

Task 2: Introduction of Web Application Security

❖ Objective

Web application security refers to a variety of processes, technologies, or methods for protecting web servers, web applications, and web services such as APIs from attack by Internet-based threats. Web application security is crucial to protecting data, customers, and organizations from data theft, interruptions in business continuity, or other harmful results of cybercrime.

❖ Required Skills

Basic Web Security, Vulnerability Identification, OWASP

❖ Common attacks against web application

- ◆ Brute force
- ◆ Credential stuffing
- ◆ SQL injection and form jacking injections
- ◆ Cross-site scripting
- ◆ Cookie poisoning
- ◆ Man-in-the-middle (MITM) and man-in-the-browser attacks
- ◆ Sensitive data disclosure
- ◆ Insecure deserialization
- ◆ Session hijacking

❖ Tools

- ◆ **Zed Attack Proxy (ZAP):**

The Zed Attack Proxy (ZAP) is a free, open-source web application scanner that helps identify potential vulnerabilities in web applications. It can be used for both automated and manual testing.

- ◆ **Web Goat:**

It is a deliberately insecure application that allows interested developers just like you to test vulnerabilities commonly found in Java-based applications that use common and popular open-source components.

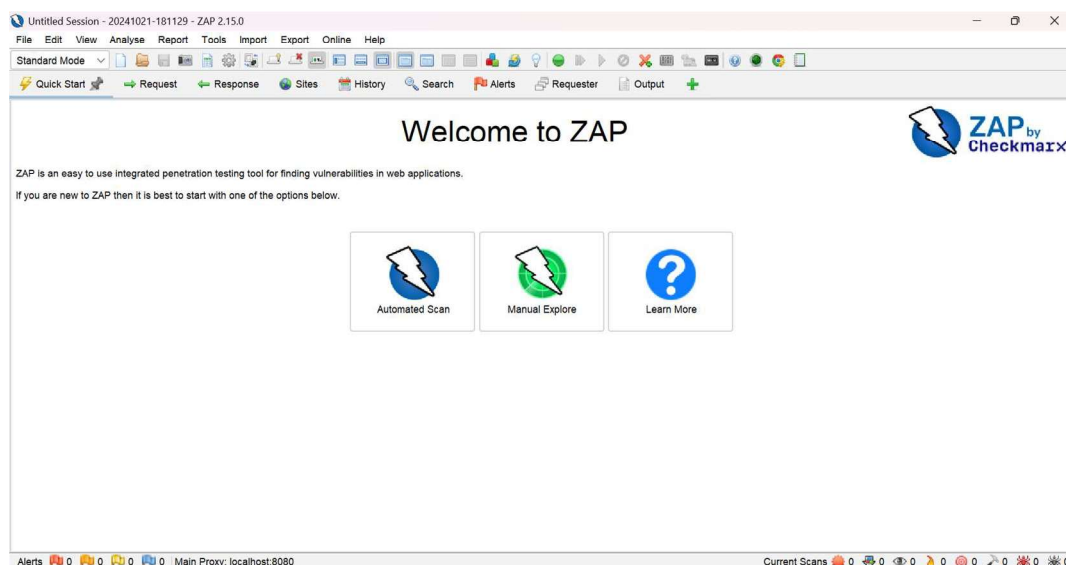
❖ Configuration

◆ Description

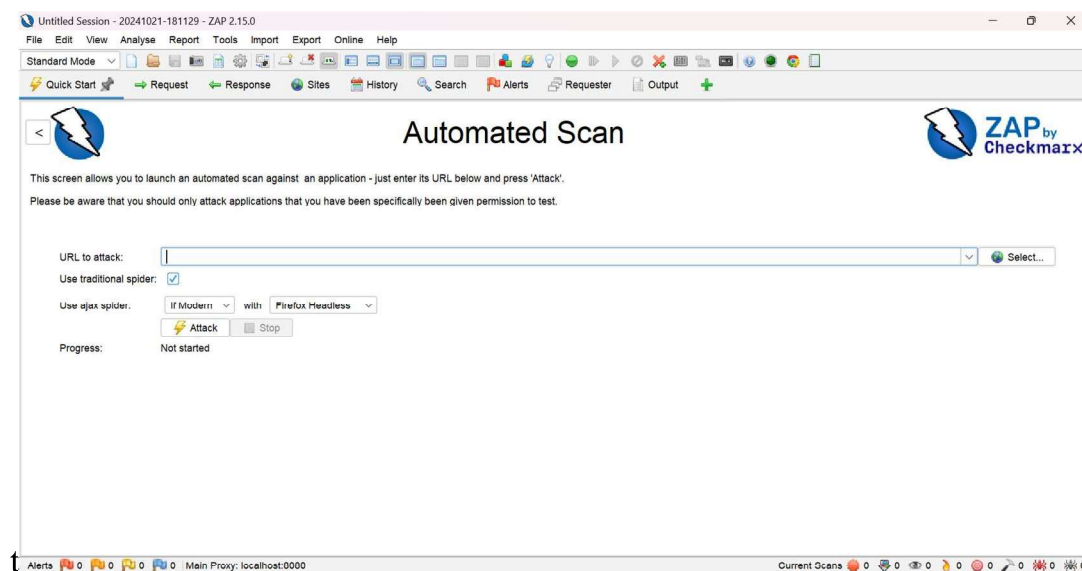
You can use any vulnerability scanner for web application testing which is zap proxy or Web goat. In my case, I used ZAP Proxy for vulnerability scanning of web application.

◆ Setup

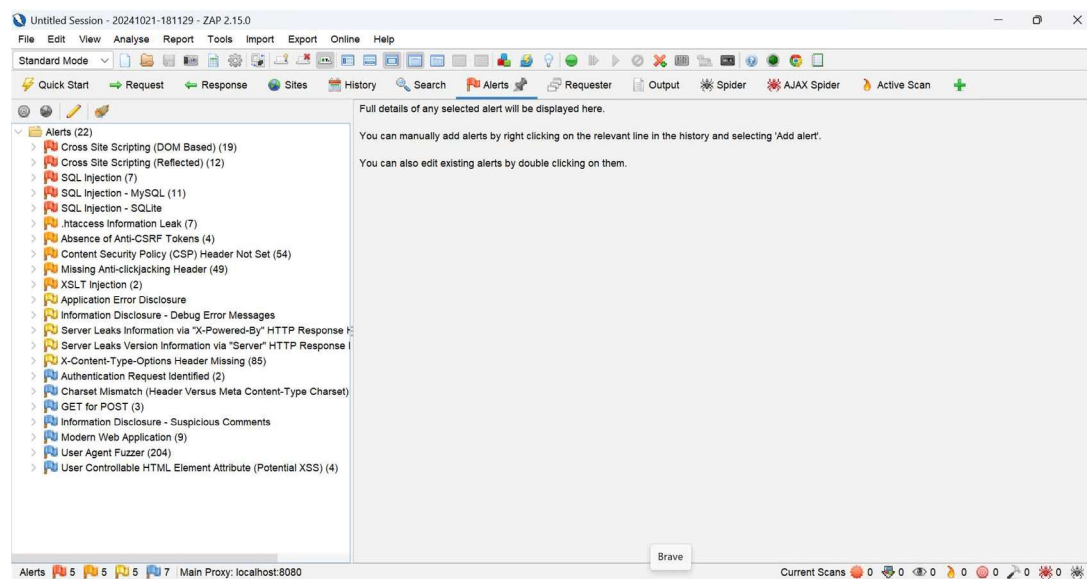
- first go to this <https://www.zaproxy.org> and download Zap proxy.
- Setup all configuration of the zap proxy
- After installation start the application and you will see this type of interface



- Click on Automated Scan and put the URL in URL of attack section of your target website and click on attack button. In my case my target website is - <http://testphp.vulnweb.com>



- After completion of scanning of given target website. You will be seen many types of vulnerability are found.



- Go to alert section. we can see there are many types of vulnerability found in targeted website like SQL, XSS, CSRF, XSLT, HT-access Information Leak etc.

❖ Explore Vulnerabilities

◆ Description of vulnerability

- SQL Injection

SQL Injection

URL: <http://testphp.vulnweb.com/AJAX/infotitle.php>

Risk: High

Confidence: Medium

Parameter: id

Attack: 9-2

Evidence:

CWE ID: 89

WASC ID: 19

Source: Active (40018 - SQL Injection)

Input Vector: Form Query

Description:

SQL injection may be possible.

Other Info:

The original page results were successfully replicated using the expression [9-2] as the parameter value
The parameter value being modified was stripped from the HTML output for the purposes of the comparison.


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 '?'


Reference:

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

- Cross site scripting injection

Cross Site Scripting (Reflected)	
JURL:	http://testphp.vulnweb.com/guestbook.php
Risk:	 High
Confidence:	Medium
Parameter:	name
Attack:	<script>alert(1);</script>
Evidence:	<script>alert(1);</script>
CWE ID:	79
WASC ID:	8
Source:	Active (40012 - Cross Site Scripting (Reflected))
Input Vector:	Form Query
Description: Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.	
Other Info:	
Solution:	
Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI	
Reference: https://owasp.org/www-community/attacks/xss/ https://cwe.mitre.org/data/definitions/79.html	


- **Cross-site request forgery (CSRF)**

Absence of Anti-CSRF Tokens	
URL:	http://testphp.vulnweb.com/search.php?test=query
Risk:	 Medium
Confidence:	Low
Parameter:	
Attack:	
Evidence:	<form action="search.php?test=query" method="post">
CWE ID:	352
WASC ID:	9
Source:	Passive (10202 - Absence of Anti-CSRF Tokens)
Input Vector:	
Description: No Anti-CSRF tokens were found in a HTML submission form. A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature	
Other Info: No known Anti-CSRF token [anticsrf, CSRFToken, __RequestVerificationToken, csrfmiddlewaretoken, authenticity_token, OWASP_CSRFTOKEN, anoncsrf, csrf_token, __csrf, __csrfSecret, __csrf_magic, CSRF, _token, _csrf_token] was found in the following HTML form: [Form 1: "goButton" "searchFor"].	
Solution: Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard.	
Reference: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html https://cwe.mitre.org/data/definitions/352.html	

- **XSLT Injection**

XSLT Injection

URL: <http://testphp.vulnweb.com/showimage.php?file=%3Cxml%3Avalue-of+select%3D%22document%28%27http%3A%2F%2Ftestphp.vulnweb.com%3A2%27%29%22%2F%3E&size=160>

Risk:  Medium

Confidence: Medium

Parameter: file

Attack: `<xsl:value-of select="document('http://testphp.vulnweb.com:22')"/>`

Evidence: failed to open stream

CWE ID: 91

WASC ID: 23

Source: Active (90017 - XSLT Injection)

Input Vector: URL Query String

Description:

Injection using XSL transformations may be possible, and may allow an attacker to read system information, read and write files, or execute arbitrary code.

Other Info:

Port scanning may be possible.

Solution:

Sanitize and analyze every user input coming from any client-side.


Reference:

<https://www.contextis.com/blog/xslt-server-side-injection-attacks>

- **Information Leak**

.htaccess Information Leak

URL: http://testphp.vulnweb.com/Mod_Rewrite_Shop/.htaccess

Risk:  Medium

Confidence: Medium

Parameter:

Attack:

Evidence: HTTP/1.1 200 OK

CWE ID: 94

WASC ID: 14

Source: Active (40032 - .htaccess Information Leak)

Alert Reference: 40032-1

Input Vector:

Description:

.htaccess files can be used to alter the configuration of the Apache Web Server software to enable/disable additional functionality and features that the Apache Web Server software has to offer.

Other Info:









Solution:

Ensure the .htaccess file is not accessible.

Reference:


https://developer.mozilla.org/en-US/docs/Learn/Server-side/Apache_Configuration_htaccess
<https://httpd.apache.org/docs/2.4/howto/htaccess.html>

Alert Tags:

Current Scans  0  0  0  0  0  0  0  0

Server Leaks Version Information via "Server" HTTP Response Header Field

URL: <http://testphp.vulnweb.com/sitemap.xml>

Risk:  Low

Confidence: High

Parameter:

Attack:

Evidence: nginx/1.19.0

CWE ID: 200

WASC ID: 13

Source: Passive (10036 - HTTP Server Response Header)

Input Vector:

Description:

The web/application server is leaking version information via the "Server" HTTP response header. Access to such information may facilitate attackers identifying other vulnerabilities your web/application server is subject to.

Other Info:

Solution:

Ensure that your web server, application server, load balancer, etc. is configured to suppress the "Server" header or provide generic details.


Reference:

<https://httpd.apache.org/docs/current/mod/core.html#servertokens>
[https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff648552\(v=pandp.10\)](https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff648552(v=pandp.10))
<https://www.troyhunt.com/shhh-dont-let-your-response-headers/>

Alert Tags:

Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)

URL: <http://testphp.vulnweb.com>

Risk:  Low

Confidence: Medium

Parameter:

Attack:

Evidence: X-Powered-By: PHP/5.6.40-38+ubuntu20.04.1+deb.sury.org+1

CWE ID: 200

WASC ID: 13

Source: Passive (10037 - Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s))

Input Vector:

Description:

The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to.

Other Info:

Solution:


Ensure that your web server, application server, load balancer, etc. is configured to suppress "X-Powered-By" headers.

Reference:









https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/01-Information_Gathering/08-Fingerprint_Web_Application_Framework
<https://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html>

Alert Tags:

Key	Value

Current Scans 

- There are many low and informational risk vulnerability

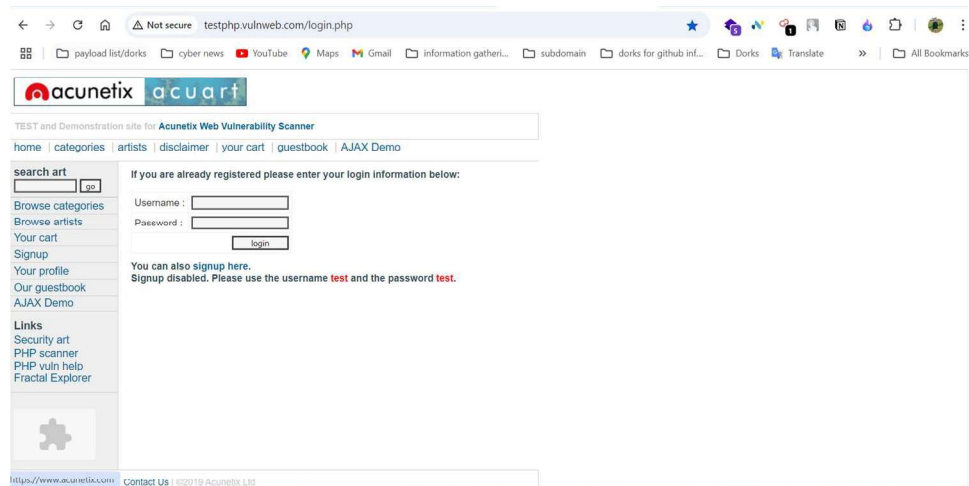
- >  X-Content-Type-Options Header Missing (85)
- >  Authentication Request Identified (2)
- >  Charset Mismatch (Header Versus Meta Content-Type Charset)
- >  GET for POST (3)
- >  Information Disclosure - Suspicious Comments
- >  Modern Web Application (9)
- >  User Agent Fuzzer (204)
- >  User Controllable HTML Element Attribute (Potential XSS) (4)

◆ Manually Attack

- For SQL injection

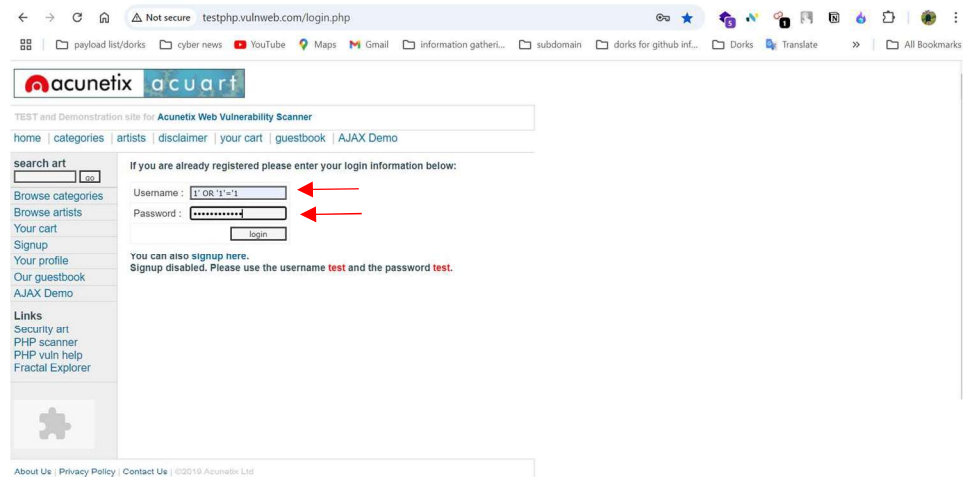
1. SQL injection (SQLi) is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. This can allow an attacker to view data that they are not normally able to retrieve. This might include data that belongs to other users, or any other data that the application can access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior. In some situations, an attacker can escalate a SQL injection attack to compromise the underlying server or other back-end infrastructure. It can also enable them to perform denial-of-service attacks.

2. Visit the <http://testphp.vulnweb.com> and go for signup. There will be seeing this type of interface

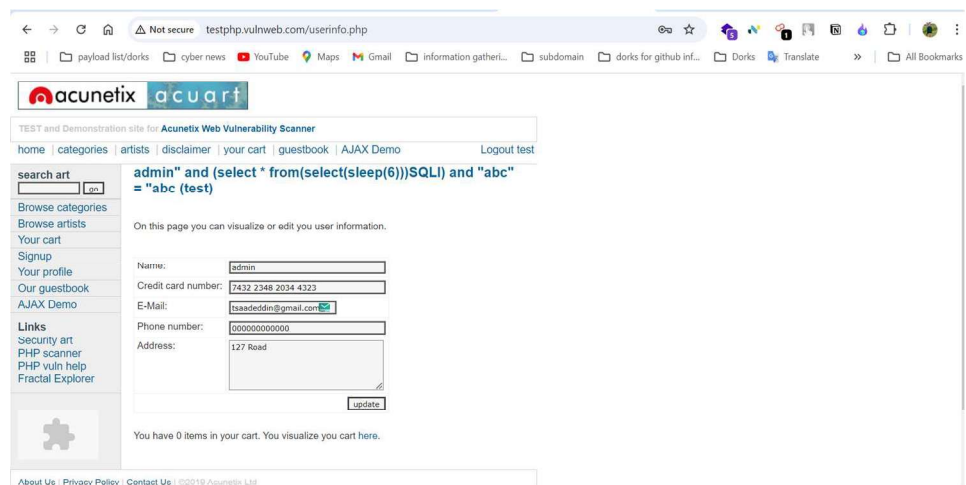


3. An attacker might get access to usernames and passwords in a database by simply inserting payload into the username and password text box.

Payload used: 1' OR '1'='1



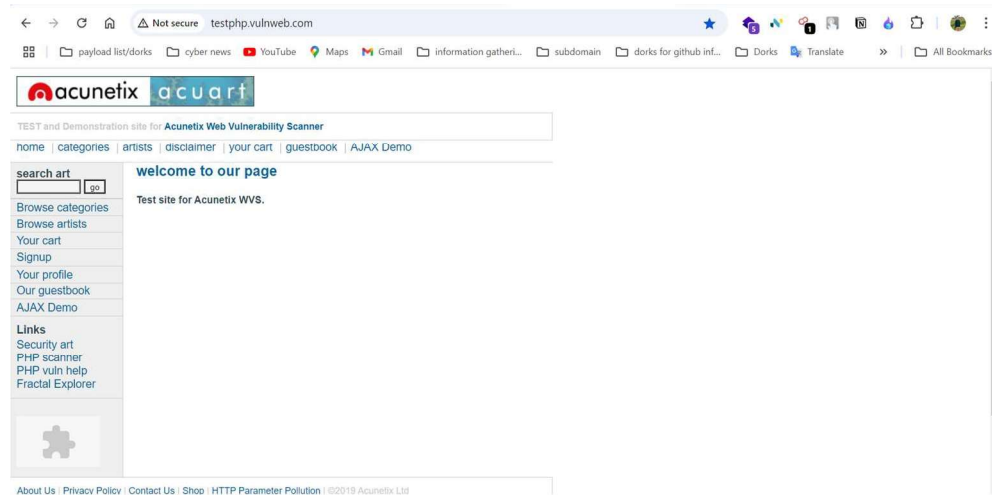
4. After inserting the payload in the username and password section. Then we got the access of admin panel of the **database**.



- **for XSS injection**

1. 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 can 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 the application's functionality and data.

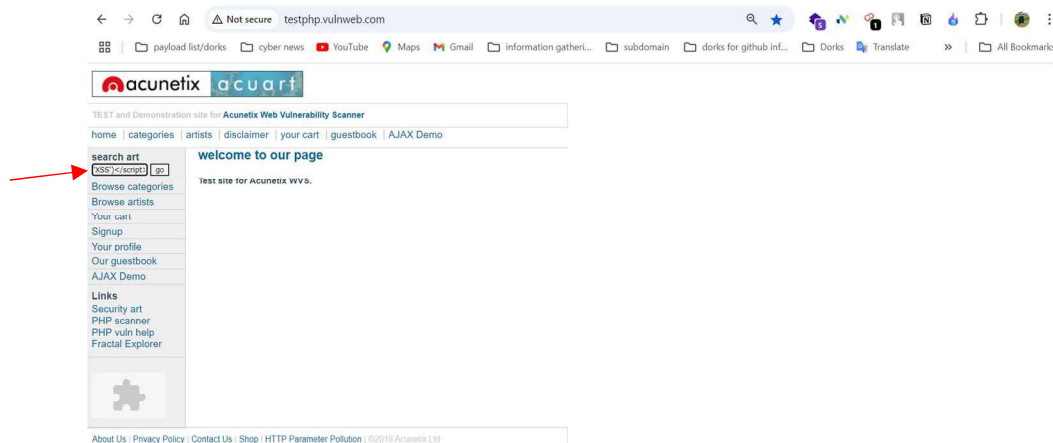
- Again, visit the <http://testphp.vulnweb.com> and go in the search bar. There will be seeing this type of interface.



- The attacker injects a payload into the website's input functionality. victim requests the web page from the web server. The web server serves the victim's browser the page with attacker's payload as part of the HTML body. The victim's browser executes the malicious script contained in the HTML body. In this case, it sends the victim's cookie to the attacker's server. The attacker now simply needs to extract the victim's cookie when the HTTP request arrives at the server. The attacker can now use the victim's stolen cookie for impersonation.

Payload used: `<script>alert('XSS')</script>`

It will be executed on server and after execution the pop-up will be generated and then we can analyze this website vulnerable from XSS or not.



- In my case, my target is vulnerable from cross-site scripting



❖ Impact and how mitigate vulnerabilities

Impact:

◆ SQL Injection -

- **Data Theft:** Attackers can extract sensitive information such as usernames, passwords, credit card numbers, and personal details from a database.
- **Data Modification or Deletion:** Unauthorized manipulation or deletion of data can lead to significant data loss or damage, affecting the integrity and reliability of the information.
- **System Takeover:** By gaining administrative access, attackers can control systems, leading to further malicious activities like additional attacks, malware installation, or unauthorized changes.
- **Financial Loss:** The costs of SQL injection attacks can be substantial, including direct expenses for system restoration and data recovery, as well as indirect losses from disrupted business operations and lost revenue.
- **Regulatory and Legal Consequences:** Businesses may face financial penalties or legal action due to data breaches, particularly those handling sensitive information, such as financial institutions or healthcare providers.
- **Reputation Damage:** A successful attack can severely damage a company's reputation, leading to long-term harm to future growth and profitability.
- **Operational Disruption:** Downtime caused by attacks can result in lost revenue and customer frustration, further impacting the business's reputation.

◆ XSS Injection –

- **Account Hijacking:** One of the most common XSS attack vectors is to hijack legitimate user accounts by stealing their session cookies. This allows attackers to impersonate victims and access any sensitive information or functionality on their behalf.
- **Stealing credentials:** A practical attack vector for XSS is to use HTML and JavaScript to steal user credentials, instead of their cookies. This can be done by cloning the login page of the web application and then using the XSS vulnerability to serve it to the victims.
- **Sensitive Data:** Another powerful attack vector for XSS is to use it to exfiltrate sensitive data (for example personal identifiable information or cardholder data) or to perform unauthorized operations, such as siphoning funds. The following example shows how it is possible to use the XML Http Request object to force the victim to send money to another user of the web application:
- **Drive-by Downloads:** Let us now take the case of a simple website, which does not hold any sensitive information, for example a company presentation website with static pages and a simple search function. If an attacker can use the website to gain control of an organization's client computers, it still represents a significant risk to the business.

◆ CSRF injection

- Data Theft
- Unauthorized fund transfers,
- Damaged client relationships
- Credentials changing

Mitigation:

◆ SQL Injection

- Implement Input Validation and Sanitization
- Use Escaping for User Input
- Utilize Parameterized Statements (Prepared Statements)
- Incorporate Stored Procedures

- Conduct Continuous Scanning and Penetration Testing
- Adopt the Least Privilege Principle
- Deploy Web Application Firewalls (WAF)

◆ XSS Injection

- Scrutinize links
- Sanitize data
- Review and test code
-

◆ CSRF Injection

- First, check if your framework has built-in CSRF protection and use it
- If the framework does not have built-in CSRF protection, add CSRF tokens to all state changing requests (requests that cause actions on the site) and validate them on the backend
- Stateful software should use the synchronizer token pattern
- Stateless software should use double submit cookies
- If an API-driven site can't use <form> tags, consider using custom request headers
- Implement at least one mitigation from Defense in Depth Mitigations section
- Same Site Cookie Attribute can be used for session cookies but be careful to NOT set a cookie specifically for a domain. This action introduces a security vulnerability because all subdomains of that domain will share the cookie, and this is particularly an issue if a subdomain has a CNAME to domains not in your control.
- Consider implementing user interaction based protection for highly sensitive operations
- Consider verifying the origin with standard headers
- Do not use GET requests for state changing operations.

