We Test Pens Incorporated

COMP90074 - Web Security Assignment 3
Katsuhide I
1132300

THREAT MODELLING REPORT FOR Bank of UniMelb Pty. Ltd. - WEB APPLICATION

Report delivered: 06/06/2021

S: Spoofing

Threat 1: Unauthorized actions by Stored Cross-site Scripting (XSS) vulnerability

Threat identified

Let us assume when a user wants to update the profile, the update will be sent to the branch manager for approval. An attacker could send a malicious script via stored Cross-site Scripting (XSS) vulnerability and run the script on the branch manager's browser. This would allow an attacker to perform unauthorised actions on their behalf.

Threat actor

The threat actor could be a bank client of the application or an external attacker who has the login credentials, targeting the branch managers.

Threat remediation

To mitigate this attack, we can encode the data from being interpreted as active content. Use HTML entity encodes: htmlentities($str); and also use an HTTP XSS protection header to ensure the functionality works as intended.

Threat 2: Unauthorized access to sensitive data by Server-Side Request Forgery (SSRF) vulnerability

Threat identified

Let us assume that in the profile a user can add their website and the system access and get the webpage to check if the website exists. An attacker can enter the local IP address in the profile page and let the server access itself. Then an attacker would be able to get the sensitive contents inside the server that unauthorized user cannot access.

Threat actor

The threat actor could be a bank client of the application or an external attacker who has the login credentials, targeting the server.

Threat remediation

To mitigate this vulnerability, we can create a whitelist of services accessible. In this case, we can block the loopback address as an input. Also, we can simply disable the function to check if the website exists.

T: Tampering

Threat 1: Modifying data in the database by SQL Injection

Threat identified

Let us assume that the application has a search bar for searching the history of transactions. An attacker could inject SQL query into the search bar and modify the data in the database.

Threat actor

The threat actor could be a bank client of the application or an external attacker who has the login credentials, targeting the database.

Threat remediation

To mitigate the risk of this vulnerability, we can use parameterised queries (prepared statement) to ensure that the query is in the intended structure.

Threat 2: Removing files on the server by Server-Side Template Injection

Threat identified

Let us assume that when a user updates the profile, the user input of the new data is dynamically embedded using template systems without proper sanitisations. An attacker could type system commands and modify the contents in the server, such as “system(‘rm sensitive.txt’)” to delete the file.

Threat actor

The threat actor could be a bank client of the application or an external attacker who has the login credentials, targeting the contents on the server.

Threat remediation

Do not allow users of the application to input the new profile information directly into a template. Also, sanitise user input. For example, if the input includes system commands, remove the commands from the input.

R: Repudiation

Threat 1: Modifying log data in the database by SQL Injection

Threat identified

Let us assume the application has a search bar for the transaction history. An attacker could inject SQL query to modify/delete the log data of attacking in the database.

Threat actor

The threat actor could be a bank client of the application or an external attacker who has the login credentials, targeting the log data in the database.

Threat remediation

To mitigate the risk of this vulnerability, we can use parameterised queries (prepared statement) to ensure that the query is in the intended structure.

Threat 2: Using stolen cookies of other users by stored Cross-site Scripting vulnerability

Threat identified

Let us assume that an attacker could successfully inject malicious scripts by stored Cross-site Scripting (XSS) vulnerability from the approval of updating the profile and obtain the cookies of other users. An attacker could use the cookie of other users in the request and spoof the identity.

Threat actor

The threat actor could be a bank client of the application or an external attacker who has the login credentials, targeting the cookie of other users.

Threat remediation

To mitigate this attack, we can encode the data from being interpreted as active content. Use HTML entity encodes: htmlentities($str); and use an HTTP XSS protection header to ensure the functionality works as intended.

I: Information Disclosure

Threat 1: developer comments in source code of the pages

Threat identified

Let us assume that the developer forgot to remove the comment in the source code and deploy the application. The developer comment might include the clue related to the system structure of the application and an attacker can get enough hints to find new vulnerabilities.

Threat actor

The thread actor could be a bank client of the application or an external attacker, trying to find new vulnerabilities.

Threat remediation

Delete all developer comments.

Threat 2: Sensitive files / directories left behind during testing / development

Threat identified

Let us assume that sensitive files/directories left behind during testing/development are not deleted or prohibited to access. An attacker can use the directory/file scanning tools to find the sensitive files/directories.

Threat actor

The threat actor could be an external attacker, scanning what files/directories are available on the webserver.

Threat remediation

To remediate this threat, prohibit access to the sensitive files/directories or delete them before the application is deployed.

D: Denial of Service

Threat 1: Sending many requests from the user's browsers by stored Cross-site Scripting vulnerability

Threat identified

Let us assume that the application has the place to inject a malicious script by stored Cross-site Scripting, such as the profile update. An attacker could inject a script that sends many requests to the web servers from the user's browsers. The webserver would then go down because of receiving too many requests and consuming all resources.

Threat actor

The threat actor could be bank clients of the application and the external attacker who obtains the login credentials, targeting other users to send many requests to the webserver.

Threat remediation

The possible remediation is to use HTML entity encodes: htmlentities($str); and use an HTTP XSS protection header to ensure the functionality works as intended.

Threat 2: Slow HTTP Denial of Service Attack

Threat identified

Let us assume that the application does not set the proper connection timeout for the requests, does not limit the number of connections to a machine and use old Apache. If an attacker keeps sending partial HTTP requests to open many connections to the webserver, the webserver will lose resources and go down.

Threat actor

The threat actor could be a bank client of the application and external attackers, targeting the webserver to be down.

Threat remediation

To reduce the risk of this threat, we can take several approaches: limit the number of connections to a single IP address, set a shorter connection timeout for a request, update the latest Apache, and use Apache modules to prevent this attack.

E: Elevation of Privilege

Threat 1: Horizontal privilege escalation by account takeover

Threat identified

Let us assume that the application has the change password function with parameters of username, old and new passwords. An attacker could change a different user's password by sending a different username as a parameter. An attacker could take over the higher privileged account.

Threat actor

The threat actor could be a bank client of the application or an external attacker who has the login credentials, targeting the account that has the higher privilege.

Threat remediation

To remediate this threat, we can add proper authorisations, such as sending a verification message to the user who wants to change the password. Once the user agrees with changing the password, the password will be finally changed.

Threat 2: Vertical privilege scalation by SQL Injection

Threat identified

Let us assume that the application stores a value that a user is an admin or not in the database and has SQL Injection vulnerability, such as the search transaction history. An attacker could inject an SQL query to change the value to change the privilege of the user.

Threat actor

The threat actor could be a bank client or external attacker who has the login credentials, trying to inject SQL query to change the privilege of the user.

Threat remediation

Use parameterised queries (prepared statement) to ensure that the query is in the intended structure.