Report

Vulnerability Report of Website: http://testasp.vulnweb.com/

My intentions were only to detect vulnerabilities in the website rather than to harm it. There are some issues in your website which i have found and are listed as below:

- **Transmission of password over HTTP**: I detected that data from the password section could be transmitted over HTTP as the website is using http for the communication.

Impact: If an attacker performs Man In The Middle attack to intercept the network traffic then he/she can steal the user credentials like password over http.

Solution: You should use HTTPS instead of using HTTP as HTTPS is a secured version of HTTP.

- **Boolean Based SQL Injection**:

  URL: http://testasp.vulnweb.com/Login.asp?RetURL=/Default.asp?

→ Identified Database Version (cached):
microsoft sql server 2014 (sp3-gdr) (kb4583463) - 12.0.6164.21 (x64) nov 1 2020 04 25 14 copyright (c) microsoft corporation express edition (64-bit) on windows nt 6.3 <x64> (build 9600) (hypervisor)

→ Identified Database User (cached): *acunetix*

→ Identified Database Name (cached): *acuforum*

- It occurs when data input by a user is interpreted as a SQL command rather than as normal data by the backend database.
- This is an extremely common vulnerability and its successful exploitation can have critical implications.

Impact: Depending on the backend database, the database connection settings and the operating system, an attacker can mount one or more of the following type of attacks successfully:

→ Reading, updating and deleting arbitrary data/tables from the database

→ Executing commands on the underlying operating system

Solution: The best way to protect your code against SQL injections is using parameterized queries (*prepared statements*). Almost all modern languages provide built-in libraries for this. Wherever possible, do not create dynamic SQL queries or SQL queries with string concatenation.

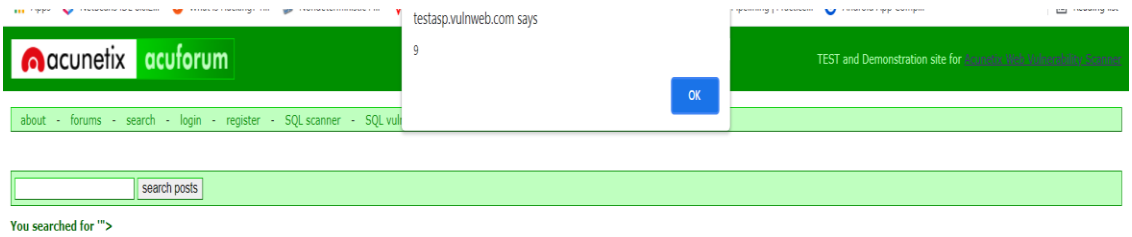
- **<u>Cross-site Scripting Attack</u>**:

→ Proof URL: http://testasp.vulnweb.com/Search.asp?tfSearch="><scRipt>alert(9)</scRipt>

OR

Script Payload Pattern: "><scRipt>alert(9)</scRipt>

The image below is a proof of this attack:



- It allows an attacker to execute a dynamic script (*JavaScript*, *VBScript*) in the context of the application.
- This allows several different attack opportunities, mostly hijacking the current session of the user or changing the look of the page by changing the HTML on the fly to steal the user's credentials.
- This happens because the input entered by a user has been interpreted as HTML/JavaScript/VBScript by the browser. Cross-site scripting targets the users of the application instead of the server.
- Although this is a limitation, since it allows attackers to hijack other users' sessions, an attacker might attack an administrator to gain full control over the application.

Solution: The issue occurs because the browser interprets the input as active HTML, JavaScript or VBScript. To avoid this, output should be encoded according to the output location and context.

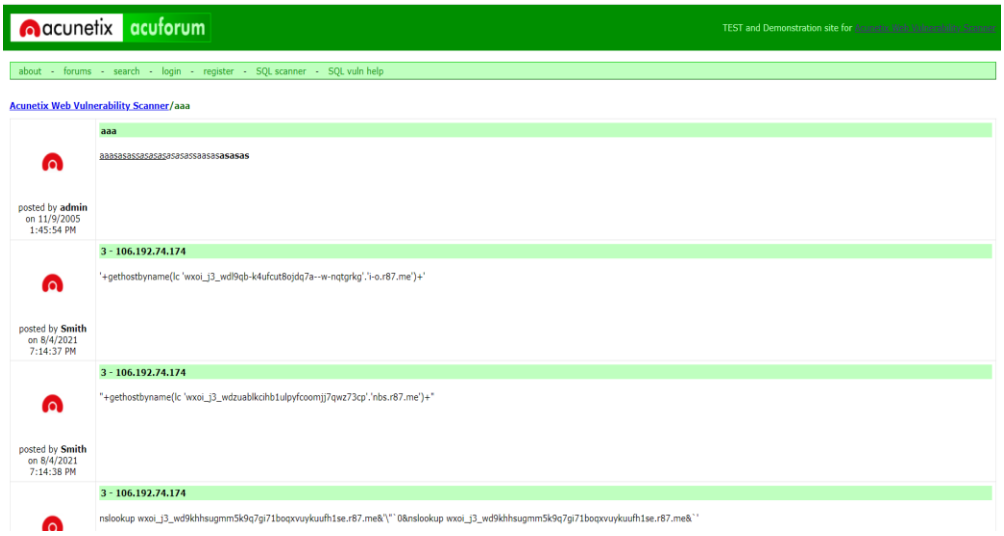
For example, if the output goes in to a JavaScript block within the HTML document, then output needs to be encoded accordingly.

Encoding can get very complex, therefore it's strongly recommended to use an encoding library such as OWASP ESAPI and Microsoft Anti-cross-site scripting.

- **Stored Cross-site Scripting Attack**: Stored XSS allows an attacker to execute dynamic scripts (*JavaScript*, *VBScript*) in the context of the application.
- This allows several different attack opportunities, mostly hijacking the current session of the user or changing the look of the page by changing the HTML on the fly, to steal the user's credentials. This happens because the input entered by the user has been interpreted by HTML/JavaScript/VBScript within the browser.
- "Stored" means that the attack will be stored in the backend system. In normal XSS attacks, an attacker needs to e-mail the victim, but in a stored XSS an attacker can just execute the attack and wait for users to see the affected page. As soon as someone visits the page, the attacker's stored payload will get executed.
- XSS targets the users of the application instead of the server. Although this is a limitation, since it only allows attackers to hijack other users' sessions, the attacker might attack an administrator to gain full control over the application.

Proof URL: http://testasp.vulnweb.com/showthread.asp?id=3

The image below is a proof of this attack:



Impact: Stored XSS is a dangerous issue that has many exploitation vectors, some of which include:

- User's session-sensitive information, such as cookies, can be stolen.
- XSS can enable client-side worms, which could modify, delete or steal other users' data within the application.
- The website can be redirected to a new location, defaced or used as a phishing site.

Solution: The issue occurs because the browser interprets the input as active HTML, JavaScript or VBScript.

To avoid this, all input and output from the application should be filtered. Output should be filtered according to the output format and location.

Typically the output location is HTML. Where the output is HTML, ensure all active content is removed prior to its presentation to the server.