CROSS SITE SCRIPTING (XSS) ATTACKS

Injection attacks

OWASP
The Open Web Application Security Project

OWASP
The Open Web Application Security Project

OWASP
The Open Web Application Security Project

TakeAways

- What is Cross-site Scripting
- Different types of Cross-Site Scripting
- Impact of Cross-Site Scripting
- Ways to identify XSS vulnerabilities
- Preventing Cross-Site Scripting attacks

OWASP
The Open Web Application Security Project

What is XSS

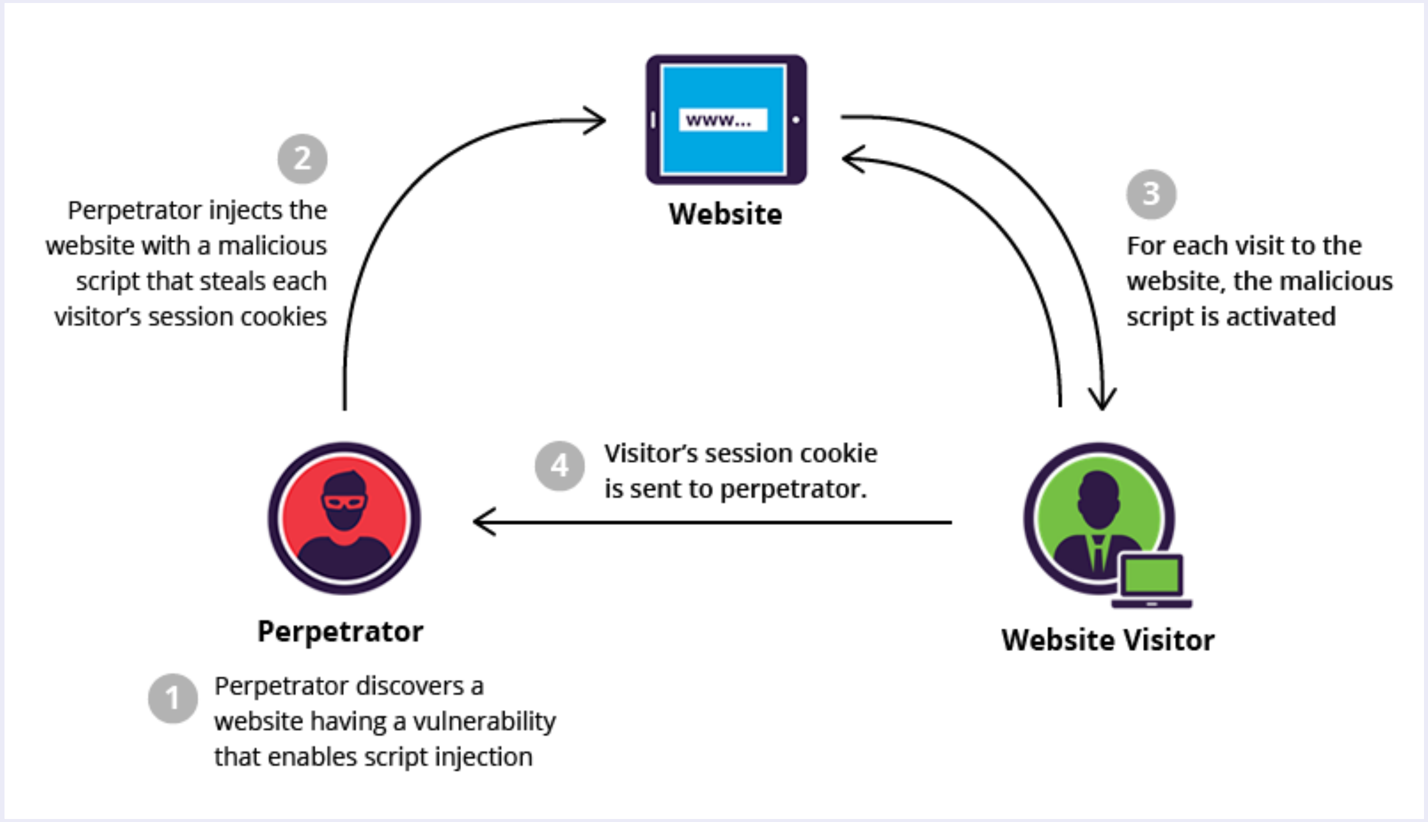
Cross site scripting (**XSS**) is a common attack vector that injects malicious code into a vulnerable web application.

XSS differs from other web attack vectors (e.g., **SQL injections**), in that it does not directly target the application itself. Instead, the users of the web application are the ones at risk.

A successful cross site scripting attack can have devastating consequences for an online business's reputation and its relationship with its clients.

OWASP
The Open Web Application Security Project

What is XSS


Website
www...
2
Perpetrator injects the website with a malicious script that steals each visitor's session cookies
3
For each visit to the website, the malicious script is activated
4
Visitor's session cookie is sent to perpetrator.
Perpetrator
Website Visitor
1
Perpetrator discovers a website having a vulnerability that enables script injection

Types of XSS

There are mainly **three** different types of Cross-site Scripting vulnerability;

- Reflected XSS

  A reflected XSS vulnerability happens when the user input from a URL or POST data is reflected on the page without being stored.

- Persistent or Stored XSS

  Stored Cross-site scripting vulnerabilities happens when the payload is saved, for example in a database and then is executed when a user opens the page. Stored cross-site scripting is very dangerous for a number of reasons

- DOM-based XSS

  The DOM Based XSS vulnerability happens in the DOM (Document Object Model) instead of part of the HTML.

OWASP
The Open Web Application Security Project

Types of XSS

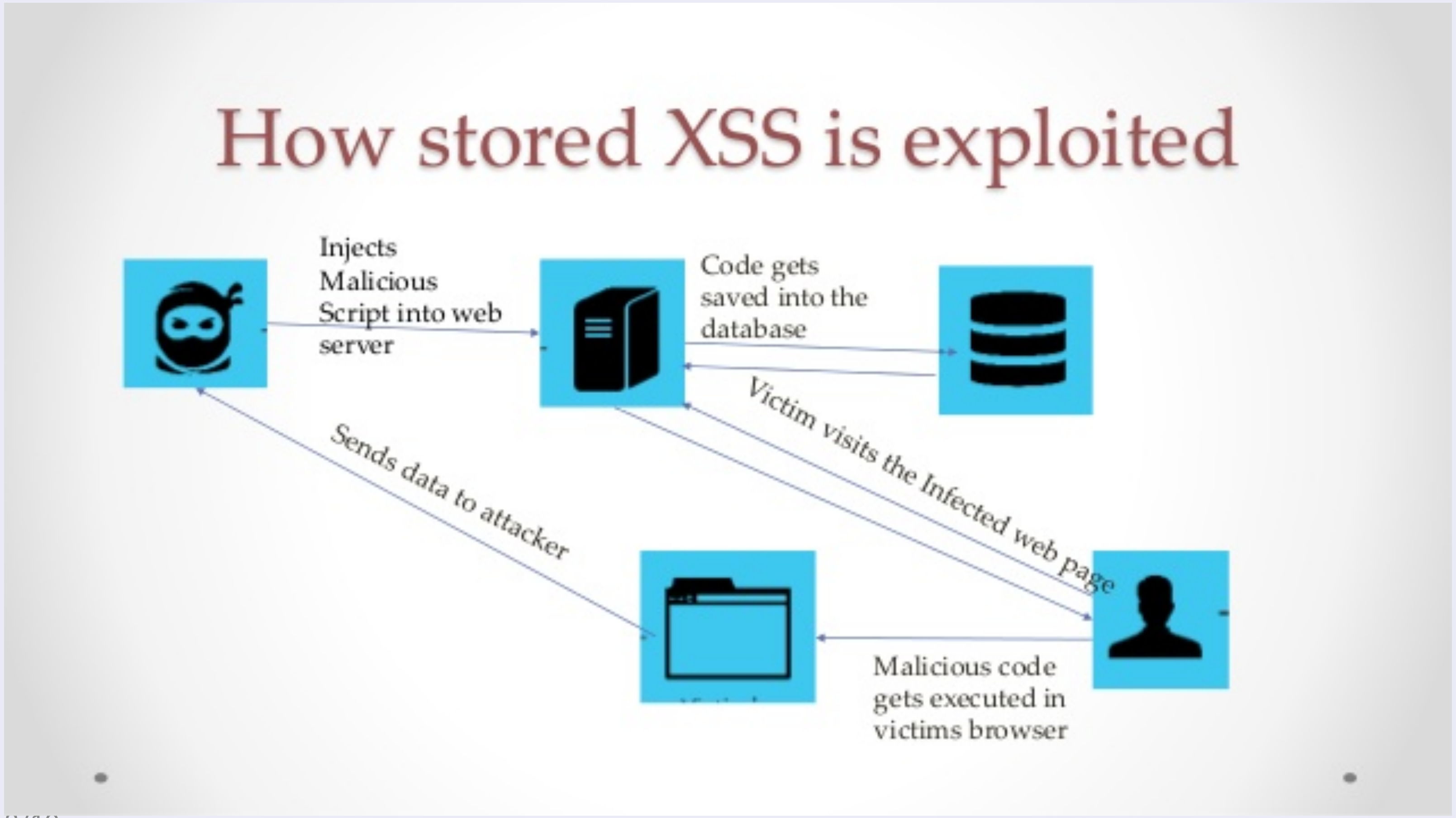
For years, most people thought of these (**Stored**, **Reflected**, **DOM**) as **three** different types of XSS, but in reality, they **overlap**. You can have both Stored and Reflected DOM Based XSS.

You can also have Stored and Reflected Non-DOM Based XSS too, but that's confusing, so to help clarify things, starting about mid 2012, the research community proposed and started using two new terms to help organize the types of XSS that can occur:

1. **Server XSS**
2. **Client XSS**

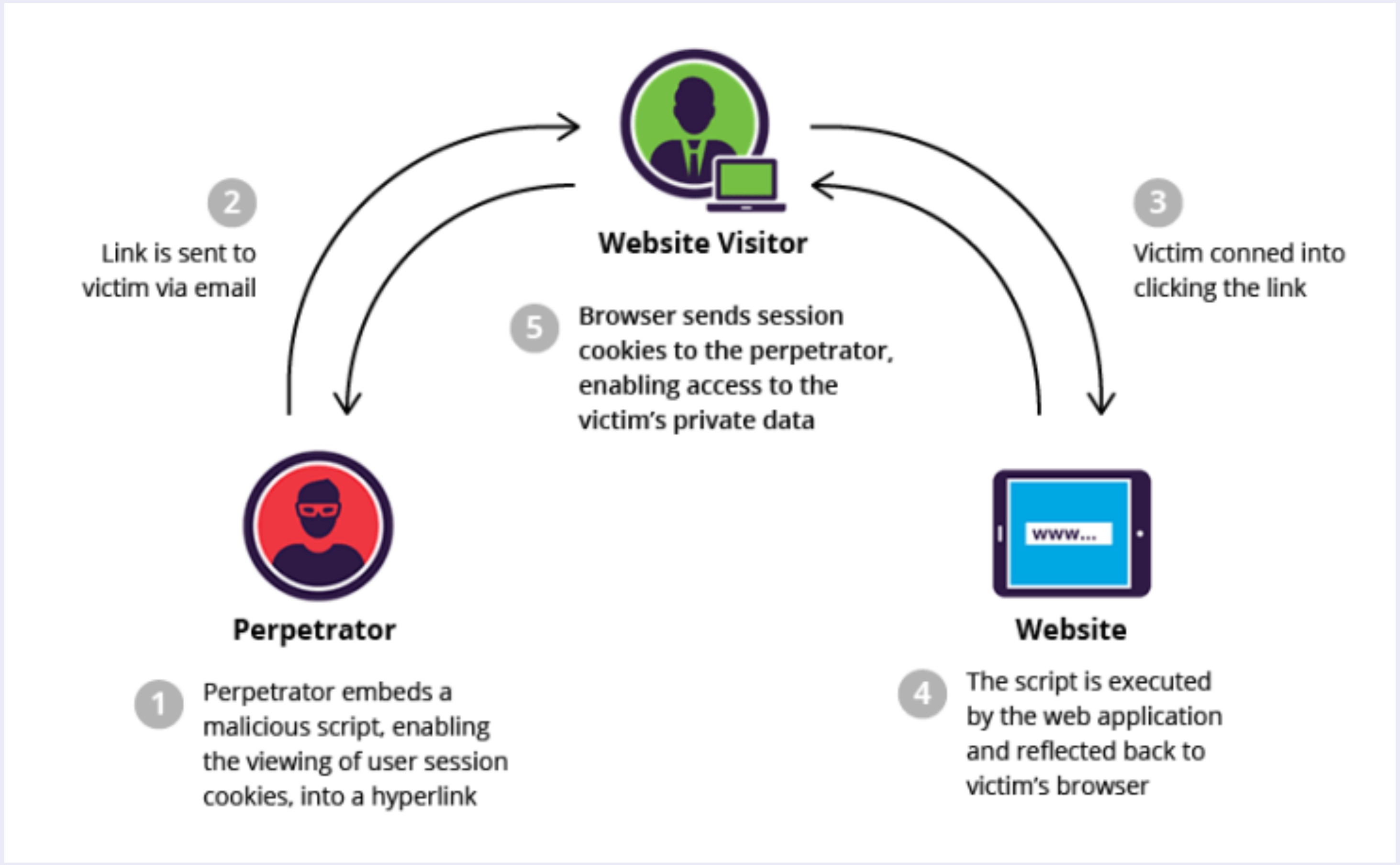
OWASP
The Open Web Application Security Project

Server XSS


How stored XSS is exploited
Injects Malicious Script into web server
Code gets saved into the database
Victim visits the Infected web page
Sends data to attacker
Malicious code gets executed in victims browser

OWASP
The Open Web Application Security Project

Client XSS


Website Visitor
2
Link is sent to victim via email
3
Victim conned into clicking the link
5
Browser sends session cookies to the perpetrator, enabling access to the victim's private data
www...
Perpetrator
Website
1
Perpetrator embeds a malicious script, enabling the viewing of user session cookies, into a hyperlink
4
The script is executed by the web application and reflected back to victim's browser

Impact of XSS

The impact of an exploited XSS vulnerability varies a lot. It ranges from

- Redirection
- Session Hijacking
- Cross Site Request forgery
- Keylogging
- Phishing

By exploiting a cross-site scripting vulnerability an attacker can impersonate the victim and take over the account. If the victim has administrative rights it might even lead to code execution on the server, depending on the application and the privileges of the account

Ways to identify & verify XSS vulnerabilities

Cross-site Scripting vulnerabilities can be identified in 2 ways namely;

- Static Analysis (Source code review)
- Dynamic analysis (Fuzzing)

Static Analysis Tools

- OWASP WAP - Web Application Protection Project
- RIPS - A static source code analyser
- Codacy: Automated code reviews & code analytics

Dynamic Analysis Tools

- Burp suite
- Hack bar Firefox addon or burp addon
- Automated vulnerability scanner (eg. Arachni)

OWASP
The Open Web Application Security Project

Brace your self demo is starting

Everybody is interested in something

Preventing Cross-Site Scripting

Prevention?

- **Never trust user input**
- **Never trust user input**
- **Never trust user input**
- **Never trust user input**
- **Never trust user input**
- **Never trust user input**
- **Never trust user input**
- **Never trust user input**
- **Never trust user input**

OWASP
The Open Web Application Security Project

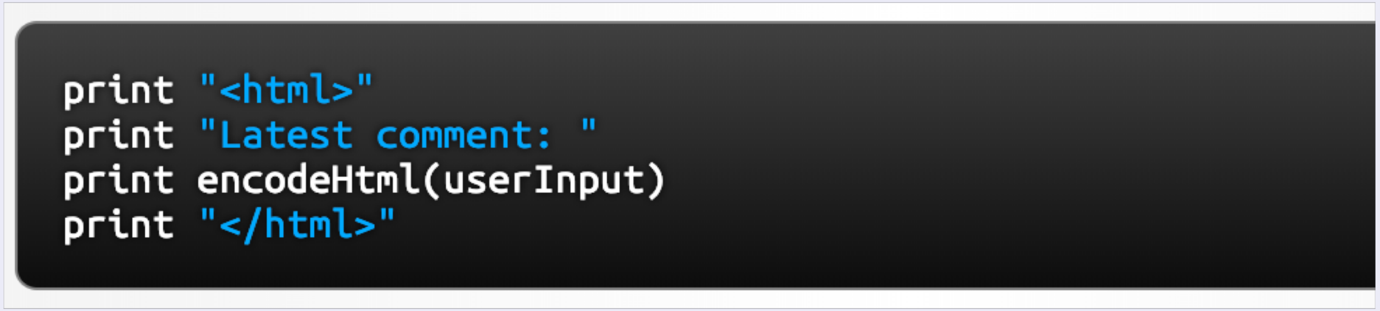
Preventing Cross-Site Scripting

Recall that an XSS attack is a type of code injection: user input is mistakenly interpreted as malicious program code. In order to prevent this type of code injection, secure input handling is needed. For a web developer, there are two fundamentally different ways of performing secure input handling:

- **Encoding**, which escapes the user input so that the browser interprets it only as data, not as code.
- **Validation**, which filters the user input so that the browser interprets it as code without malicious commands.

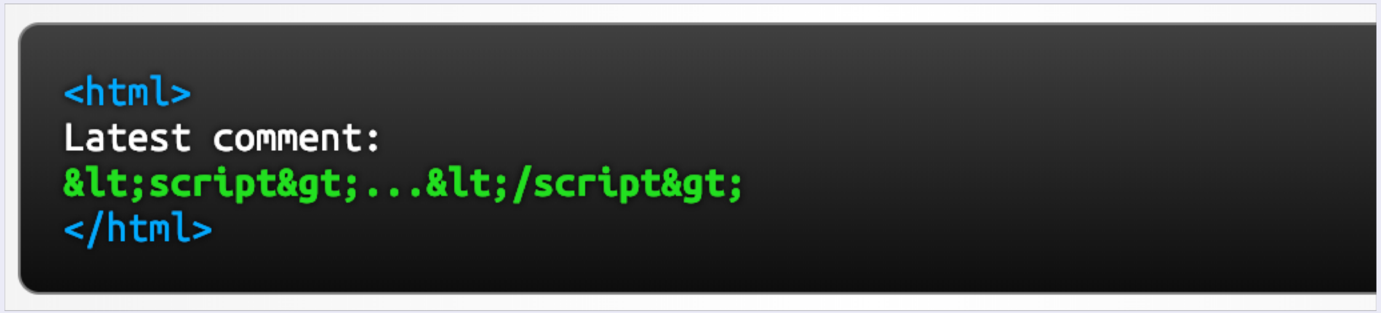
Preventing XSS - Encoding

Encoding is the act of escaping user input so that the browser interprets it only as data, not as code. The following pseudocode is an example of how user input could be encoded using HTML escaping



```
print "<html>"
print "Latest comment: "
print encodeHtml(userInput)
print "</html>"
```

If the user input were the string <script>...</script>, the resulting HTML would be as follows



```
<html>
Latest comment:
&lt;script&gt;...&lt;/script&gt;
</html>
```

Preventing XSS - Validating

Validation is the act of filtering user input so that all malicious parts of it are removed, without necessarily removing all code in it. One of the most recognizable types of validation in web development is allowing some HTML elements (such as <em> and <strong>) but disallowing others (such as <script>).

There are two main characteristics of validation that differ between implementations:

Classification strategy: User input can be classified using either blacklisting or whitelisting.

Validation outcome: User input identified as malicious can either be rejected or sanitised.

OWASP
The Open Web Application Security Project

XSS is not the user's problem like any other security vulnerability. If it is affecting your users, it affects you.

I hope that you found this talk useful

References

https://www.netsparker.com
https://www.acunetix.com
https://excess-xss.com/
https://www.incapsula.com
https://www.owasp.org
https://www.google.com