**MAKERERE UNIVERSITY**

**COLLEGE OF COMPUTING AND INFORMATION SCIENCES**

**DEPARTMENT OF COMPUTER SCIENCE**

**COURSE UNIT:** BSE 3105 ADVANCED PROGRAMMING

**LECTURER:** Mr. LWOMWA JOSEPH

**ASSIGNMENT:** TEST 2

|  |  |  |
| --- | --- | --- |
| NAME | REGISTRATION NUMBER | STUDENT NUMBER |
| OKOT EMMANUEL | 16/U/10916/PS | 216015844 |

Question One

1. Potential vulnerabilities that could be exploited in the system:
   1. Software vulnerabilities such as:
      1. Insufficient testing
      2. Memory safety violations which causes buffer overruns
      3. Privilege-confusion such as cross-site request forgery
      4. Race conditions such as SQL injection
   2. Human vulnerabilities such as:
      1. Inadequate security awareness
2. How the vulnerabilities can be exploited
   1. An attacker can install malware to export sensitive data
   2. An insider can disclose sensitive data during review of security implications
3. A threat model is a security-based analysis that helps determine the highest-level security risks posed to the product and how attacks can manifest themselves. The goal is to determine which threats require mitigation and how to mitigate the threats enabling one think of the application in a relatively formal way.
   1. It helps understand the application better.
   2. It helps find bugs.
   3. It helps find complex bugs.
   4. It helps new team members understand the application in detail.
   5. It should read by other product teams that build on the product.
   6. It provided a strict guideline for patch and update management.
4. The threat-modeling process is as follows:
   1. Identify assets: A list of assets that require protection is built. Assets are either concrete or abstract.
   2. Create an Architecture Overview: Identify what the application does and create an application architecture diagram. The technologies at each stage are identified.
   3. Decompose the Application: The application is broken down and the composition of the application is known. 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.
   4. Determine the threats to the system:
      1. Assemble the threat-modeling team. Gather people from the product group to perform the initial threat process, that is look at the application or design and work out how an attacker could compromise the system.
      2. Categorize the threats by applying the following strategies;
         1. 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.
         2. Using Threat Trees. They describe the decision-making process an attacker would go through to compromise the component.
         3. Using Attack Trees: They are conceptual diagrams showing how an asset can be attacked.
   5. Document threats: The document includes the threat target and threat description. It will also include attack techniques, vulnerabilities and countermeasures. During this stage, the designer will;
      1. Choose how to respond to the threats:
      2. Choose techniques to mitigate the threats: Designer determines how to allay the threats.
      3. Choose the appropriate technologies for the identified techniques
   6. Rank the threats by decreasing the risk: After capturing the threats, determine the most important threats. The threats can be ranked by;
      1. Multiplying damage potential by likelihood of vulnerability
      2. Using DREAD (Damage potential, Reproducibility, Exploitability, Affected Users, Discoverability)
5. Diagrams used in;
   1. Step 3

Data Flow Diagrams (DFDs) and parts of the Unified Modeling Language (UML) are used to decompose the application into its key components.

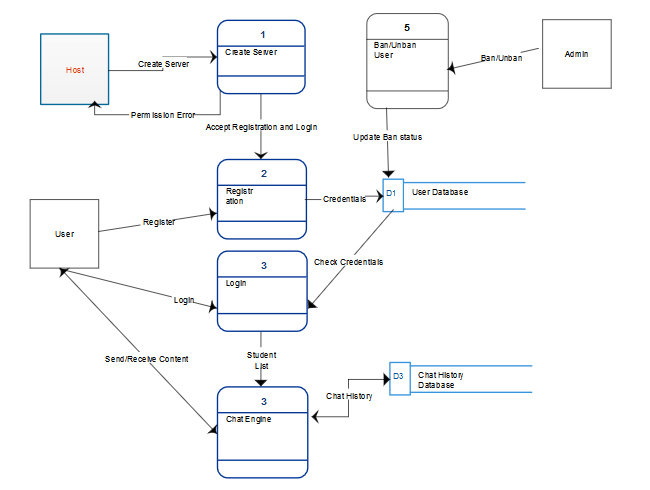


Figure 1 DFD Diagram for Chat Engine

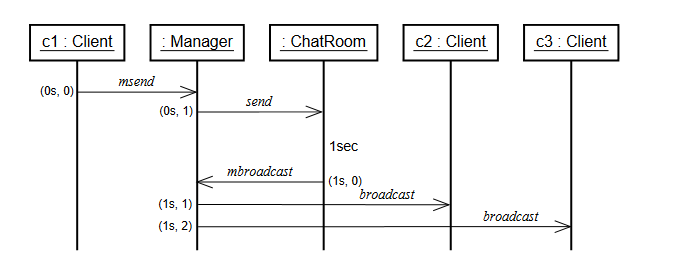


Figure 2 Sequence diagram of the message pattern

* 1. Step 4

Attack trees are employed during threat determination

Obtain sensitive data over the network

Obtain as clear text

Inject spam and malware

Attacker obtains user sensitive data

* 1. Step 6

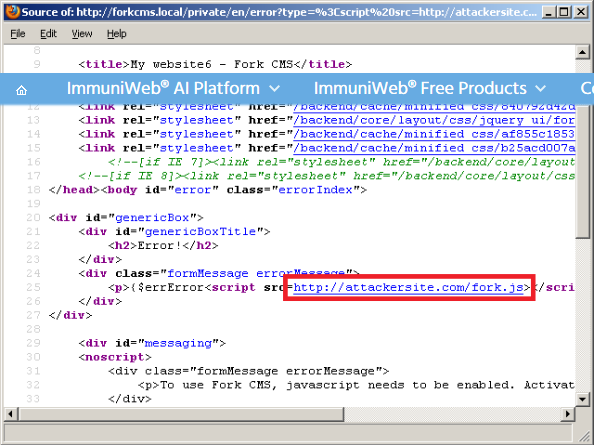
1. Application of the secure design principles.
   1. Develop regression tests for all previously fixed vulnerabilities in an example of learn from mistakes. With news reports about a similar incident, threats were determined.
   2. Use defense in depth. The system is built in a way that in the event of the failure, the system responds in a way such that access or data are denied. This prevents attackers from obtaining sensitive data
   3. Grant least privilege. If a glitch is noticed using the chat engine in least privilege mode, it will only run and affect in least privilege.
   4. Authenticate requests. Anticipate attacks.
   5. Making sure guidelines evolve as you learn new vulnerabilities makes us of promote privacy.
   6. Fail secure. In the event of failure of the system, default is to deny access to other users.
   7. Offering ease to users to give access to make security usable
   8. Designing simple mechanisms which can easily be understood. In the event of the error, it can easily be understood and fixed is an example of economic mechanism.
   9. Proportionality principle
   10. Fixing bugs that deviate from the guidelines makes application of assuming secrets are not safe. Regardless of the code’s age, attackers will keep on attacking.
   11. Require training for all personnel for detailed coverage on the subject makes application of securing the weakest link
   12. Adhering to design and code guidelines in order to Audit and Monitor the system
2. Vulnerabilities
   1. **Cross-Site Scripting** is a flaw in one Web page that renders client-side data tied to that domain insecure. An attacker uses a web application to send malicious code generally in form of browser side script, to a different user.

For example, **Cookie theft**, a malicious user can steal cookies and use them to gain access to the application.

Successful exploitation requires that user, who is logged-in into the application, follows a specially crafted link to vulnerable website:



the resultant page looks as follows:



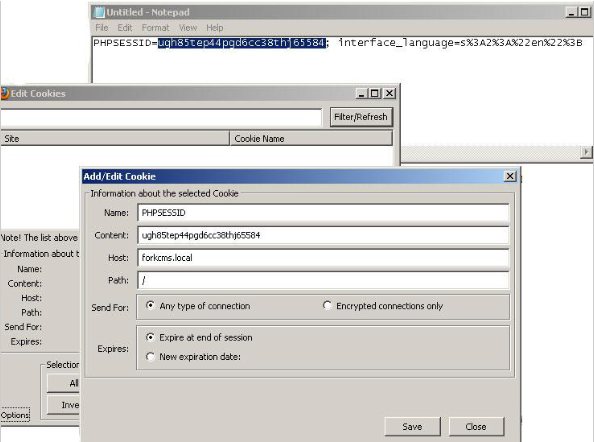
This means that the fork.js script located at the attackersite.com will be loaded and executed. The fork.js script contains the following code:



The above code sends HTTP POST request to file http://attackersite.com/fork.php that logs all data and redirects the victim back to the original website. The fork.php script contains the following code



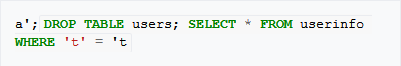
An attacker can use the received data to create cookies and gain access to the application. This can be achieved with any cookie editor plugin:



Using the Cookie Editor plugin for Firefox create the necessary cookies. After that, visit the vulnerable website:

* 1. **SQL injection** is a code injection, used to attack data-driven applications, in which nefarious SQL statements are inserted into entry field.

For example, in **incorrectly filtered escape characters**



This causes the deletion of the “users” table as well as the selection of all data from the “userinfo” table (in essence revealing the information of every user).

* 1. **Cross-site request** **forgery** is a weakness caused by insufficient or absent verification of the HTTP request origin. An attacker might trick a user into visiting a specially crafted webpage and forge request to the vulnerable application from client’s browser.

For example

1. Mitigations
   1. **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

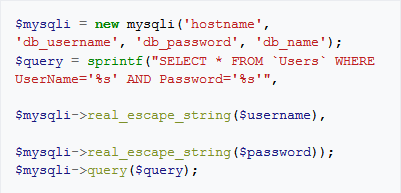
(see the below example)

<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.

* 1. **Escaping to mitigate SQL injection** to solve SQL injection

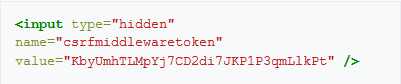
A straightforward, though error-prone way to prevent injections is to escape characters that have a special meaning in SQL. The manual for an SQL DBMS explains which characters have a special meaning, which allows creating a comprehensive blacklist of characters that need translation. For instance, every 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 PHP it is usual to escape parameters using the function mysqli\_real\_escape\_string(); before sending the SQL query:



* 1. **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

1. Exception/ error handling is a practice where the program continues to execute in an intended way when it comes across an error or an exception. 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

Regular expressions refer to the set of rules or combinations that denotes the structure of data usually strings. For example, a regular expression that can be used to allow emails.

(/ ^([a-z]+)([a-z A-Z0-9\_\-.]+) @ ([a-zA-Z0-9\_\-.]+)\.[a-zA-Z]{2,5}$/). Can be used to make sure that only strings of the above format can be accepted past the mechanism.

Question Two

1. Input sanitization describes cleansing and scrubbing user input to prevent it from jumping the fence and exploiting security holes.
2. Vulnerability attacks likely to affect the system
   1. SQL injection is a code injection, used to attack data-driven applications, in which nefarious SQL statements are inserted into entry field.
   2. Cross-Site request forgery is a weakness caused by insufficient or absent verification of the HTTP request origin. An attacker might trick a user into visiting a specially crafted webpage and forge request to the vulnerable application from client’s browser.
3. $safe\_data = filter\_input (INPUT\_GET, 'comment', FILTER\_SANITIZE\_SPECIAL\_CHARS);
4. Basic ways to sanitize objects are:
   1. **Fixed transformation** in which all occurrences of the object are replaced by a fixed string.
   2. **Typed transformation** in which the replacing objects are related when the type of the object being replaced are the same.
5. Other secure programming practices include;
   1. **Heed compiler warnings**: Compile code using the highest warning level available for your compiler and eliminate warnings by modifying code.
   2. **Keep it simple**: Keep the design as simple and small as possible.
6. It means defensive mechanisms have been designed to secure system from SQL injection