Vulnerability Assessment and Penetration Testing Report (VAPT)

Application Name : M360 Demo1
Type : Web
Date of Assessment : 19-12-2023
Report Prepared By: Eklavya Saraswat
Team : Application Security Team whitehat@timesinternet.in

Executive Summary

Purpose of the Assessment

This manual security testing report provides an in-depth analysis of our application's security posture, conducted in accordance with the Open Web Application Security Project (OWASP) testing standards. The assessment focused on addressing the OWASP Top 10 vulnerabilities, aiming to identify and mitigate potential security risks within the application.

Summary of Findings
The assessment revealed a range of vulnerabilities and security issues within the application. Key findings include:

- OWASP Top 10 Vulnerabilities: High and medium-severity vulnerabilities related to the OWASP Top 10 were identified, including [Injection issues].
- General Findings: No Additional vulnerabilities were found, such as inadequate error handling etc.
- Data Exposure: No instances of data exposure and information leakage were detected.

Issue	Host	Severity	Confidence	Status
XSS (Cross-Site Scripting)	**https://demo1.publishstory.co/**	**High**	**Confirmed**	**Open**

Curl Snippet :

curl -i -s -k -X $'GET' \ -H $'Host: demo1.publishstory.co' -H $'Sec-Ch-Ua: \"Not_A Brand\";v=\"8\", \"Chromium\";v=\"120\"' -H $'Sec-Ch-Ua-Mobile: ?0' -H $'Sec-Ch-Ua-Platform: \"macOS\"' -H $'Upgrade-Insecure-Requests: 1' -H $'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.71 Safari/537.36' -H $'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7' -H $'Sec-Fetch-Site: same-origin' -H $'Sec-Fetch-Mode: navigate' -H $'Sec-Fetch-User: ?1' -H $'Sec-Fetch-Dest: document' -H $'Accept-Encoding: gzip, deflate, br' -H $'Accept-Language: en-GB,en-US;q=0.9,en;q=0.8' -H $'Priority: u=0, i' \ -b $'m360auth=083103af2055015b140c3af0a882d021ca49bc3a7d404111c8f34a24b37cb2a1' \ $'https://demo1.publishstory.co/topics/poop?yde1y\"><script>alert(\'XSS\')</script>gqtr9=1'

Description :

Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data.

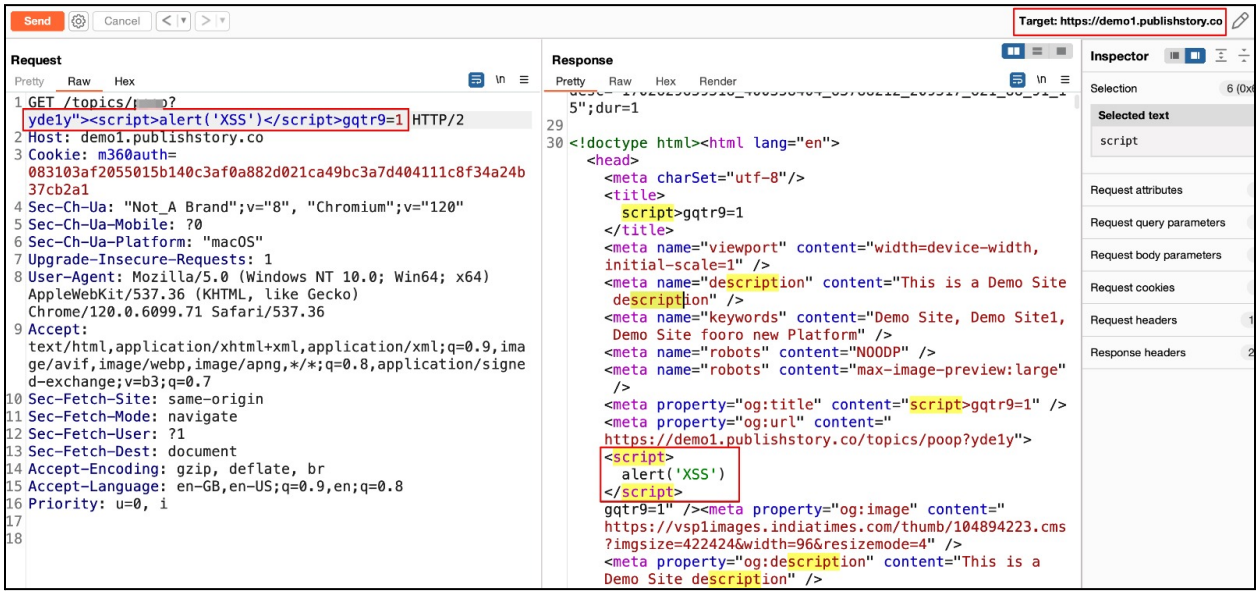
Reflected XSS is the simplest variety of cross-site scripting. It arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

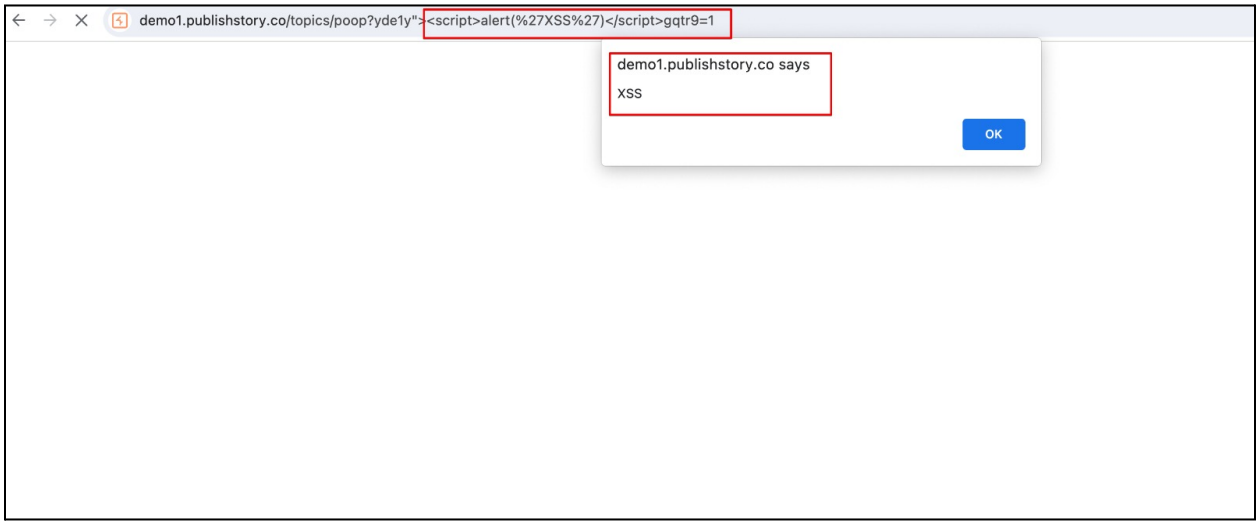
Steps to produce :

Functionality : Search Box

The application permits users to utilize the search functionality to meet their specific needs. The input value for the search is inserted into the value attribute of a few html tags. We manipulated the value attribute by balancing it and closing the tag, introducing a basic XSS payload. After clicking the search button, an XSS popup alert was triggered.

Proof of concepts :





Impact :

The actual impact of an XSS attack generally depends on the nature of the application, its functionality and data, and the status of the compromised user. For example:

- In a brochureware application, where all users are anonymous and all information is public, the impact will often be minimal.
- In an application holding sensitive data, such as banking transactions, emails, or healthcare records, the impact will usually be serious.
- If the compromised user has elevated privileges within the application, then the impact will generally be critical, allowing the attacker to take full control of the vulnerable application and compromise all users and their data.

An attacker who exploits a cross-site scripting vulnerability is typically able to:

- Impersonate or masquerade as the victim user.
- Carry out any action that the user is able to perform.
- Read any data that the user is able to access.
- Capture the user's login credentials.
- Perform virtual defacement of the web site.
- Inject trojan functionality into the web site.

Mitigation :

Preventing cross-site scripting is trivial in some cases but can be much harder depending on the complexity of the application and the ways it handles user-controllable data.

In general, effectively preventing XSS vulnerabilities is likely to involve a combination of the following measures:

- Filter input on arrival. At the point where user input is received, filter as strictly as possible based on what is expected or valid input.
- Encode data on output. At the point where user-controllable data is output in HTTP responses, encode the output to prevent it from being interpreted as active content. Depending on the output context, this might require applying combinations of HTML, URL, JavaScript, and CSS encoding.
- Use appropriate response headers. To prevent XSS in HTTP responses that aren't intended to contain any HTML or JavaScript, you can use the Content-Type and X-Content-Type-Options headers to ensure that browsers interpret the responses in the way you intend.
- Content Security Policy. As a last line of defense, you can use Content Security Policy (CSP) to reduce the severity of any XSS vulnerabilities that still occur.
- HTTP Only and Secure Cookies: Set the "HttpOnly" flag on cookies to prevent them from being accessed through JavaScript, reducing the risk of session theft.

Environmental Limitations:

NA

Lack of Details:

NA

Recommendations :

- To address these findings and enhance the security of Demo1, the following recommendations are provided:
- Remediate OWASP Top 10 Vulnerabilities: Prioritize and resolve OWASP Top 10 vulnerabilities promptly to reduce the application's attack surface.
- Enhance Authentication and Authorization: Strengthen authentication mechanisms and ensure proper authorization checks are in place.
- Implement Robust Session Management: Enhance session management controls to prevent session hijacking and fixation.
- Data Protection: Encrypt sensitive data both in transit and at rest to mitigate data exposure risks.
- Access Control: Review and enforce access controls to prevent unauthorized access to sensitive resources.
- Logging and Monitoring: Enhance logging practices to facilitate incident detection and response.

Conclusion :

The VAPT assessment has highlighted significant security vulnerabilities in the Demo1 web application , which could expose it to various cyber threats. Addressing these findings is critical to ensure the confidentiality, integrity, and availability of the application.

Next Steps :

We recommend that the team should take the following steps:

- Review the detailed findings and recommendations in subsequent sections of this report.
- Prioritize and plan remediation efforts based on the severity and potential impact of the identified vulnerabilities.
- Continuously monitor the application's security posture and conduct regular security assessments to stay ahead of emerging threats.