the pattern 'word:' followed by a 3 letter word (details below):



The code match = re.search(pat, str) stores the search result in a variable named "match". Then the if-statement tests the match -- if true the search succeeded and match.group() is the matching text (e.g. 'word:cat'). Otherwise if the match is false (None to be more specific), then the search did not succeed, and there is no matching text.

The 'r' at the start of the pattern string designates a python "raw" string which passes through backslashes without change which is very handy for regular expressions (Java needs this feature badly!). I recommend that you always write pattern strings with the 'r' just as a habit.

3) WTFR forms