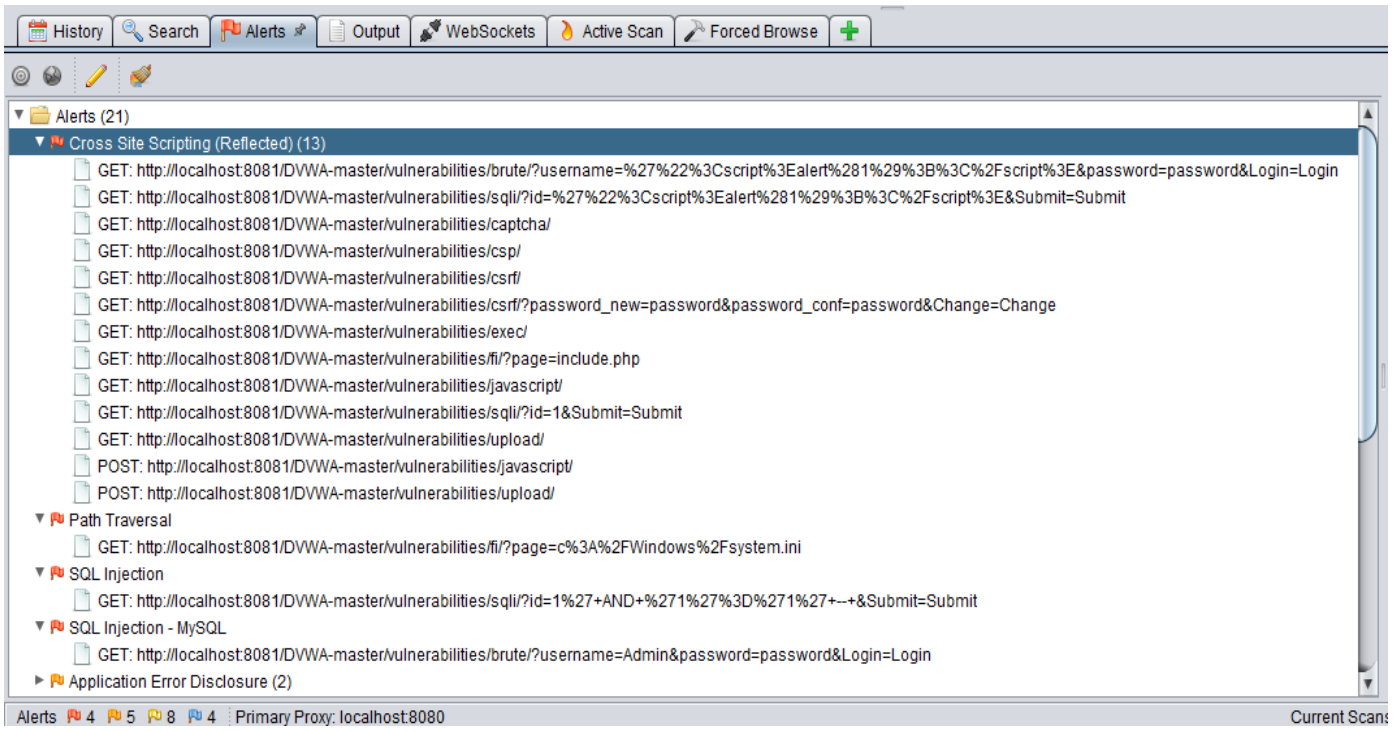
DVWA Vulnerability Report

1) Report vulnerabilities detected in DVWA web application when scanned with OWASP ZAP.

Below detected vulnerabilities are other than lab assignment.



2) Discuss any five commonly occuring vulnerabilities of your choice.

I. Path Traversal

The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal.

Most web sites restrict user access to a specific portion of the file-system, typically called the “web document root” or “CGI root” directory. These directories contain the files intended for user access and the executable necessary to drive web application functionality. To access files or execute commands anywhere on the file-system, Path Traversal attacks will utilize the ability of special-characters sequences.

The most basic Path Traversal attack uses the “../” special-character sequence to alter the resource location requested in the URL. Although most popular web servers will prevent this technique from escaping the web document root, alternate encodings of the “../” sequence may help bypass the security filters. These method variations include valid and invalid Unicode-encoding ("..%u2216” or

“..%c0%af”) of the forward slash character, backslash characters ("..") on Windows-based servers, URL encoded characters “%2e%2e%2f”), and double URL encoding ("..%255c”) of the backslash character.

II. Parameter Tempering

Parameter tampering is a web-based attack targeting the application business logic in order to perform or achieve a specific malicious task/attack different from the intended behaviour of the web application.

Parameter manipulation caused an error page or Java stack trace to be displayed. This indicated lack of exception handling and potential areas for further exploit.

III. X-Frame-Options Header Not Set

Clickjacking: X-Frame-Options header missing
Clickjacking (User Interface redress attack, UI redress attack, UI redressing) is a malicious technique of tricking a Web user into clicking on something different from what the user perceives they are clicking on, thus potentially revealing confidential information or taking control of their computer while clicking on seemingly innocuous web pages.

The server didn't return an **X-Frame-Options** header which means that this website could be at risk of a clickjacking attack. The X-Frame-Options HTTP response header can be used to indicate whether or not a browser should be allowed to render a page inside a frame or iframe. Sites can use this to avoid clickjacking attacks, by ensuring that their content is not embedded into other sites.

IV. SQL Injection

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data:

V. X-Content-Type-Options Header Missing

The Anti-MIME-Sniffing header X-Content-Type-Options was not set to ‘nosniff’. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body,

potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.

3) What are the security consequencies of such vulnerabilities being present in your web application?

I. Path Traversal

Path Traversal attacks are performed when the vulnerable application allows uncontrolled access to files and directories, to which the user should not usually have access. The attack vector is the parameters passed on the application, representing paths to resources, on which specific operations are to be performed – reading, writing, listing the contents of the directory. Path manipulation is done, for example, by adding the string "../".

Revealing redundant information

Successful attack using the vulnerability of Path Traversal, it is possible to list the contents of any directory. The attacker then gets additional information about the application architecture (file and directory structure). They may be useful, for example, to discover the elements of the tested system.

Another potential attack scenario is to read the contents of files, containing the history of executed commands on the server (e.g. Bash_history). Access to this type of files may lead to the disclosure of access data to local as well as external services (e.g. password to the database server).

Revealing configuration files

Access to configuration files. This threat may be particularly crucial if the attacker gains access to information about services that are available from outside.

Remote code execution

The vulnerabilities associated with Path Traversal are not limited to the ability to read files. If the application incorrectly validates the data when uploading files to the server, theoretically, there may be a situation when the attacker will have an impact on the path of writing the file on the server. This, in turn, depending on the server configuration, can directly lead to a situation when it will be possible to place the file anywhere in the directory tree. Here, of course, you must take into account the restrictions associated with the permissions where the write operation is performed.

II. Parameter Tempering

Parameter tampering is a web-based attack targeting the application business logic in order to perform or achieve a specific malicious task/attack different from the intended behaviour of the web application.

The attack involves modifying application data, such as user credentials and permissions, price and quantity of products, etc, by manipulating the parameters that are being exchanged between the

client and the application's server. Parameter tampering is considered to be simple but quite effective.

Parameter tampering can often be done with:

- Cookies
- Form fields
- HTTP headers
- URL query strings
- Cross site Scripting

III. X-Frame Options Header Not Set

Clickjacking (User Interface redress attack, UI redress attack, UI redressing) is a malicious technique of tricking a Web user into clicking on something different from what the user perceives they are clicking on, thus potentially revealing confidential information or taking control of their computer while clicking on seemingly innocuous web pages.

IV. Sql Injection

A successful SQL Injection attack can have very serious consequences.

- Attackers can use SQL Injections to find the credentials of other users in the database. They can then impersonate these users. The impersonated user may be a database administrator with all database privileges.
- SQL lets you select and output data from the database. An SQL Injection vulnerability could allow the attacker to gain complete access to all data in a database server.
- SQL also lets you alter data in a database and add new data. For example, in a financial application, an attacker could use SQL Injection to alter balances, void transactions, or transfer money to their account.
- You can use SQL to delete records from a database, even drop tables. Even if the administrator makes database backups, deletion of data could affect application availability until the database is restored. Also, backups may not cover the most recent data.
- In some database servers, you can access the operating system using the database server. This may be intentional or accidental. In such case, an attacker could use an SQL Injection as the initial vector and then attack the internal network behind a firewall.

V. X-Content-Type-Options Header Missing

The doesn't have a header settings for X-Content-Type Options which means it is vulnerable to MIME sniffing. The only defined value, "nosniff", prevents Internet Explorer and Google Chrome from MIME-sniffing a response away from the declared content-type. This also applies to Google Chrome when downloading extensions. This reduces exposure to drive-by download attacks and sites serving user uploaded content that by clever naming could be treated by MSIE as executable or dynamic HTML files.

4) What are proper programmming practices/countermesures that need to adapted to mitigate these vulnerabilties?

I. Path Traversal

Solution:

The basic form of protection against this vulnerability is the proper validation of input data. If possible, it is also recommended to use mechanisms that identify the files on the disk, e.g. through unique identifiers (UUID). Also, remember to verify file permissions (e.g. appropriate permissions for running a web server).

Blocking the attack by substituting the text

In web applications, you can encounter security that searches for the occurrence of the string, "../", in the path to the file. The protection consists of removing such strings from the text representing the name of the file or the path to it. This is quite an unfortunate solution because, through simple manipulation, the protection is easy to overcome. When passing the following string to the application, only its middle part will be removed:

II. Parameter Tempering

Solution:

Parameter tampering can be prevented by the following means:

- Use a whitelist format for the application's inputs.
- Use web application firewalls for utmost protection.
- Encrypt the session cookies to prevent tampering.
- If the cookie originated from the client-side, such as a referrer it should not be used to make any security decisions.
- Avoid including parameters into the query string.

III. X-Frame Options Header Not Set

Solution:

Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).

IV. Sql Injection

Solution:

The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.

If you discover an SQL Injection vulnerability, for example using an Acunetix scan, you may be unable to fix it immediately. For example, the vulnerability may be in open source code. In such cases, you can use a web application firewall to sanitize your input temporarily.

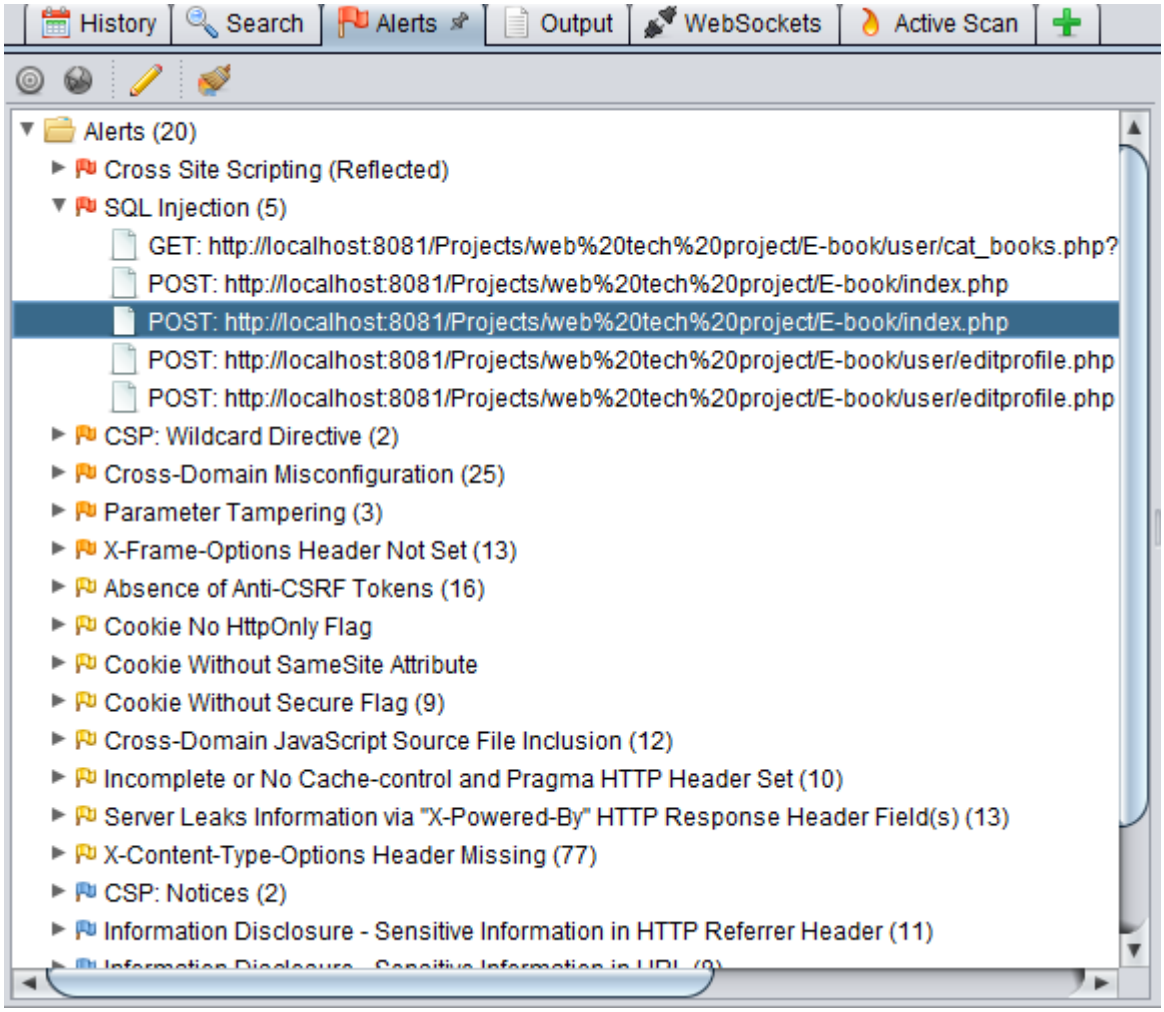
V. X-Content-Type-Options Header Missing

Solution:

Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

5) If you have scanned your own web application using OWASP ZAP, report which vulnerabilities were detected.

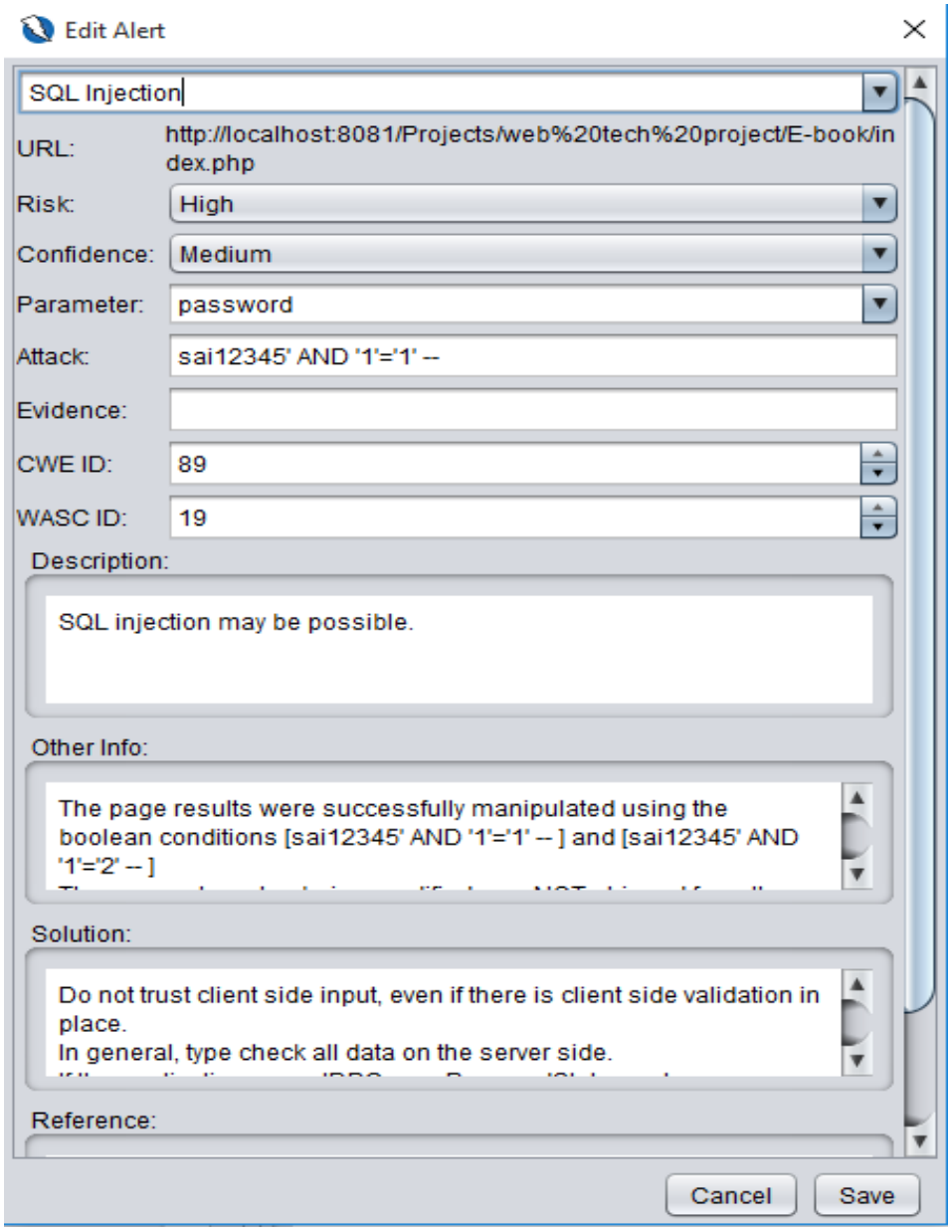
Used my Sem 4 Web tech Project to find vulnerability



1)

Vulnerability: <u>Sql Injection</u>

Page: login Page



Solution :

Do not trust client side input, even if there is client side validation in place.
In general, type check all data on the server side.

If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'

If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.

If database Stored Procedures can be used, use them.
Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!

Do not create dynamic SQL queries using simple string concatenation.
Escape all data received from the client.

Apply an 'allow list' of allowed characters, or a 'deny list' of disallowed characters in user input.
Apply the principle of least privilege by using the least privileged database user possible.

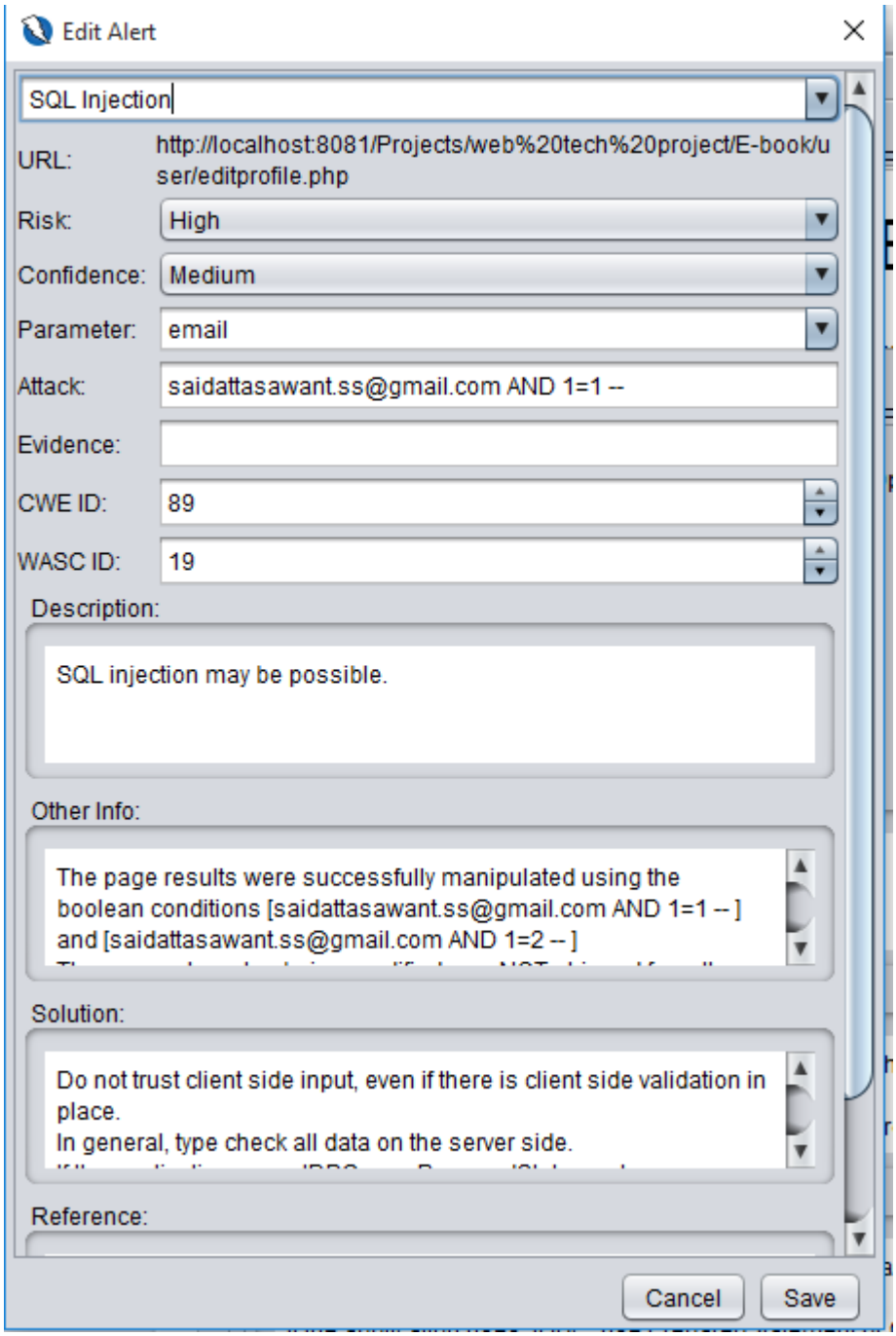
In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact.

Grant the minimum database access that is necessary for the application

2)

Vulnerability: <u>Sql Injection</u>

Page: Edit profile (Email field)

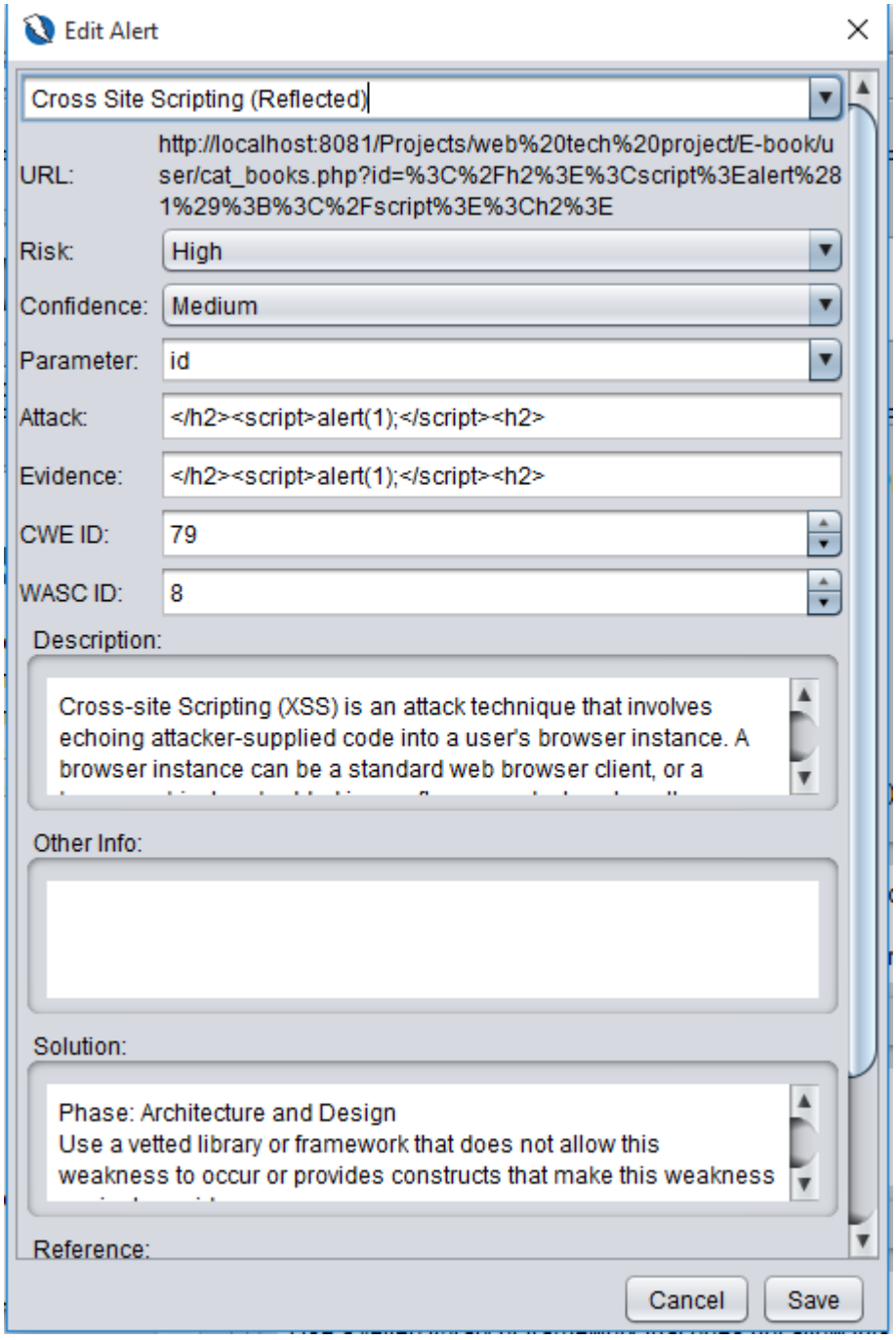


Solution: same as above sql injection.

3)

Vulnerability: <u>Cross site scripting (Reflected)</u>

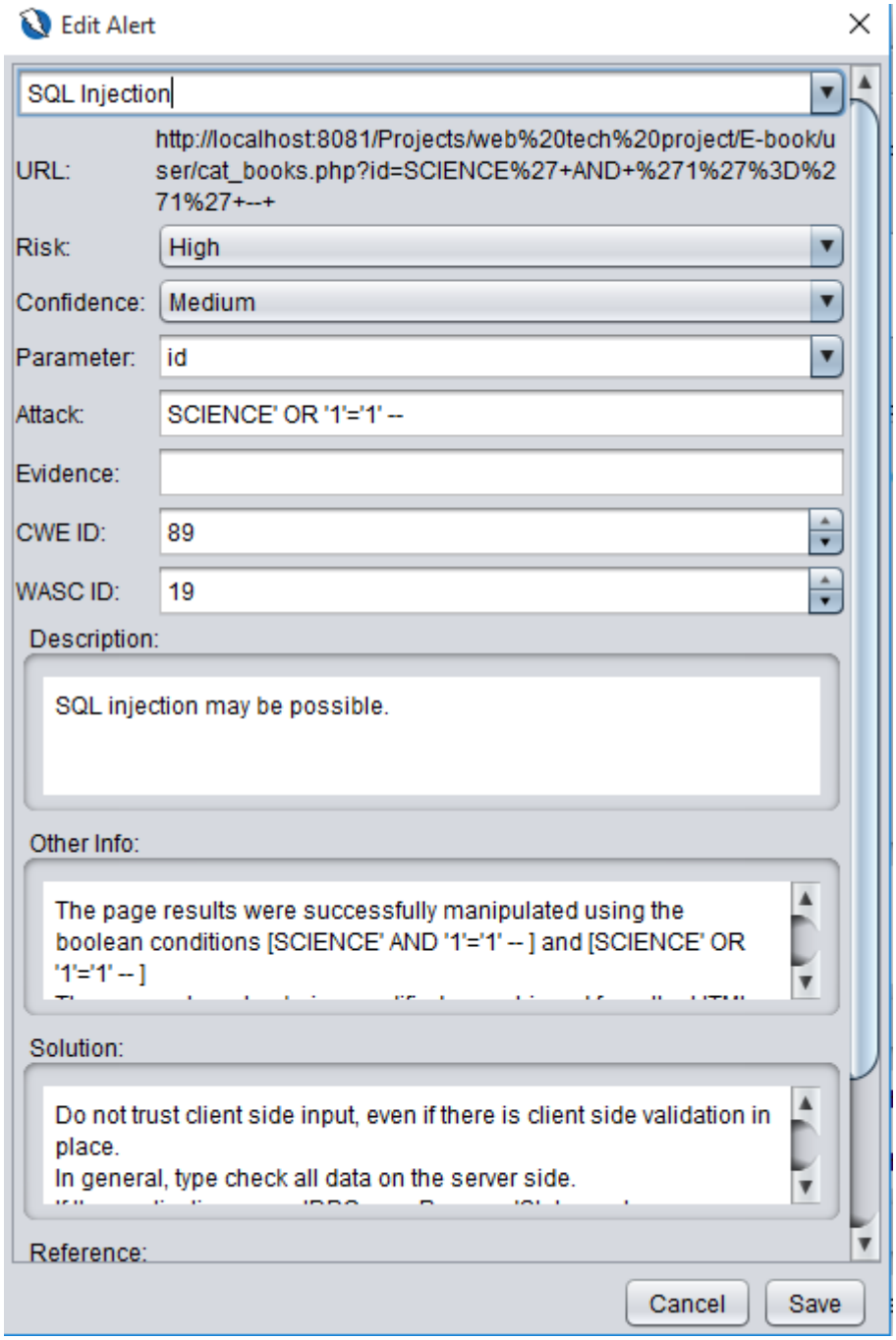
Page : Category books



4)

Vulnerability : Sql Injection

Page : Describe Books

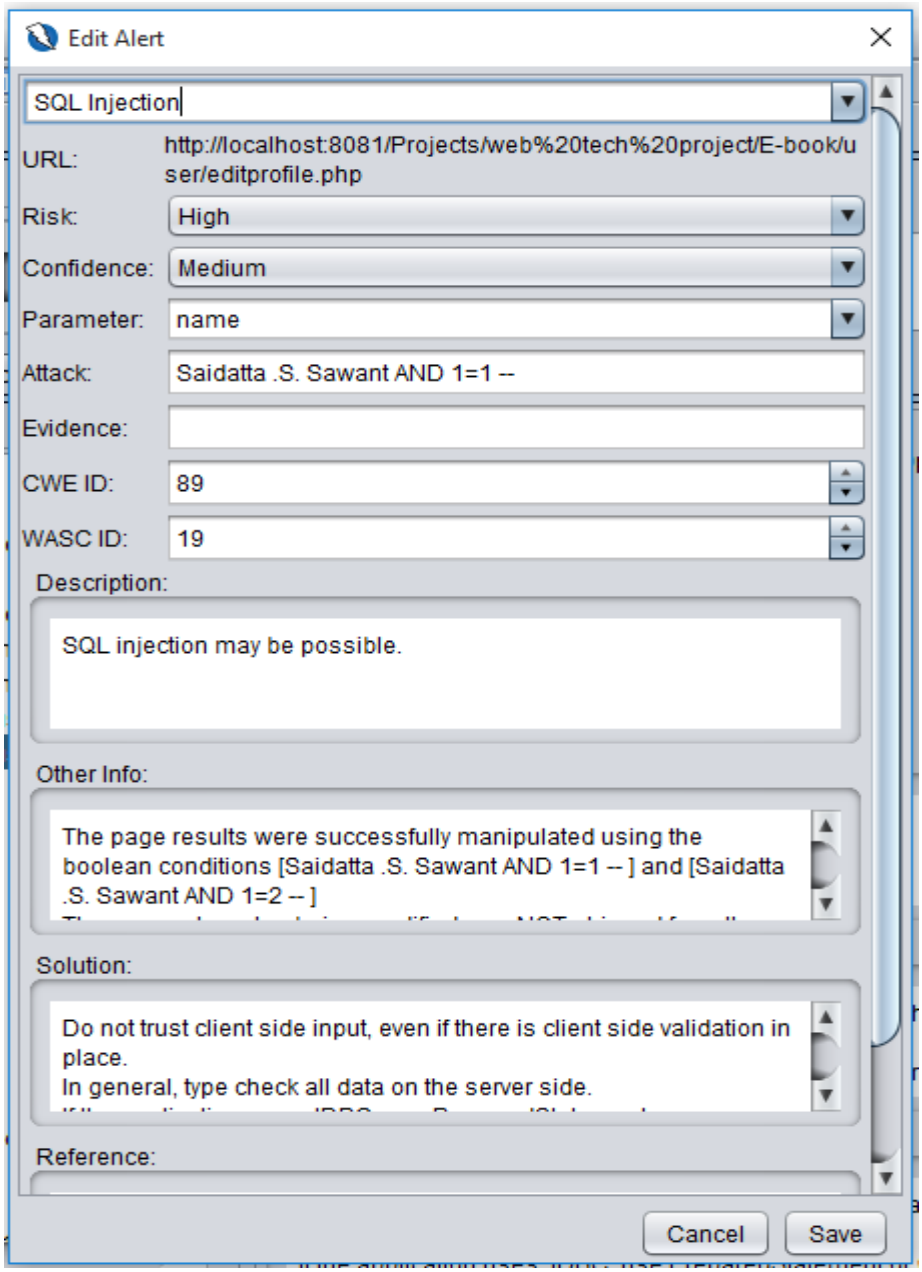


Solution: same as above sql injection

5)

Vulnerability: <u>Sql Injection</u>

Page : edit profile (NAME Field)



Solution : same as above sql injection.