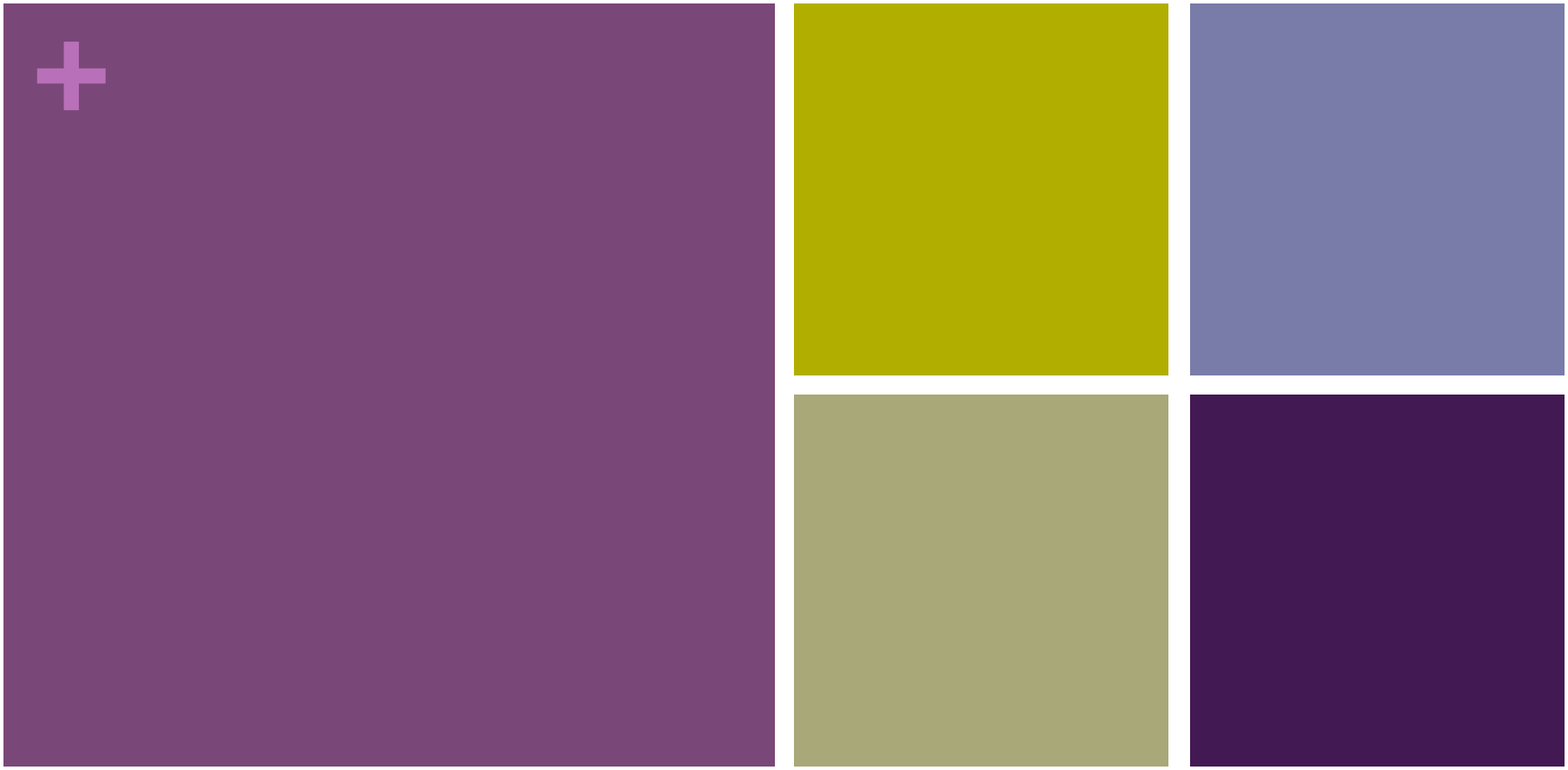
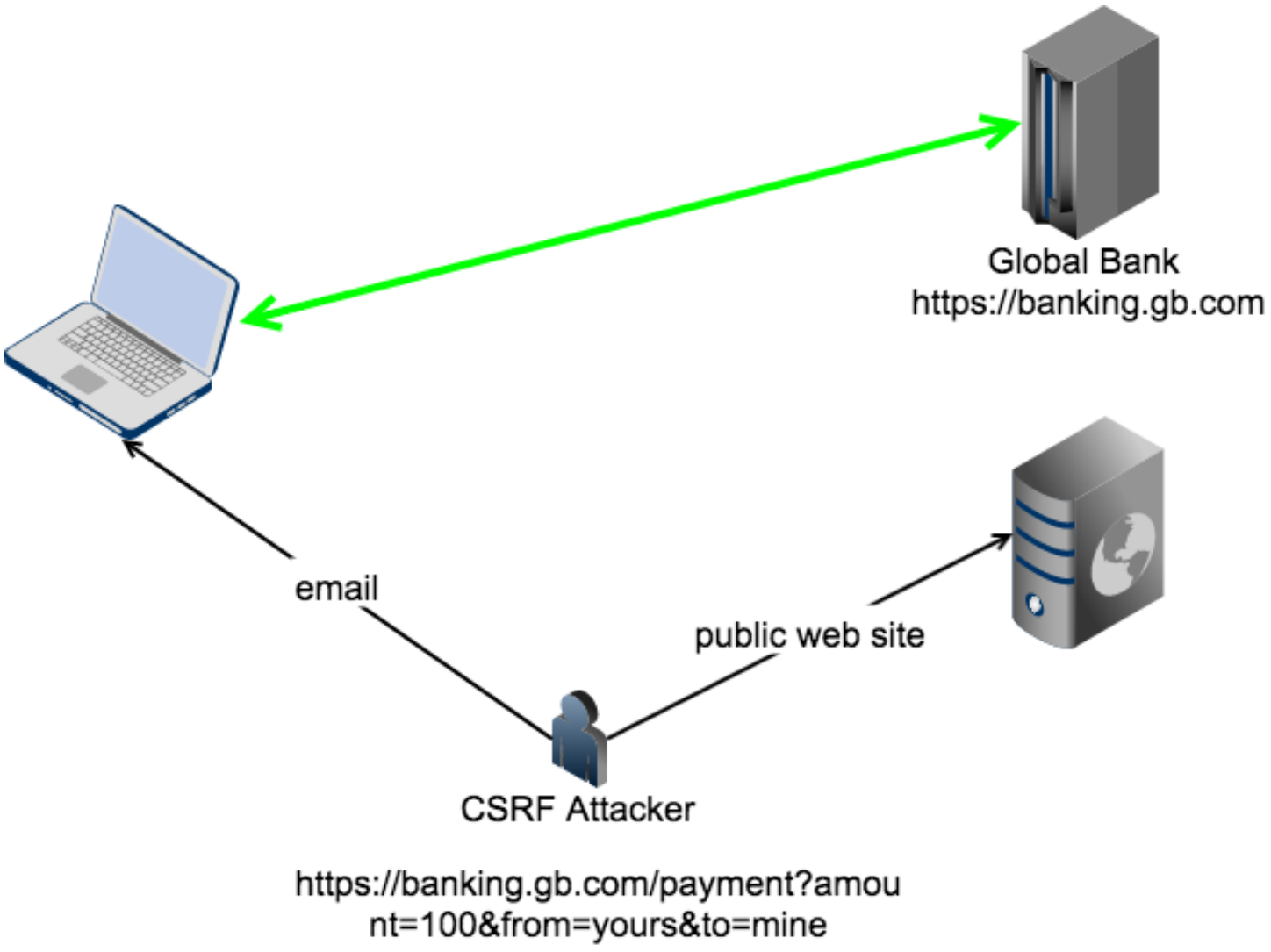


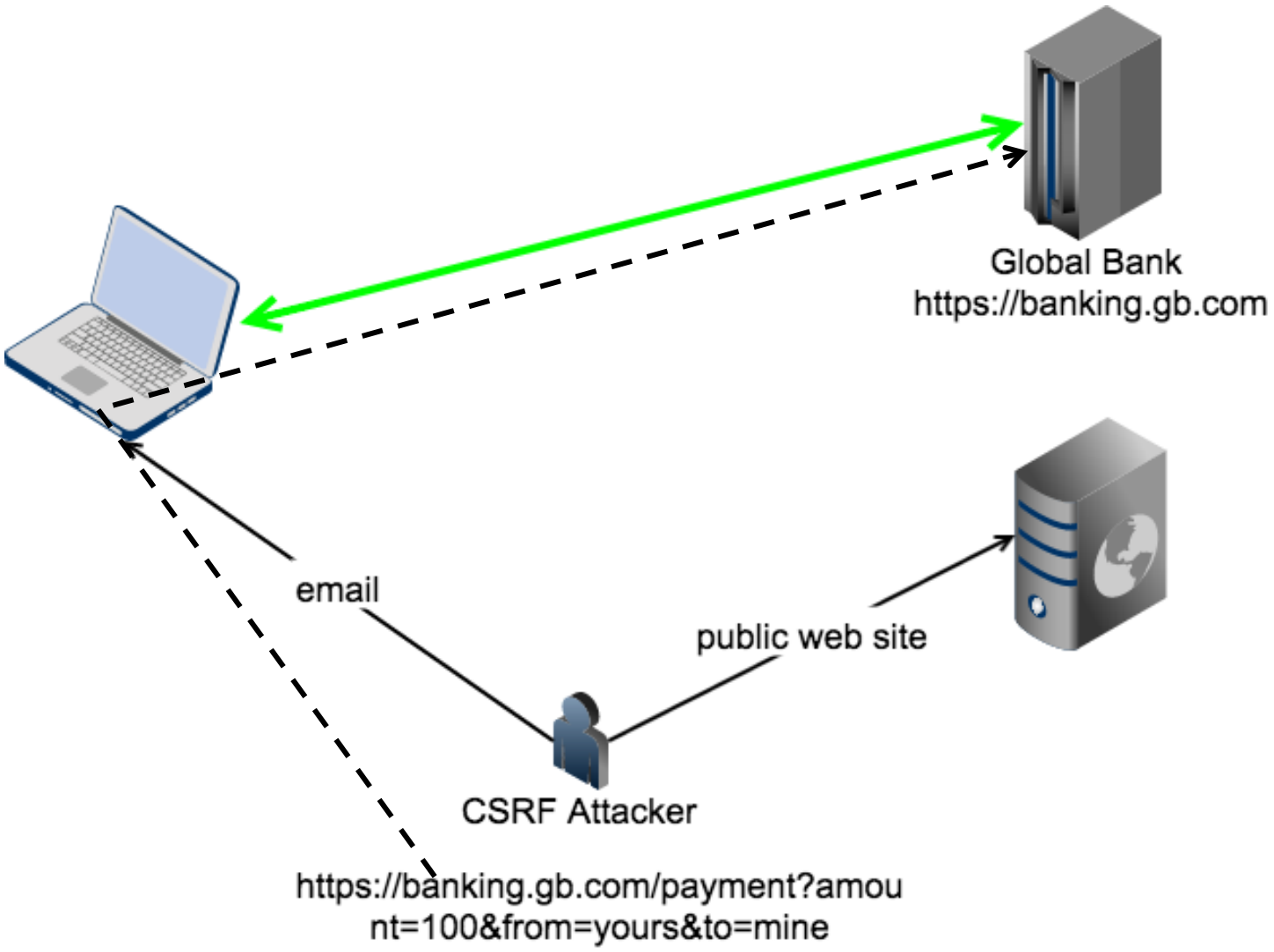
Fixing CSRF Effectively

Lu Zhao
lzhao@netsuite.com

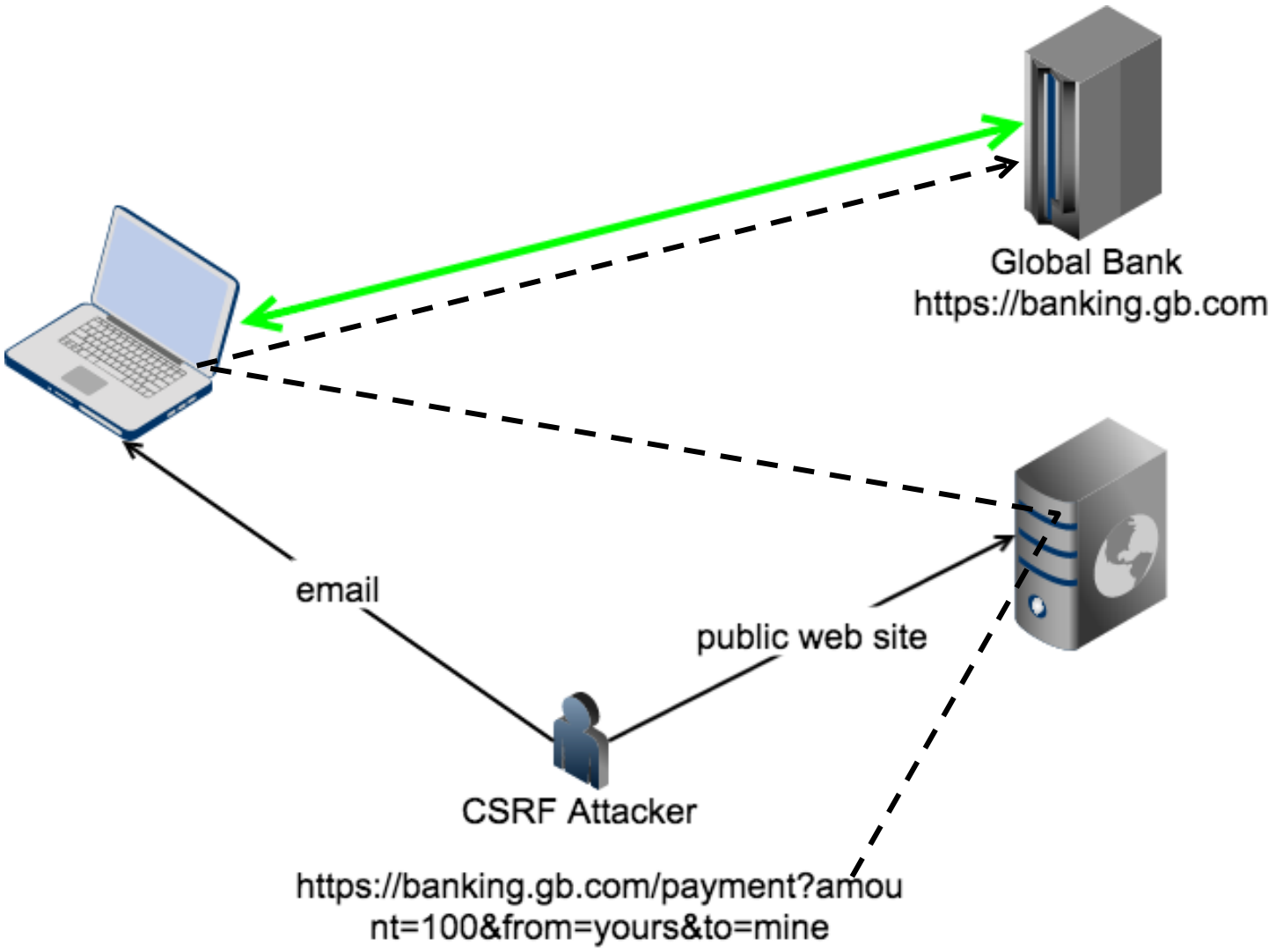
CSRF – Cross Site Request Forgery



CSRF – Cross Site Request Forgery



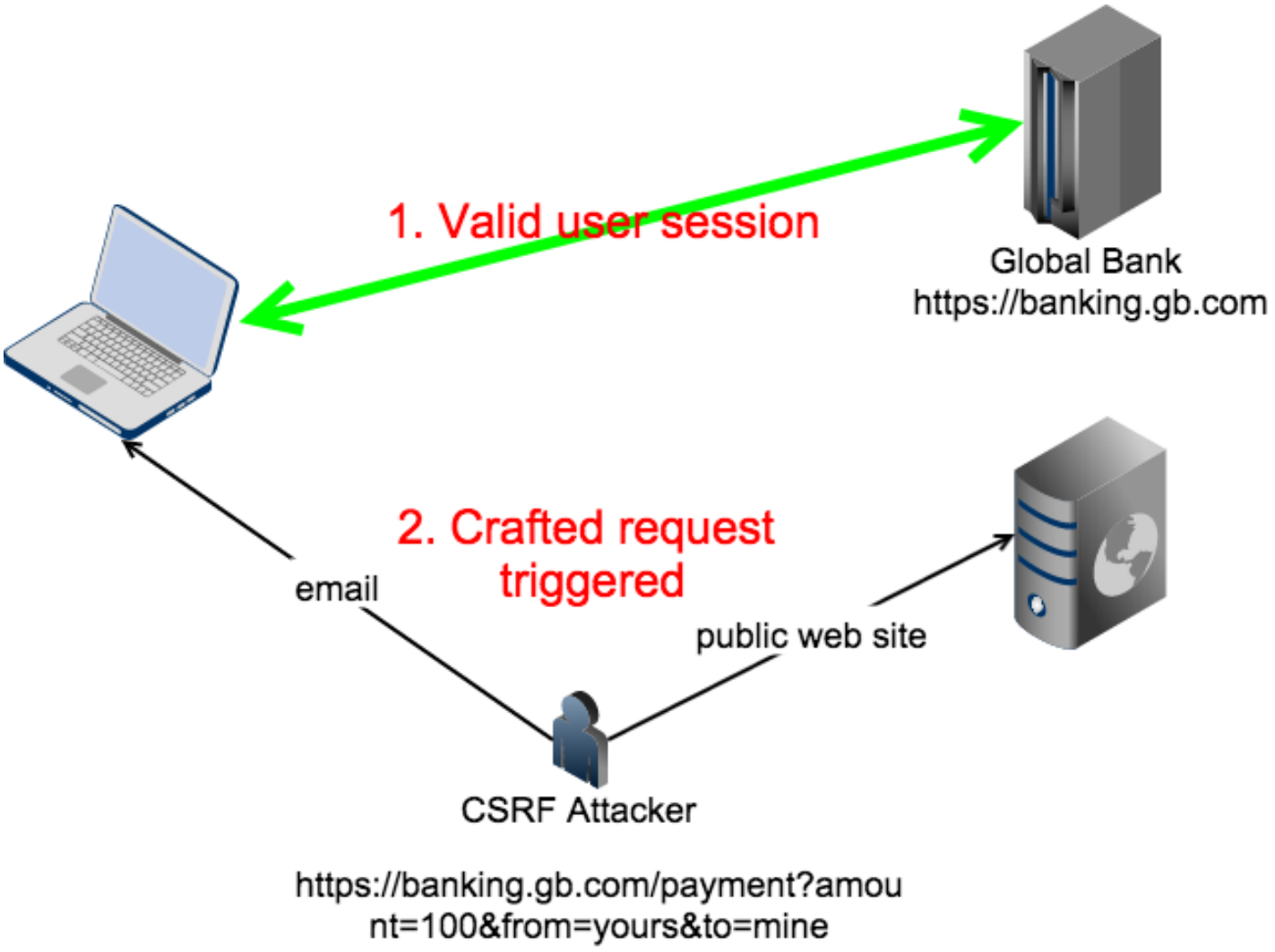
CSRF – Cross Site Request Forgery



CSRF

- The malicious request is not made the user,
  but the server does not know this,
  thinks that it is made by the user,
  and happily services the request.

Two Conditions



Trigger Request

- Attacker sends fishing email
- Attacker posts the request to other sites
- Using scripts
  - Many automation and programming tricks

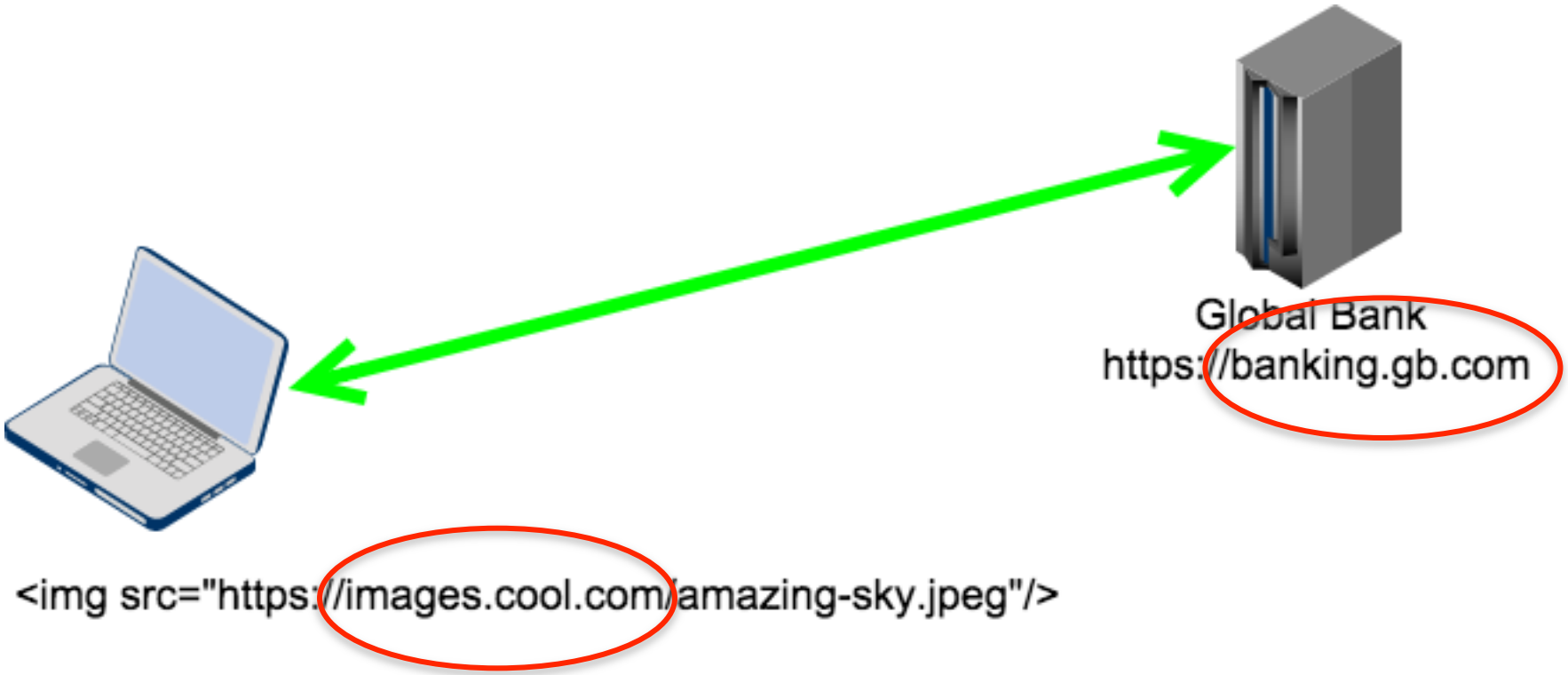
Good User Behavior

- Logoff immediately after using a Web application
- Do NOT allow your browser to save username/passwords
- Do NOT allow site to remember your login
- Do NOT user the same browser to access sensitive applications and surf the Internet freely
- Use No-Script like plugins to prevent scripts from submitting POST requests automatically

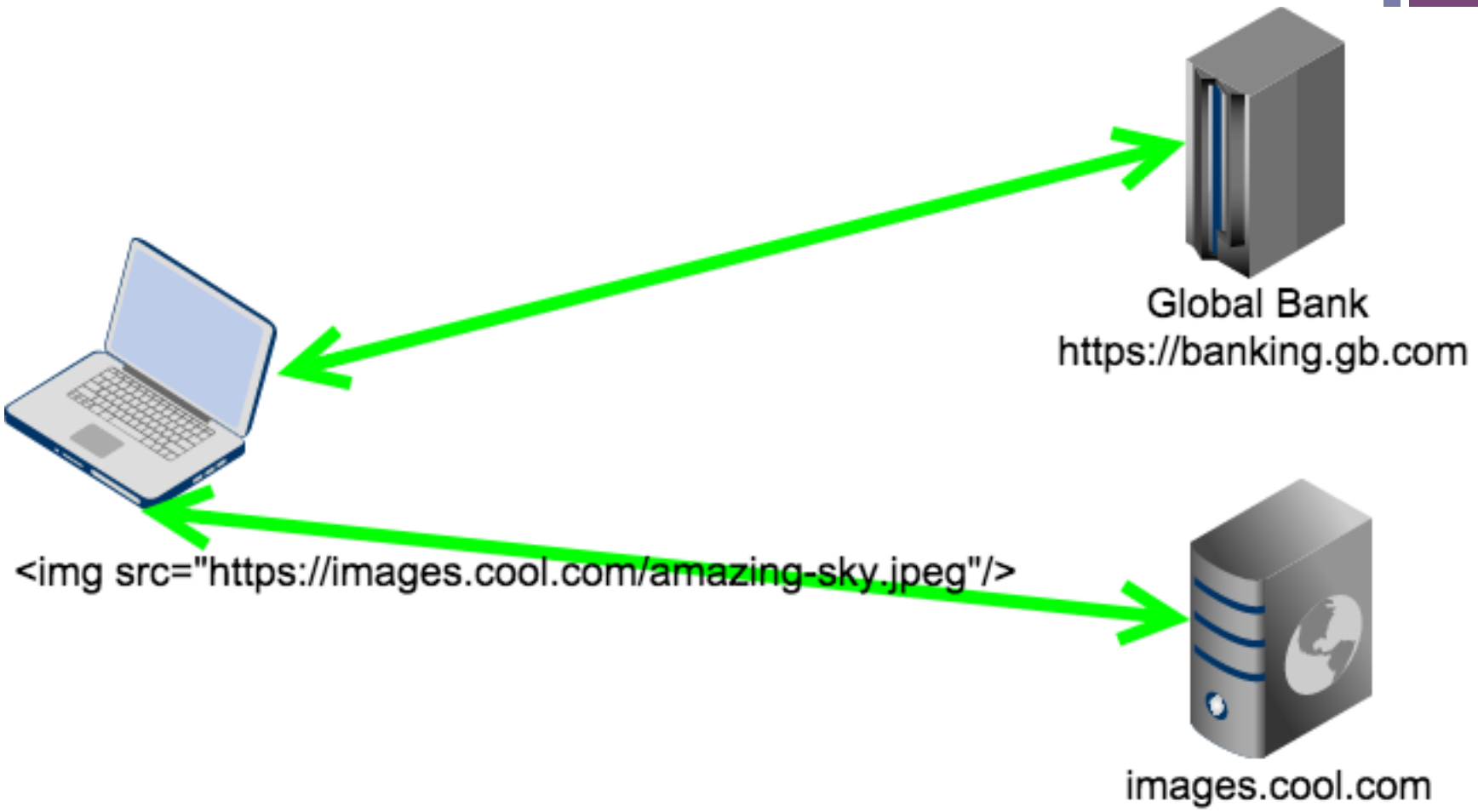
from OWASP

Can a Web site count on good user behaviors?

Why Can CSRF Happen



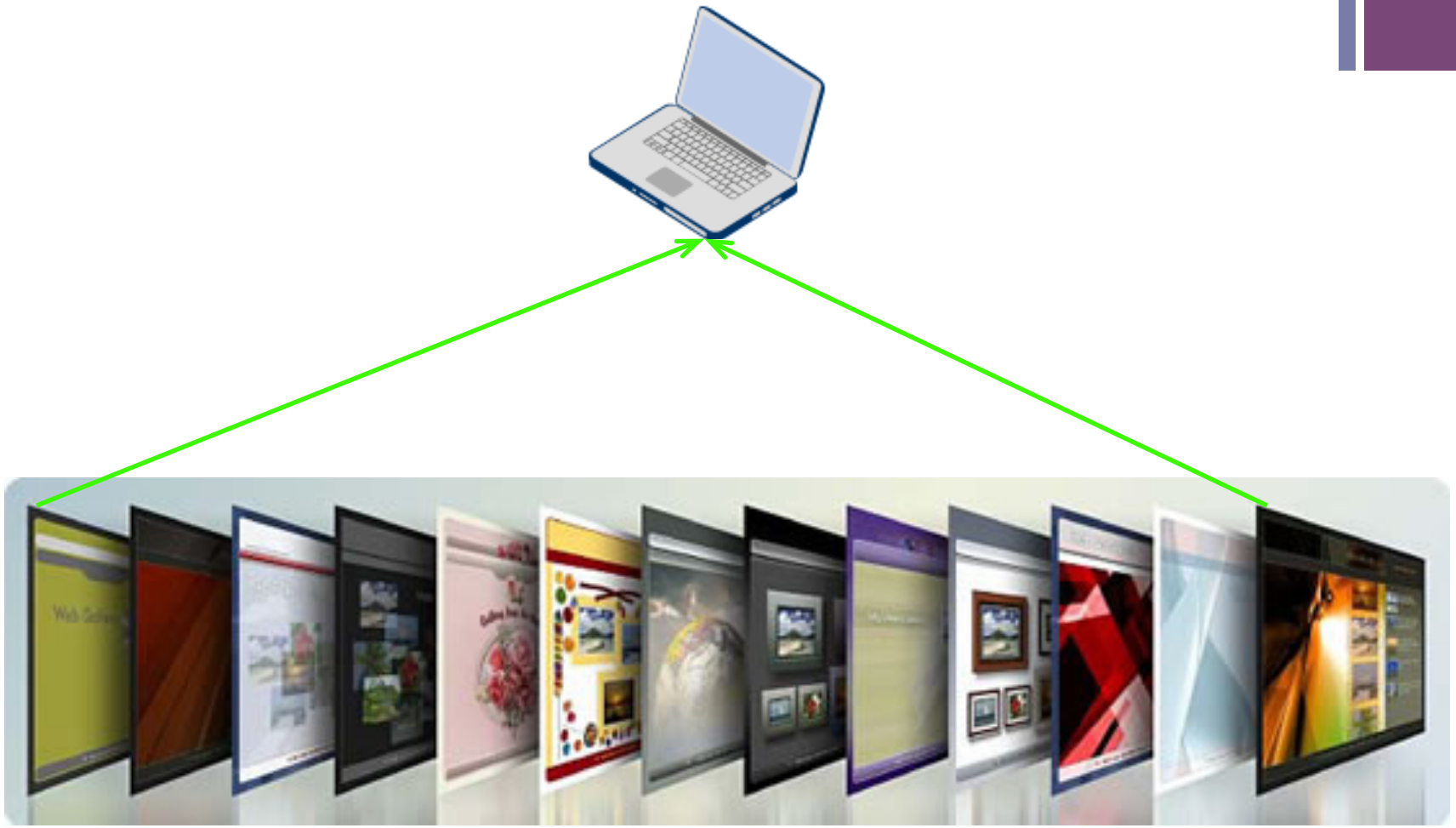
Open Web


Global Bank
https://banking.gb.com

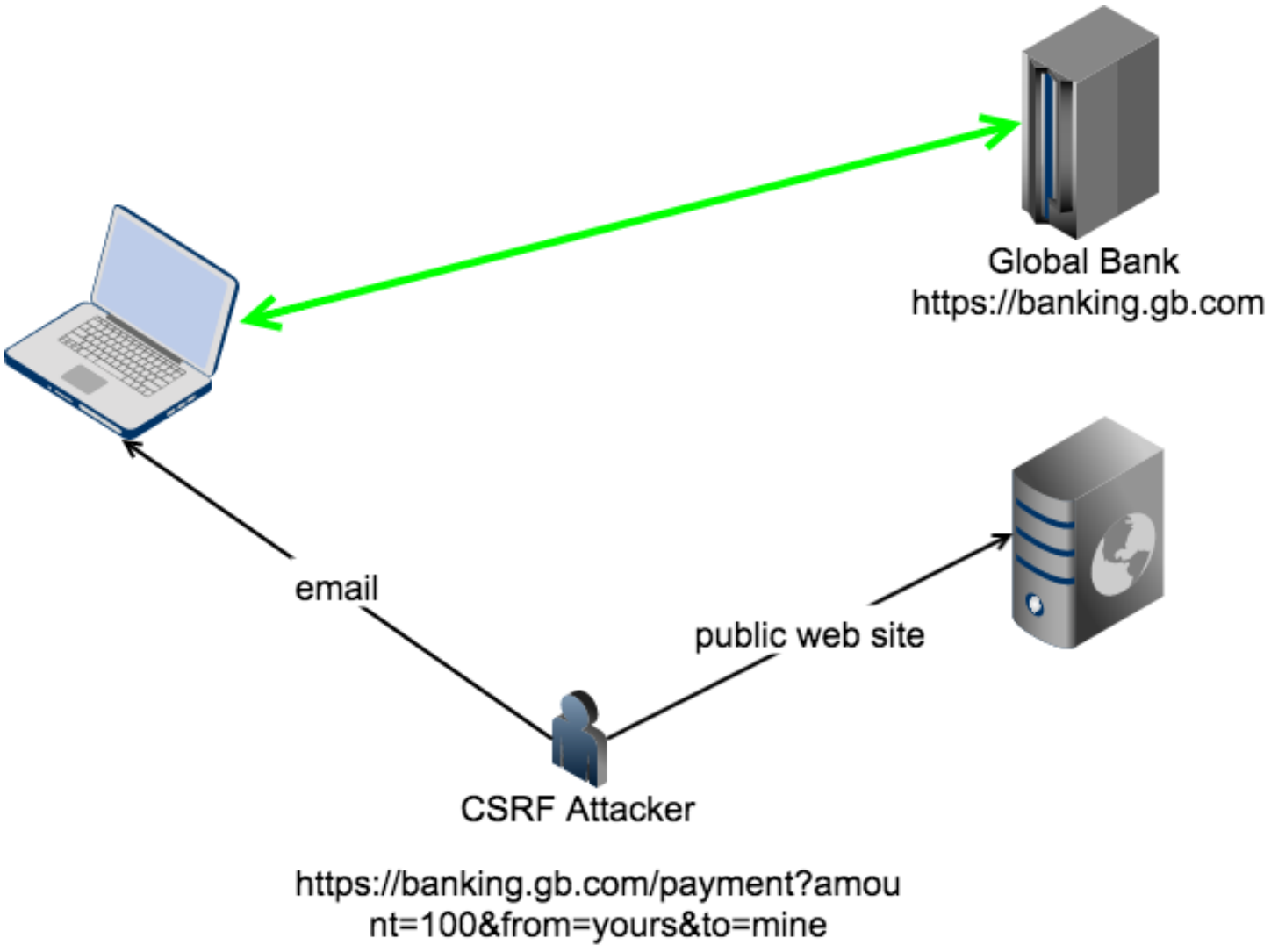
images.cool.com

Open Web

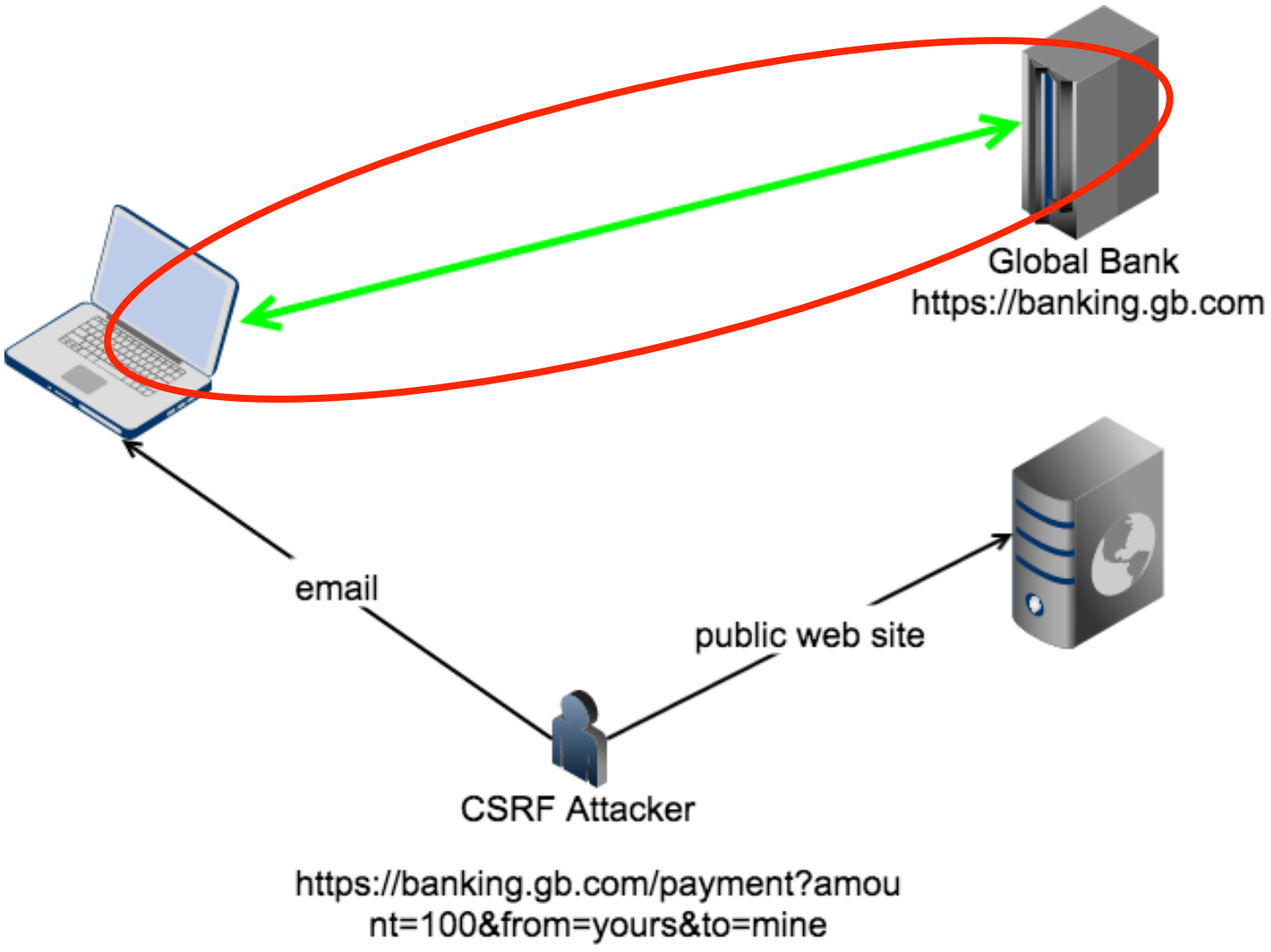
Amazing

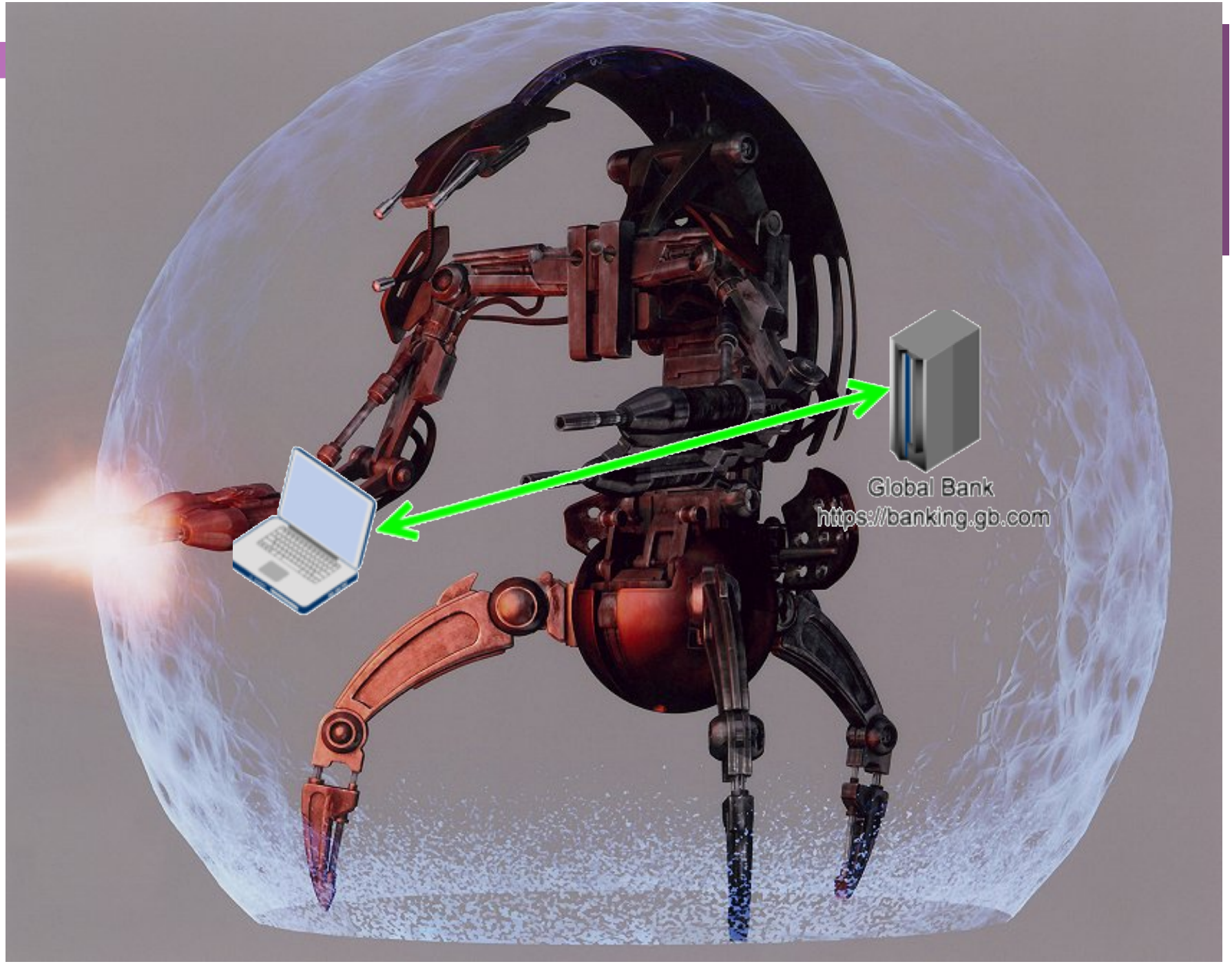


Open Web - **CSRF**

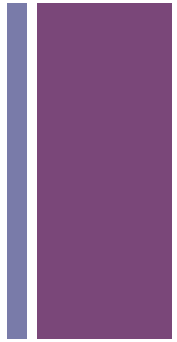


Open Web – **No CSRF**




Global Bank
https://banking.gb.com

Browser Features

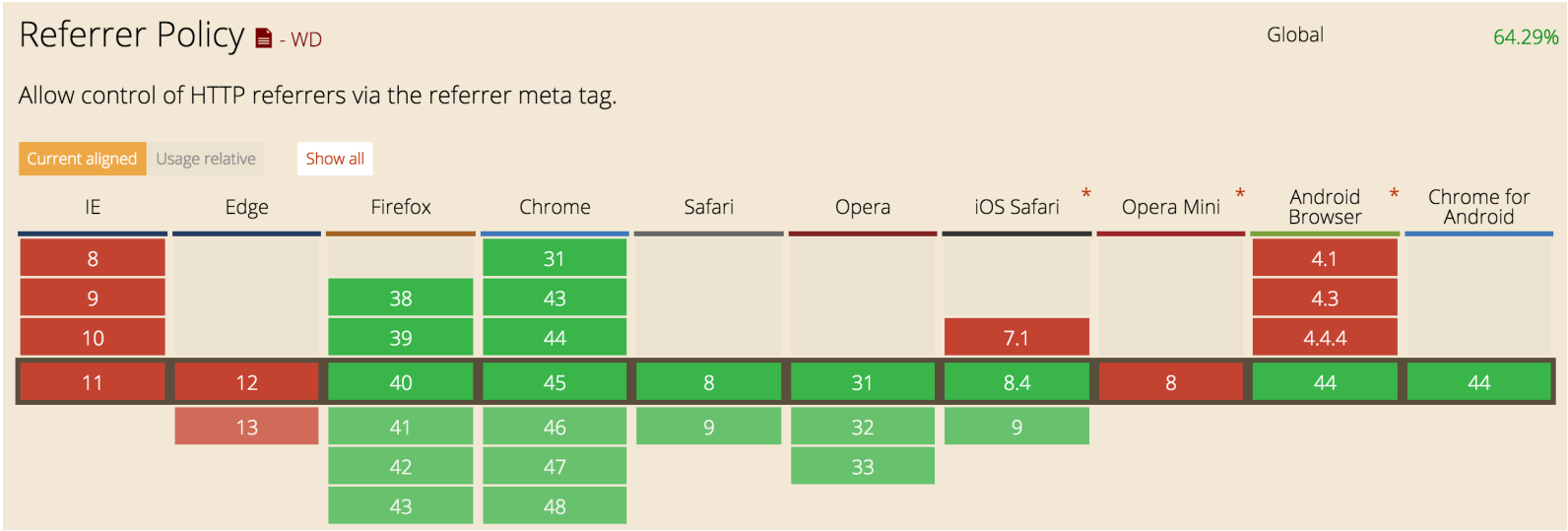


- HTTP Headers
  - Origin
  - Referer

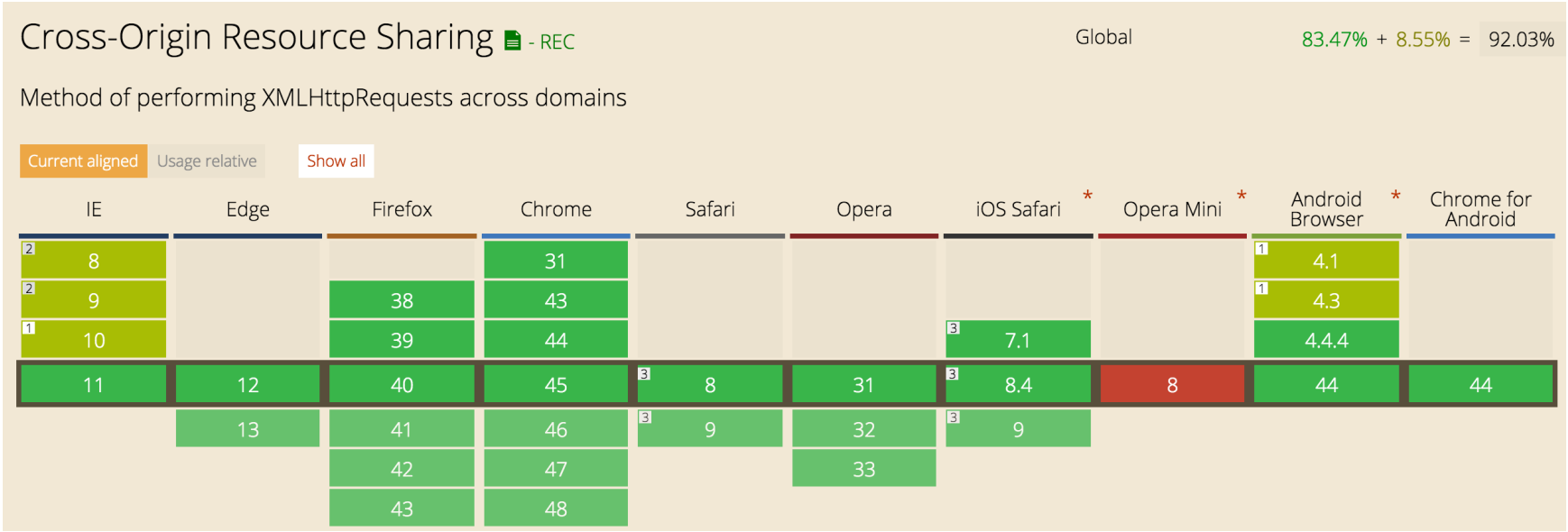
Browser Features

- HTTP Headers
  - Origin
  - Referer
  - Content Security Policy (CSP)

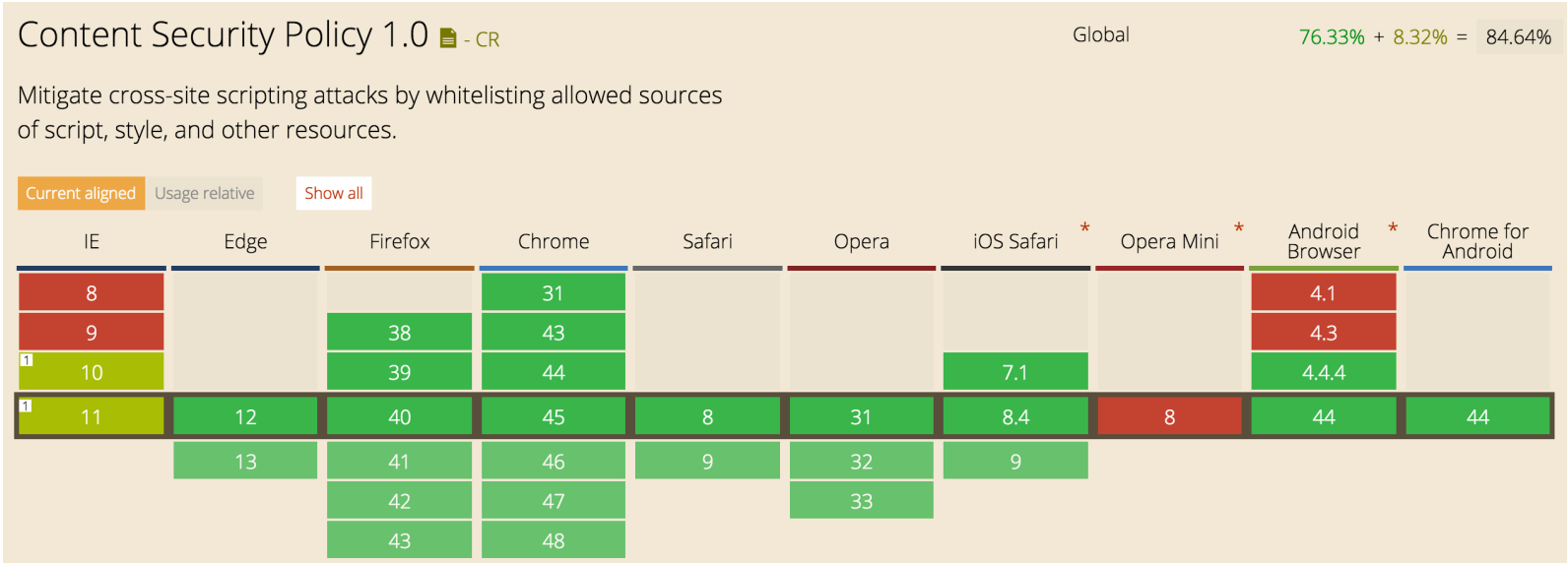
The Referer Header



The Origin Header



CSP


Content Security Policy 1.0 - CR
Global
76.33% + 8.32% = 84.64%
Mitigate cross-site scripting attacks by whitelisting allowed sources of script, style, and other resources.
Current aligned
Usage relative
Show all
IE
Edge
Firefox
Chrome
Safari
Opera
iOS Safari *
Opera Mini *
Android Browser *
Chrome for Android
8
9
10
11
12
13
31
38
39
40
41
42
43
43
44
45
46
47
48
8
9
31
32
33
7.1
8.4
9
8
4.1
4.3
4.4.4
44
44

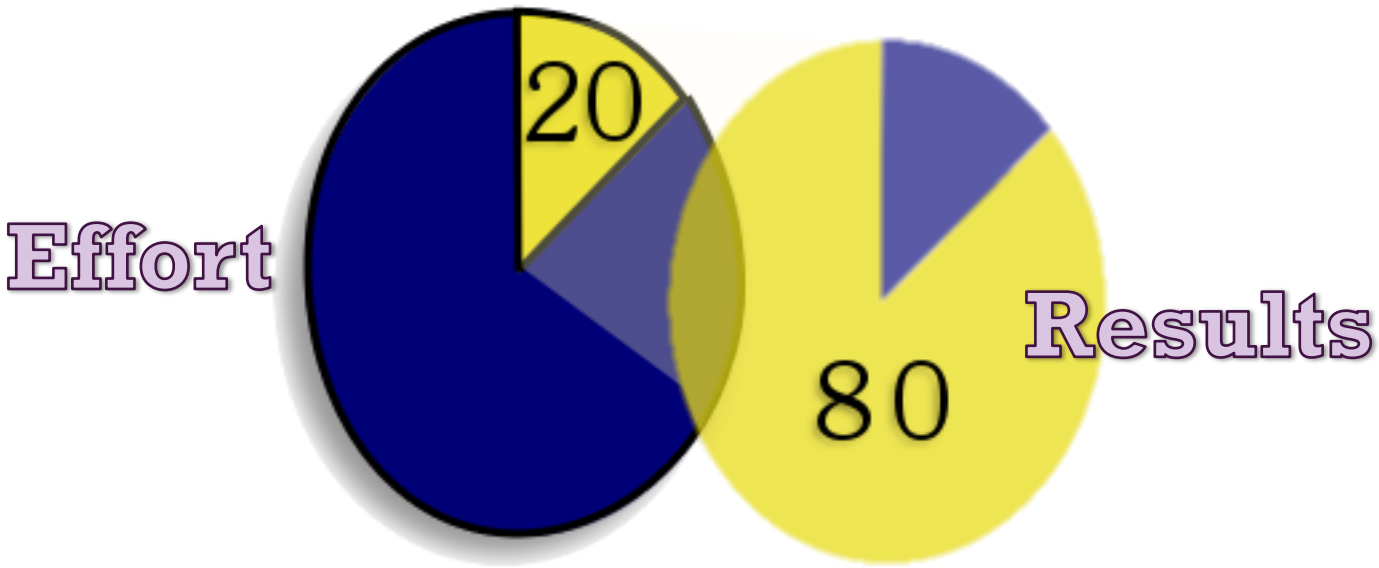
Using the Origin Header

- An API for a service handler to call

```
void verifyOriginHeader(HTTPRequest request);
```

- Implementation
  - Whitelist accepting domains
  - As a SaaS provider, allow customers to set the whitelist via configuration file or database query

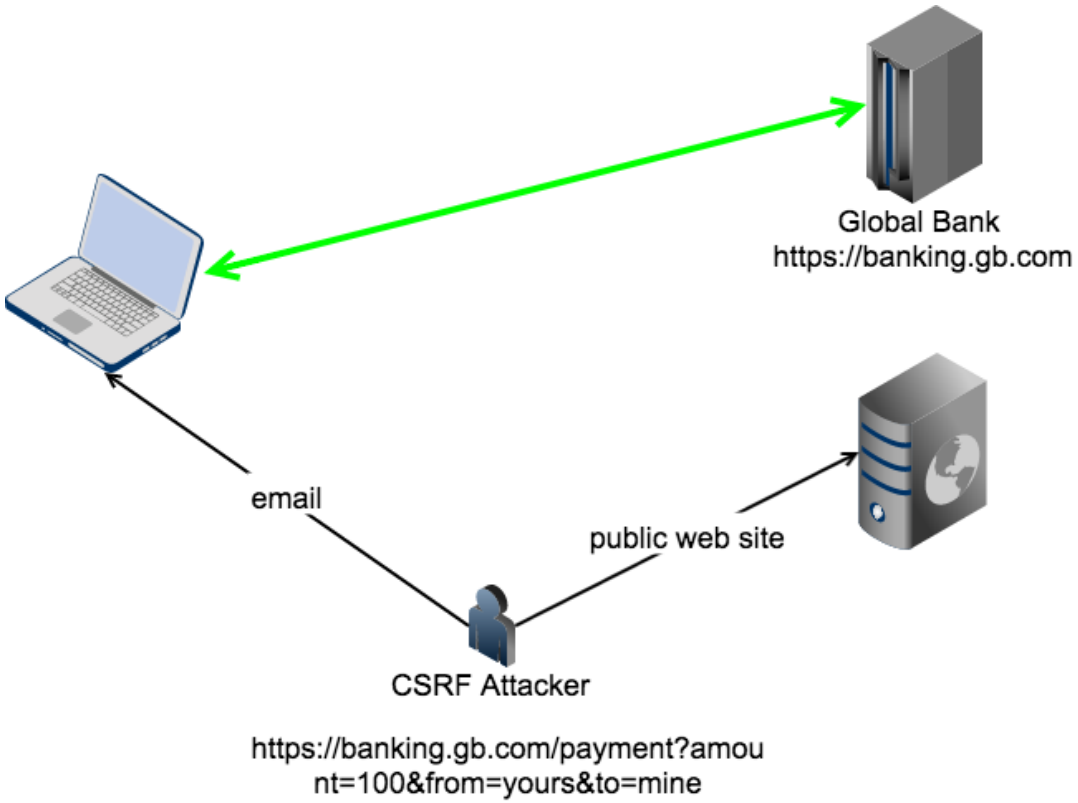
Pareto Empirical Principle: 80 - 20



Server Side – Central Question

Server needs to know if a request comes from itself

https://banking.gb.com/payment?amount=4&from=c&to=s

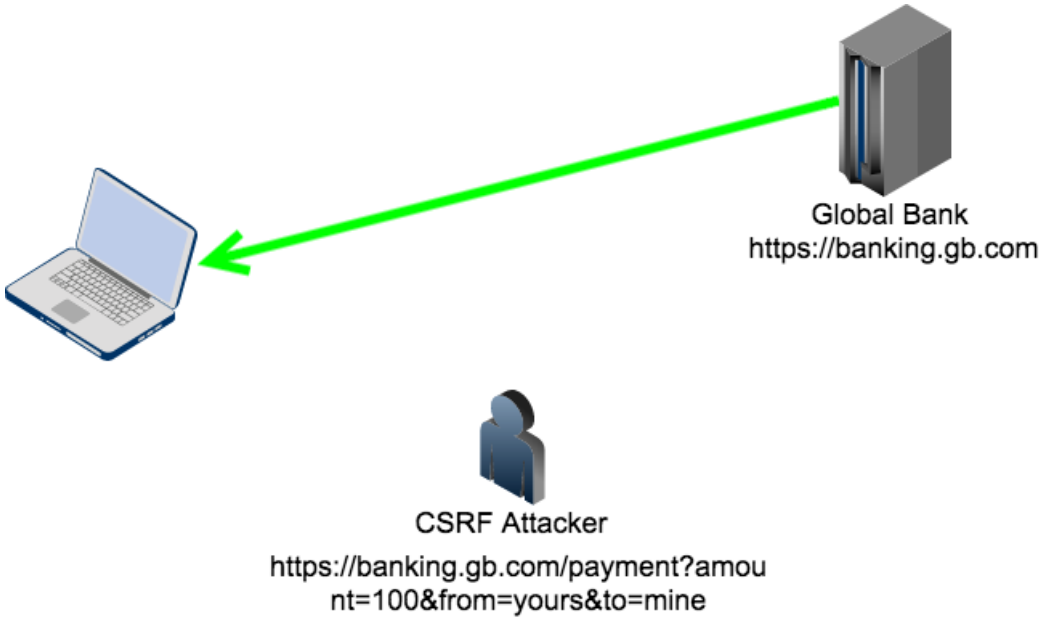


Solution: Cryptographic Secret

- Use a cryptographic secret to identify its own URLs

  CSRF Token/Nonce

https://banking.gb.com/payment?amount=9999&from=c&to=s&csrf=0123456789abcdef0123456789abcdef

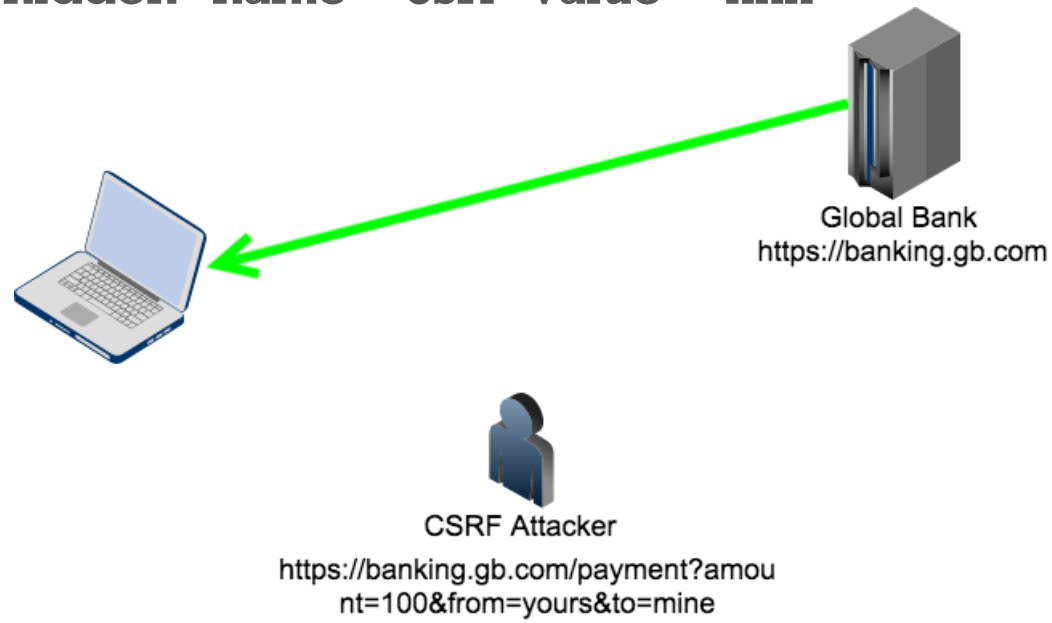


Solution: Cryptographic Secret

- Use a cryptographic secret to identify its own URLs
  CSRF Token/Nonce

<form method="POST" action="…&csrf=xxx>…</form>

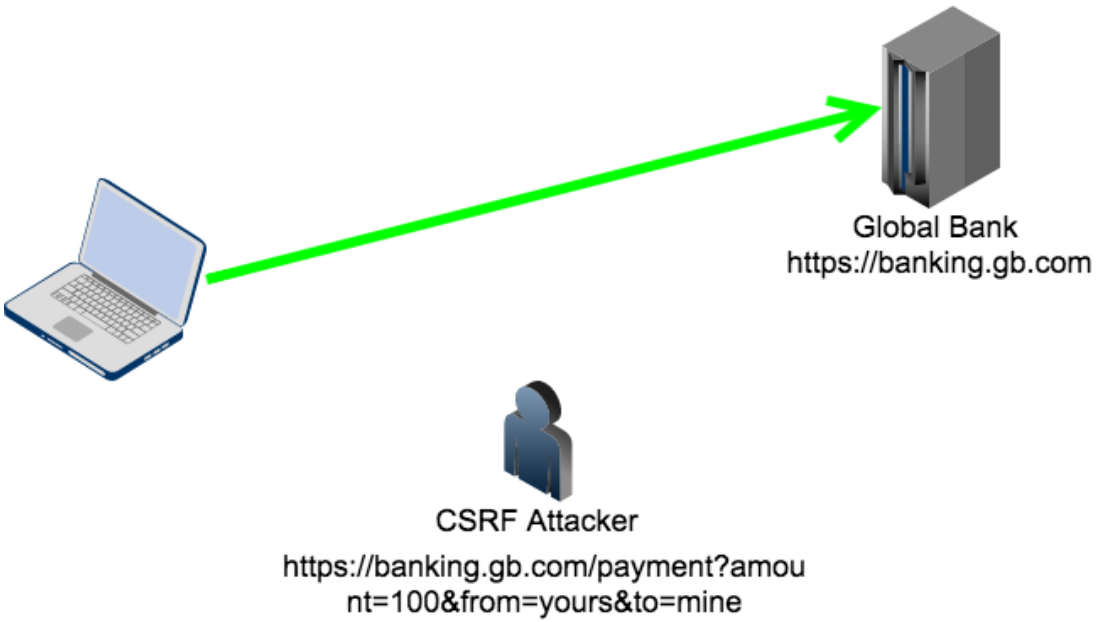
<input type="hidden" name="csrf" value="xxx">



Solution: Cryptographic Secret

- Request handler validates CSRF tokens

```
void verifyCSRFToken(HTTPRequest request);
```



CSRF Token Management

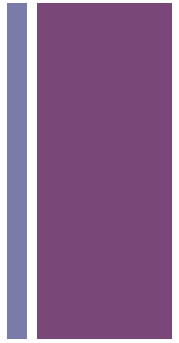
- Cryptographically strong
  - 256 bits, java.security.SecureRandom
- Rotation: creation and expiration
  - Create a new token for each new request
  - FIFO queue, Least recently used, …
- Internal APIs hidden from Web application
  - Only used by CSRF APIs

CSRF APIs

- Tag a request with a CSRF token
  - `String stampURL(String url);`
  - `FormGenerator stampURL(FormGenerator form);`
  - `String getToken();`

- Check the token
  - `void verifyCSRFToken(HTTPRequest request);`

POST vs. GET



- GET
  - cached, logged, etc.
  - more exposure than a form hidden field
  - RFC 2616 recommends as a *safe* method
  - related to token secret management
- POST

Server vs. Client Store

- Server stores CSRF tokens in user's session
- Client stores CSRF tokens in cookie

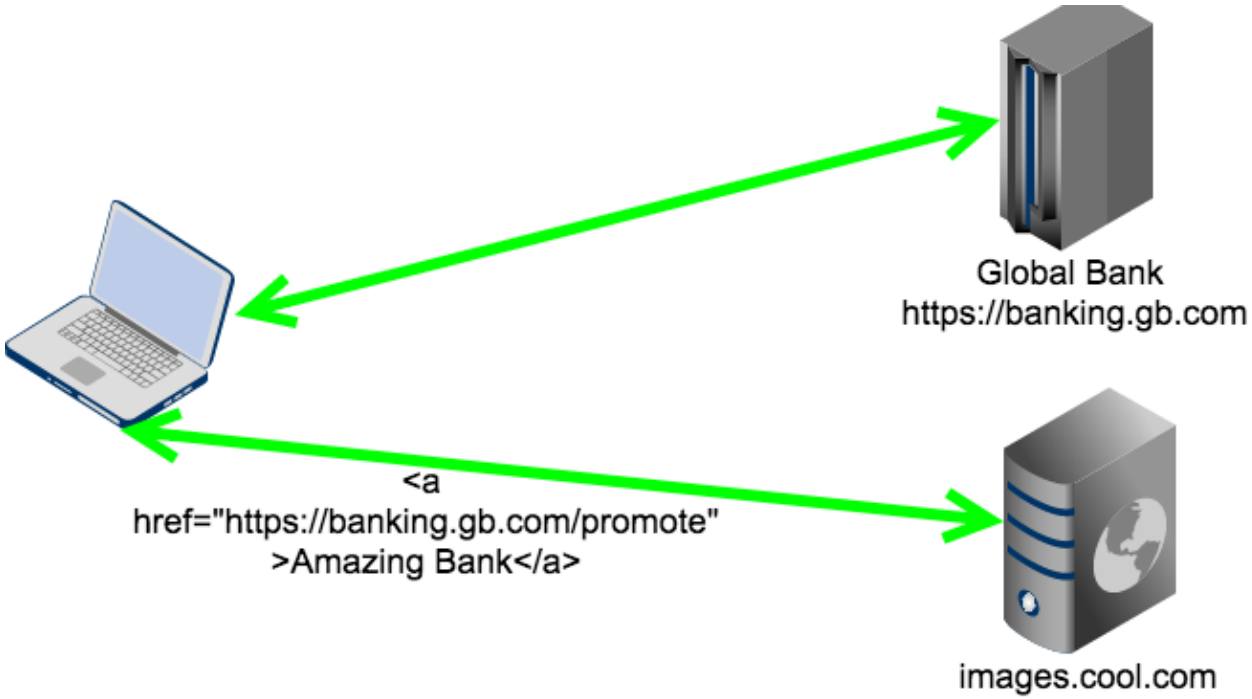
  aka. double submit cookies

Error Handling

- Return error page
- Return a page educating user about CSRF
- Internal logging

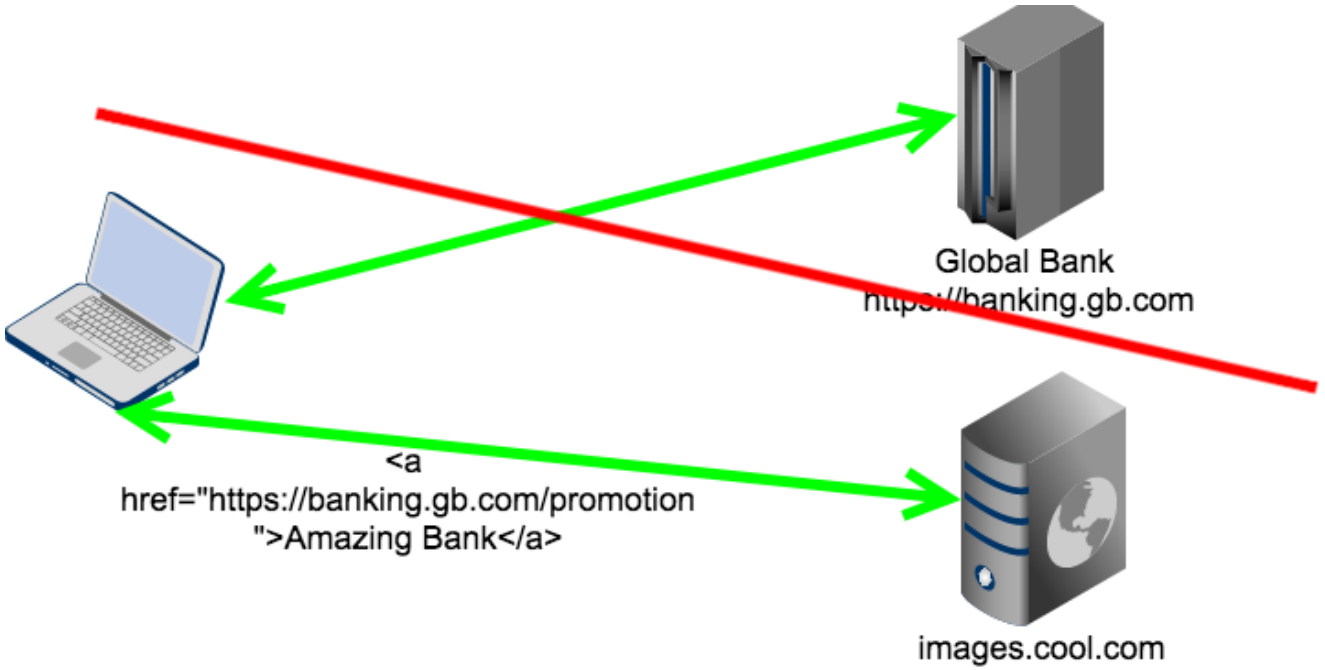
CSRF Boundary

- Not want to check requests for public landing pages



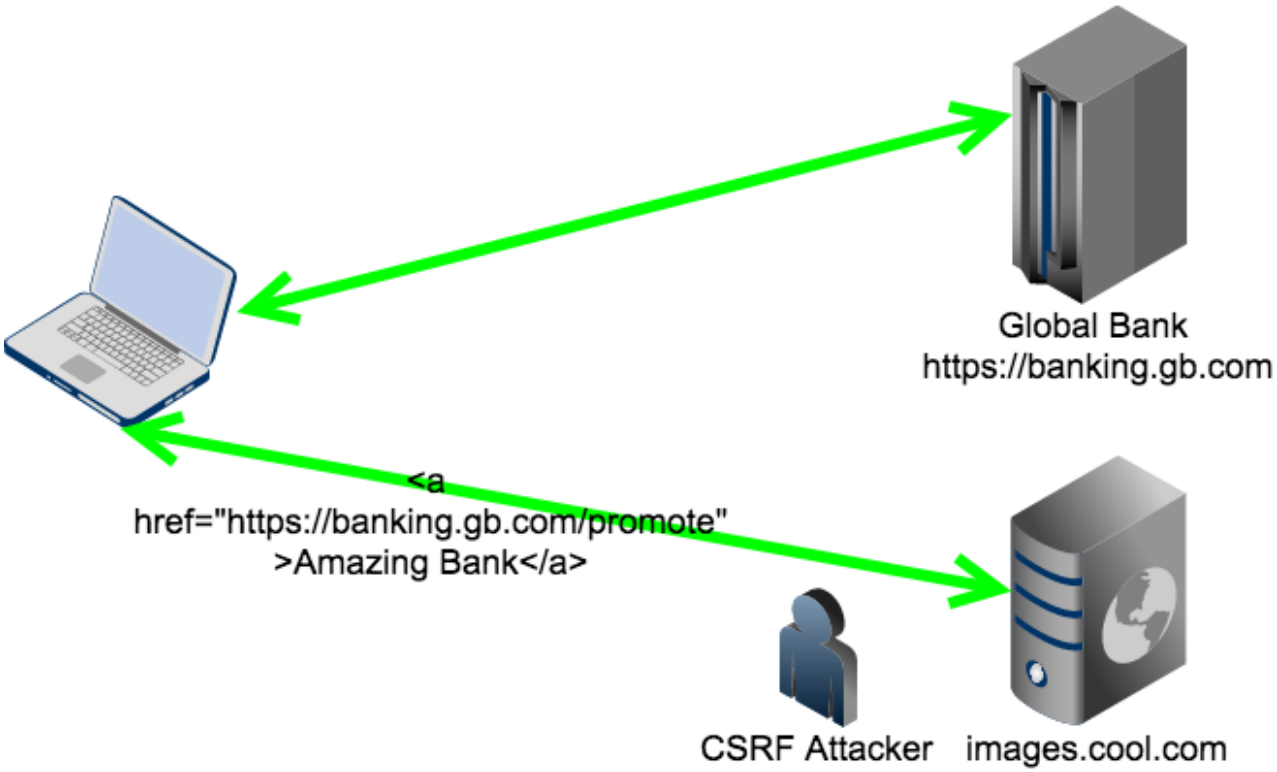
CSRF Boundary

- Cut yourself out of the Web



Draw a CSRF Boundary in URLs

The most difficult task in protecting a legacy web site



Draw a CSRF Boundary in URLs

- Lesson

Distinguish public resources and non-public resources before screwed up

Thank You!