**Multi-Vulnerability Report for:**

[**http://testasp.vulnweb.com**](http://testasp.vulnweb.com)

**1. Vulnerability Report: Persistent XSS in Web Application**

**Overview:**

The web application located at <http://testasp.vulnweb.com/showthread.asp?id=4> is susceptible to a Persistent Cross-Site Scripting (XSS) vulnerability. This vulnerability occurs when attacker-supplied code is injected into the web application and stored for a period of time, targeting users who access the affected content.

**Vulnerability Details:**

* **URL****:** <http://testasp.vulnweb.com/showthread.asp?id=4>
* **Affected Parameter:** tfText in the POST request
* **Attack Vector:** Malicious code injected in the tfText parameter

**Technical Details:**

1. **Request:**

POST http://testasp.vulnweb.com/showthread.asp?id=4 HTTP/1.1 host: testasp.vulnweb.com user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36 pragma: no-cache cache-control: no-cache content-type: application/x-www-form-urlencoded referer: http://testasp.vulnweb.com/showthread.asp?id=4 content-length: 21 Cookie: ASPSESSIONIDASSARBSC=OPCNPHLDALMGDPDNGEGMFGBM tfSubject=ZAP&tfText=<div><script>alert(1);</script><div>



1. **Response:**

HTTP/1.1 200 OK Cache-Control: private Content-Type: text/html Server: Microsoft-IIS/8.5 X-Powered-By: ASP.NET Date: Fri, 05 Jan 2024 15:30:31 GMT Content-Length: 21738

(Response Body - 21738 bytes of content)  


**Exploitation:**

The attacker successfully injects a script within the tfText parameter, leading to the execution of arbitrary JavaScript code when the affected page is viewed.

htmlCopy code

<div><script>alert(1);</script><div>

**Steps to Reproduce:**

1. **Prepare a JavaScript Payload:**
   * Create a JavaScript payload, for instance, alerting the user's cookies.

javascriptCopy code

alert(document.cookie);

1. **Inject Payload into tfText Parameter:**
   * Craft a POST request with the payload injected into the tfText parameter.

httpCopy code

POST http://testasp.vulnweb.com/showthread.asp?id=4 HTTP/1.1 host: testasp.vulnweb.com user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36 pragma: no-cache cache-control: no-cache content-type: application/x-www-form-urlencoded referer: http://testasp.vulnweb.com/showthread.asp?id=4 content-length: 21 Cookie: ASPSESSIONIDASSARBSC=OPCNPHLDALMGDPDNGEGMFGBM tfSubject=ZAP&tfText=<div><script>alert(document.cookie);</script><div>

1. **Send the Crafted Request:**
   * Send the crafted POST request to the web application, injecting the payload into the tfText parameter.
2. **Visit the Affected Page:**
   * Have a user visit the thread at <http://testasp.vulnweb.com/showthread.asp?id=4> to trigger the execution of the injected script.

**Impact:**

A user viewing the compromised content is exposed to the injected script, potentially leading to unauthorized actions, cookie theft, or redirection to malicious content. Persistent XSS poses a significant threat to the integrity and trust of the affected web application.

**Recommendations:**

1. **Architecture and Design:**
   * Implement input validation mechanisms on both client and server sides.
   * Employ vetted libraries or frameworks that facilitate proper encoding and escaping.
2. **Implementation:**
   * Specify character encoding (e.g., ISO-8859-1 or UTF-8) for all generated web pages.
   * Set session cookies to be HttpOnly to mitigate XSS attacks against user session cookies.
3. **Security Checks:**
   * Duplicate security checks on the server side to avoid bypassing client-side checks by attackers.
   * Employ structured mechanisms to enforce the separation between data and code.

**Conclusion:**

Addressing these recommendations will enhance the security posture of the web application, reducing the risk of XSS attacks and safeguarding user interactions.  
  
  
Certainly, here is a sample report for the identified Cross-Site Scripting (Reflected) vulnerability:

* 1. **Cross-Site Scripting (Reflected) Vulnerability Report**

**Overview**

**Target:** [**http://testasp.vulnweb.com**]   
**Vulnerability Type:** Cross-Site Scripting (Reflected)

**Description**

A Cross-Site Scripting (XSS) vulnerability has been identified in the application, specifically in the **tfText** parameter of the POST request to **http://testasp.vulnweb.com/showthread.asp?id=4**. This vulnerability allows an attacker to inject and execute arbitrary JavaScript code in the context of other users' browsers, compromising the security of the application.

**Attack Scenario**

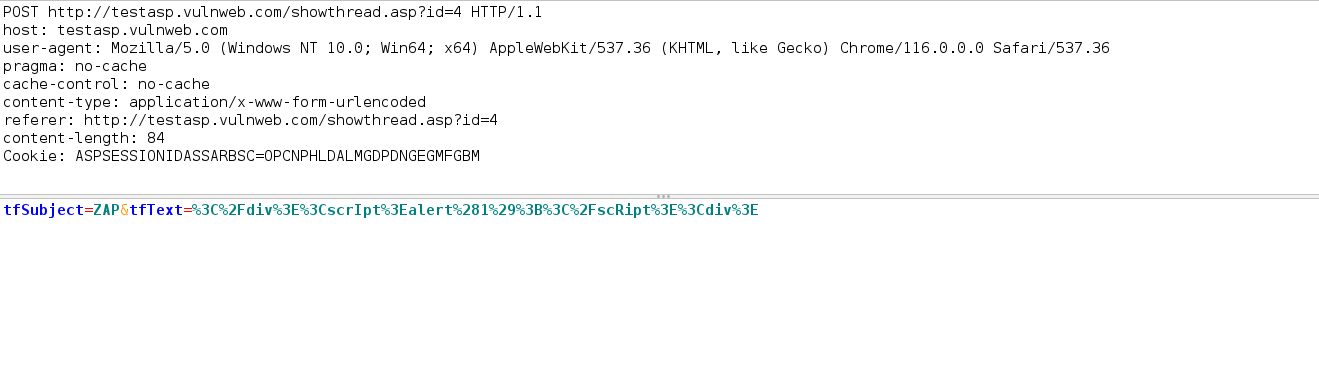
The attacker exploits the vulnerability by injecting a crafted payload in the **tfText** parameter. Upon successful execution, the payload is reflected in the response, demonstrating that the application does not properly validate and sanitize user inputs.

**Technical Details**

**Request**

POST http://testasp.vulnweb.com/showthread.asp?id=4 HTTP/1.1 host: testasp.vulnweb.com user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36 pragma: no-cache cache-control: no-cache content-type: application/x-www-form-urlencoded referer: http://testasp.vulnweb.com/showthread.asp?id=4 content-length: 84 Cookie: ASPSESSIONIDASSARBSC=OPCNPHLDALMGDPDNGEGMFGBM tfSubject=ZAP&tfText=%3C%2Fdiv%3E%3CscrIpt%3Ealert%281%29%3B%3C%2FscRipt%3E%3Cdiv%3E

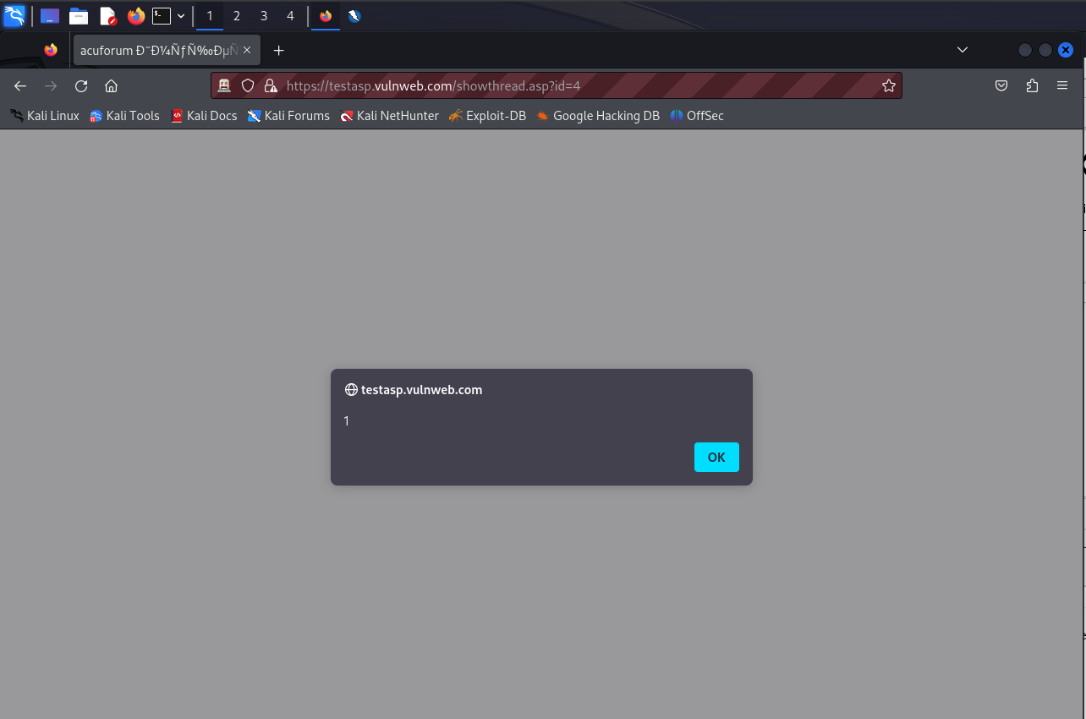
Request :-



**Response**

HTTP/1.1 200 OK Cache-Control: private Content-Type: text/html Server: Microsoft-IIS/8.5 X-Powered-By: ASP.NET Date: [Date] Content-Length: 20871 </div><scrIpt>alert(1);</scRipt><div>



**Poc:   
  
  
Recommendations**

1. **Input Validation:** Implement thorough input validation on both client and server sides. Ensure that user inputs are properly validated and sanitized to prevent malicious script injection.
2. **Content Security Policy (CSP):** Implement and enforce a strict Content Security Policy to mitigate the impact of XSS attacks by controlling which resources can be loaded.
3. **Output Encoding:** Use proper output encoding techniques to sanitize user inputs before rendering them on web pages.
4. **Security Mechanisms:** Utilize vetted libraries or frameworks designed to prevent XSS vulnerabilities, such as Microsoft's Anti-XSS library, OWASP ESAPI Encoding module, or Apache Wicket.

**Conclusion**

The identified XSS vulnerability poses a significant risk to the security of the application. Implementing the recommended measures will help mitigate this risk and enhance the overall security posture of the application.

* 1. **External Redirect Vulnerability**

**Description**

An External Redirect vulnerability has been identified in the application, specifically in the **RetURL** parameter of the GET request to **http://testasp.vulnweb.com/Logout.asp**. This vulnerability allows an attacker to manipulate the redirection URL, leading users to an external site. While URL redirectors are not inherently a security vulnerability, they can be exploited by attackers for phishing purposes, misleading users about the true destination.

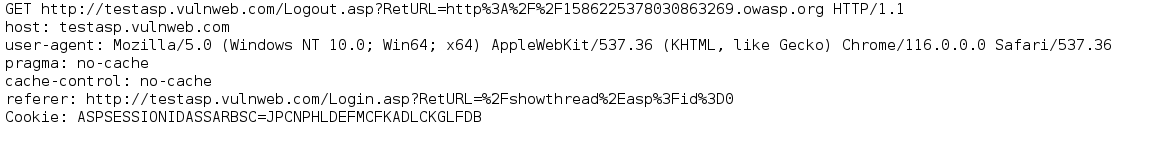
**Attack Scenario**

The attacker takes advantage of the External Redirect vulnerability by modifying the **RetURL** parameter to an external URL (**http://1586225378030863269.owasp.org**). As a result, users may be redirected to a malicious site without their knowledge or consent.

**Technical Details**

**Request**

GET http://testasp.vulnweb.com/Logout.asp?RetURL=http%3A%2F%2F1586225378030863269.owasp.org HTTP/1.1 host: testasp.vulnweb.com user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36 pragma: no-cache cache-control: no-cache referer: http://testasp.vulnweb.com/Login.asp?RetURL=%2Fshowthread%2Easp%3Fid%3D0 Cookie: ASPSESSIONIDASSARBSC=JPCNPHLDEFMCFKADLCKGLFDB



**Response**

HTTP/1.1 302 Object moved Cache-Control: private Content-Type: text/html Location: http://1586225378030863269.owasp.org Server: Microsoft-IIS/8.5 X-Powered-By: ASP.NET Date: [Date] Content-Length: 157 <head><title>Object moved</title></head> <body><h1>Object Moved</h1>This object may be found <a HREF="http://1586225378030863269.owasp.org">here</a>.</body>



**Recommendations**

1. **Allow List Validation:** Implement an allow list validation strategy for the **RetURL** parameter. Restrict the acceptable redirection URLs to a predefined list of approved domains or URLs.
2. **Intermediate Disclaimer Page:** Introduce an intermediate disclaimer page that warns users about leaving the site. Ensure a clear message is provided, and implement a timeout or user confirmation before redirection occurs.
3. **Input Validation:** Perform thorough input validation on all user inputs, including parameters, cookies, and headers. Reject inputs that do not strictly conform to specifications.
4. **Security Education:** Educate developers on the risks associated with open redirects and the importance of validating and sanitizing user inputs.

**Conclusion**

The identified External Redirect vulnerability poses a potential risk of phishing and social engineering attacks. Implementing the recommended measures will help mitigate this risk and enhance the overall security of the application.

* 1. **Path Traversal Vulnerability**

**Description**

A Path Traversal vulnerability has been identified in the application, specifically in the **item** parameter of the GET request to **http://testasp.vulnweb.com/Templatize.asp**. This vulnerability allows an attacker to navigate outside the web document root directory and potentially access sensitive files on the web server, as demonstrated by attempting to access **Windows/system.ini**.

**Attack Scenario**

The attacker exploits the Path Traversal vulnerability by manipulating the **item** parameter, providing a crafted path (**../../../../../../../../../../../../../../../../Windows/system.ini**). This allows unauthorized access to files outside the intended directory, potentially exposing sensitive information.

**Technical Details**

**Request**

GET http://testasp.vulnweb.com/Templatize.asp?item=..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2FWindows%2Fsystem.ini HTTP/1.1 host: testasp.vulnweb.com User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8 Accept-Language: en-US,en;q=0.5 Referer: http://testasp.vulnweb.com/ Connection: keep-alive Cookie: ASPSESSIONIDASSARBSC=DADNPHLDGEBLLNCMOMGPPEEI Upgrade-Insecure-Requests: 1



**Response**

httpCopy code

HTTP/1.1 200 OK Cache-Control: private Content-Type: text/html Server: Microsoft-IIS/8.5 X-Powered-By: ASP.NET Date: [Date] Content-Length: 3180



**Recommendations**

1. **Input Validation:** Implement thorough input validation on the **item** parameter. Apply an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not adhere to specifications or transform it into a safe format.
2. **Path Canonicalization:** Use a built-in path canonicalization function, such as **realpath()** in languages like C, that produces the canonical version of the pathname. This effectively removes ".." sequences and symbolic links.
3. **File Permission Restrictions:** Apply strict file permission restrictions to limit access to sensitive files. Ensure that the application runs with the lowest privileges necessary to perform required tasks.
4. **Security Education:** Educate developers on the risks associated with path traversal vulnerabilities and the importance of secure input handling.

**Conclusion**

The identified Path Traversal vulnerability poses a significant risk of unauthorized access to sensitive files on the web server. Implementing the recommended measures will help mitigate this risk and enhance the overall security of the application.

1. **SQL Injection Vulnerability**

**Description**

A SQL Injection vulnerability has been identified in the application, specifically in the **tfSubject** parameter of the POST request to **http://testasp.vulnweb.com/showthread.asp?id=4**. This vulnerability allows an attacker to manipulate SQL queries and potentially retrieve unauthorized data from the database.

**Attack Scenario**

The attacker successfully manipulated the SQL query using boolean conditions in the **tfSubject** parameter. The provided examples include [ZAP' AND '1'='1' -- ] and [ZAP' OR '1'='1' -- ]. The vulnerability allows the attacker to retrieve more data than originally intended, as observed by the increased response length.

**Technical Details**

**Request**

POST http://testasp.vulnweb.com/showthread.asp?id=4 HTTP/1.1 host: testasp.vulnweb.com user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36 pragma: no-cache cache-control: no-cache content-type: application/x-www-form-urlencoded referer: http://testasp.vulnweb.com/showthread.asp?id=4 content-length: 50 Cookie: ASPSESSIONIDASSARBSC=OPCNPHLDALMGDPDNGEGMFGBM Request body (50 bytes) tfSubject=ZAP%27+AND+%271%27%3D%271%27+--+&tfText=



**Response**

HTTP/1.1 200 OK Cache-Control: private Content-Type: text/html Server: Microsoft-IIS/8.5 X-Powered-By: ASP.NET Date: [Date] Content-Length: 21762



**Recommendations**

1. **Parameterized Queries:** Utilize parameterized queries with tools such as **PreparedStatement** or **CallableStatement** to prevent SQL injection. Parameterized queries ensure that user input is treated as data, not executable code.
2. **Input Validation:** Implement strict input validation on the server side. Avoid trusting client-side input, even when client-side validation is present. Perform type checking on all incoming data.
3. **Stored Procedures:** Whenever possible, use stored procedures to interact with the database. Avoid dynamic SQL queries using simple string concatenation.
4. **Escape User Input:** Escape all data received from the client before incorporating it into SQL queries. Apply an 'allow list' of allowed characters or a 'deny list' of disallowed characters in user input.
5. **Least Privilege Principle:** Grant the minimum database access necessary for the application. Avoid using highly privileged database users like 'sa' or 'db-owner' to minimize the impact of potential SQL injection attacks.

**Conclusion**

The identified SQL Injection vulnerability poses a significant risk of unauthorized access to sensitive data in the database. Implementing the recommended measures will help mitigate this risk and enhance the overall security of the application.

1. **SQL Injection (MsSQL)**

**Description**

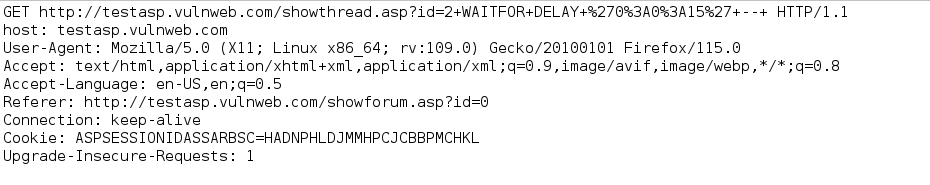
A SQL Injection vulnerability has been identified in the application, specifically in the **id** parameter of the GET request to **http://testasp.vulnweb.com/showthread.asp?id=2+WAITFOR+DELAY+%270%3A0%3A15%27+--+**. This vulnerability allows an attacker to control the query execution time, potentially leading to a delay-based attack.

**Attack Scenario**

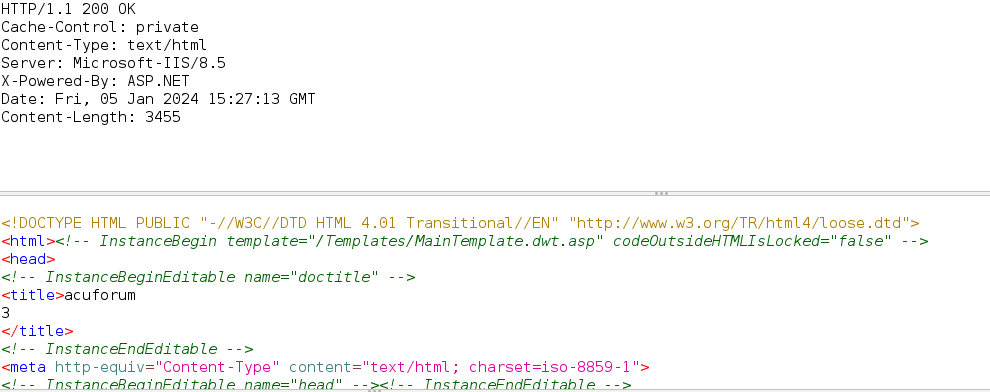
The attacker successfully manipulated the SQL query using the parameter value **[2 WAITFOR DELAY '0:0:15' -- ]**, causing the request to take approximately 30,382 milliseconds. The original unmodified query with the value **[2]** took only 342 milliseconds. This indicates that the query time is controllable by the attacker.

**Technical Details**

**Request**

GET http://testasp.vulnweb.com/showthread.asp?id=2+WAITFOR+DELAY+%270%3A0%3A15%27+--+ HTTP/1.1 host: testasp.vulnweb.com User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:109.0) Gecko/20100101 Firefox/115.0 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8 Accept-Language: en-US,en;q=0.5 Referer: http://testasp.vulnweb.com/showforum.asp?id=0 Connection: keep-alive Cookie: ASPSESSIONIDASSARBSC=HADNPHLDJMMHPCJCBBPMCHKL Upgrade-Insecure-Requests: 1   


**Response**

HTTP/1.1 200 OK Cache-Control: private Content-Type: text/html Server: Microsoft-IIS/8.5 X-Powered-By: ASP.NET Date: [Date] Content-Length: 3455   


**Recommendations**

1. **Parameterized Queries:** Utilize parameterized queries with tools such as **PreparedStatement** or **CallableStatement** to prevent SQL injection. Parameterized queries ensure that user input is treated as data, not executable code.
2. **Input Validation:** Implement strict input validation on the server side. Avoid trusting client-side input, even when client-side validation is present. Perform type checking on all incoming data.
3. **Stored Procedures:** Whenever possible, use stored procedures to interact with the database. Avoid dynamic SQL queries using simple string concatenation.
4. **Escape User Input:** Escape all data received from the client before incorporating it into SQL queries. Apply an 'allow list' of allowed characters or a 'deny list' of disallowed characters in user input.
5. **Least Privilege Principle:** Grant the minimum database access necessary for the application. Avoid using highly privileged database users like 'sa' or 'db-owner' to minimize the impact of potential SQL injection attacks.

**Conclusion**

The identified SQL Injection vulnerability poses a significant risk due to the ability to control query execution time. Implementing the recommended measures will help mitigate this risk and enhance the overall security of the application.