

CSE467: Computer Security

11. Client-side Web Security (2)

Seongil Wi

HW2: Web Hacking Competition



- Hacking practice: Capture the Flag (CTF)
- Challenge open (competition start): 4/14 (Mon)
- Due date (writeup report): 5/2 (Fri)
- CTF server URL: You can obtain it by solving Problem 0 😊
 - This server can only be accessed from the UNIST internal network.
 - Please use a *VPN* to access from outside! Just log in to <https://vpn.unist.ac.kr> and turn on VPN
- ID: [Your Student ID]@unist.ac.kr
- PW: [Your Password]

HW2: Web Hacking Competition

- 11 Challenges

CSE467ScoreboardChallengesAdmin PanelNotificationsProfileSettings

Challenges

HW2

1. Login 10	2. Password 20	3. Password++ 30	4. Check Duplicate 60
5. Get Color 10	6. Uploader 20	7. Uploader++ 40	8. Search V1 15
9. Search V2 25	10. Service Center 30	11. Search V3 40	

HW2: Web Hacking Competition



- 11 Challenges
- Each flag is in the following format: `flag{[0-9a-f{32}]}`
 - e.g., `flag{1a79a4d60de6718e8e5b326e338ae533}`
- Do not attack the CTF environments, including web services!

HW2: Web Hacking Competition

5

The student who is the quickest to complete all the problems will earn bonus points!

Recap: How to Prevent XSS Attacks?



6

#1: Input validation/sanitization

- Any user input must be preprocessed before it is used inside HTML
- Option 1-1: Implement your own sanitization logic (not recommended)
- Option 1-2: Use the good escaping libraries
 - E.g., `htmlspecialchars(string)`, `htmlentities(string)`, ...

#2: Content Security Policy (CSP)

- A new security mechanism supported by modern browsers
- Next lecture!

Content-Security Policy (CSP)



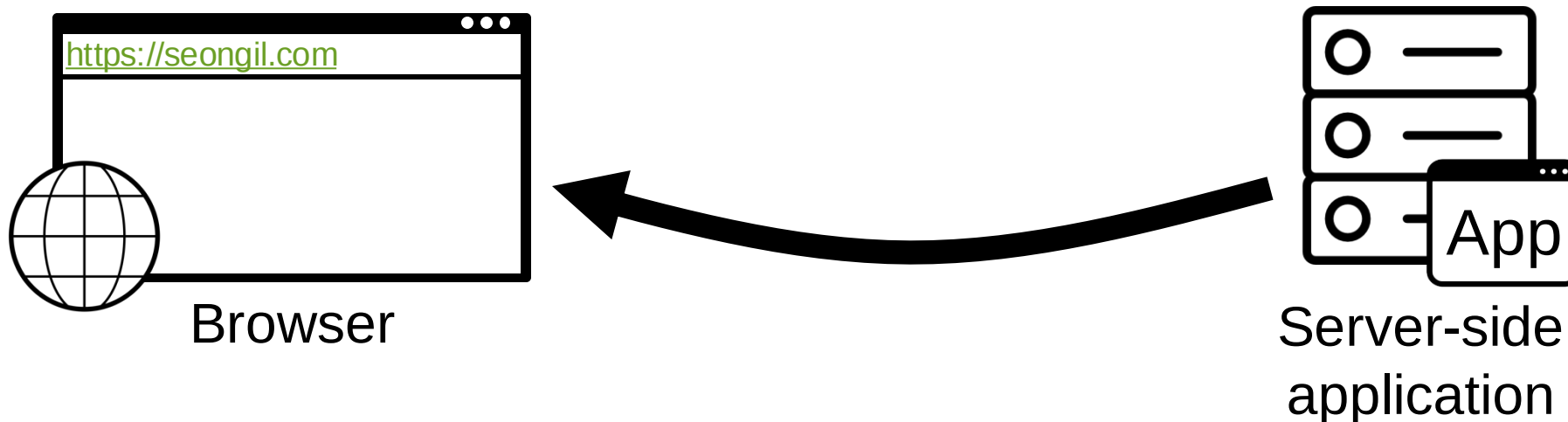
Content Security Policy (CSP)

- Explicitly allow resources which are trusted by the developer
 - Servers declare trusted sources

CSP Workflow

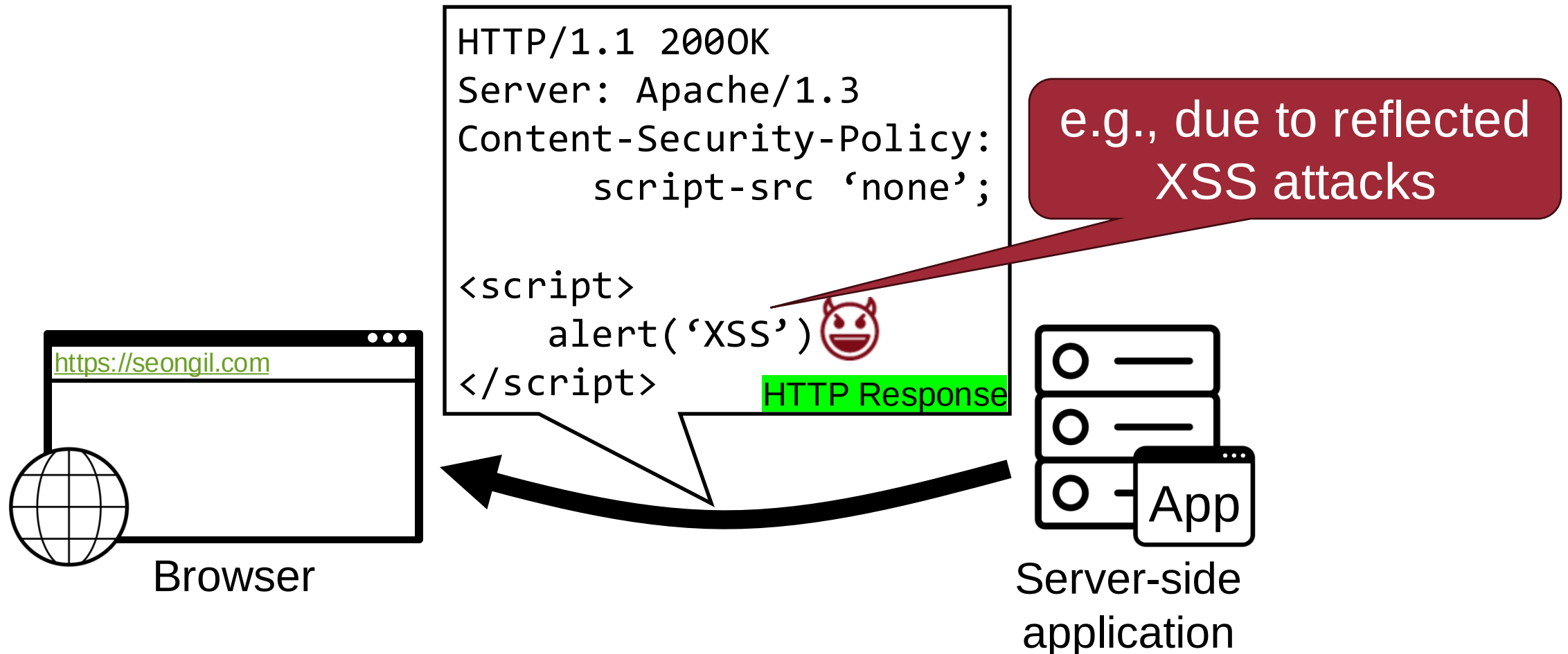


- Explicitly allow resources which are trusted by the developer
 - Servers declare trusted sources



CSP Workflow

- Explicitly allow resources which are trusted by the developer
 - Servers declare trusted sources



CSP Workflow

- Explicitly allow resources which are trusted by the developer
 - Servers declare trusted sources

HTTP/1.1 200OK

Server: Apache/1.3

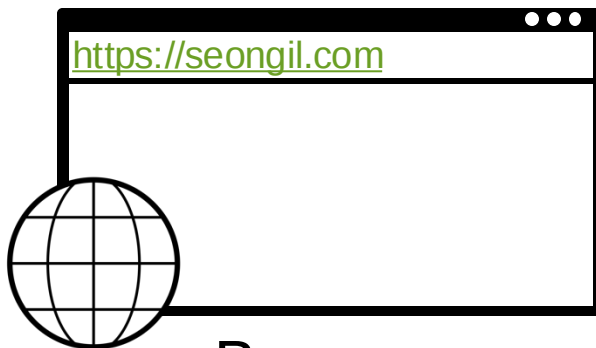
Content-Security-Policy:
script-src 'none';



```
<script>  
    alert('XSS')  
</script>
```



HTTP Response



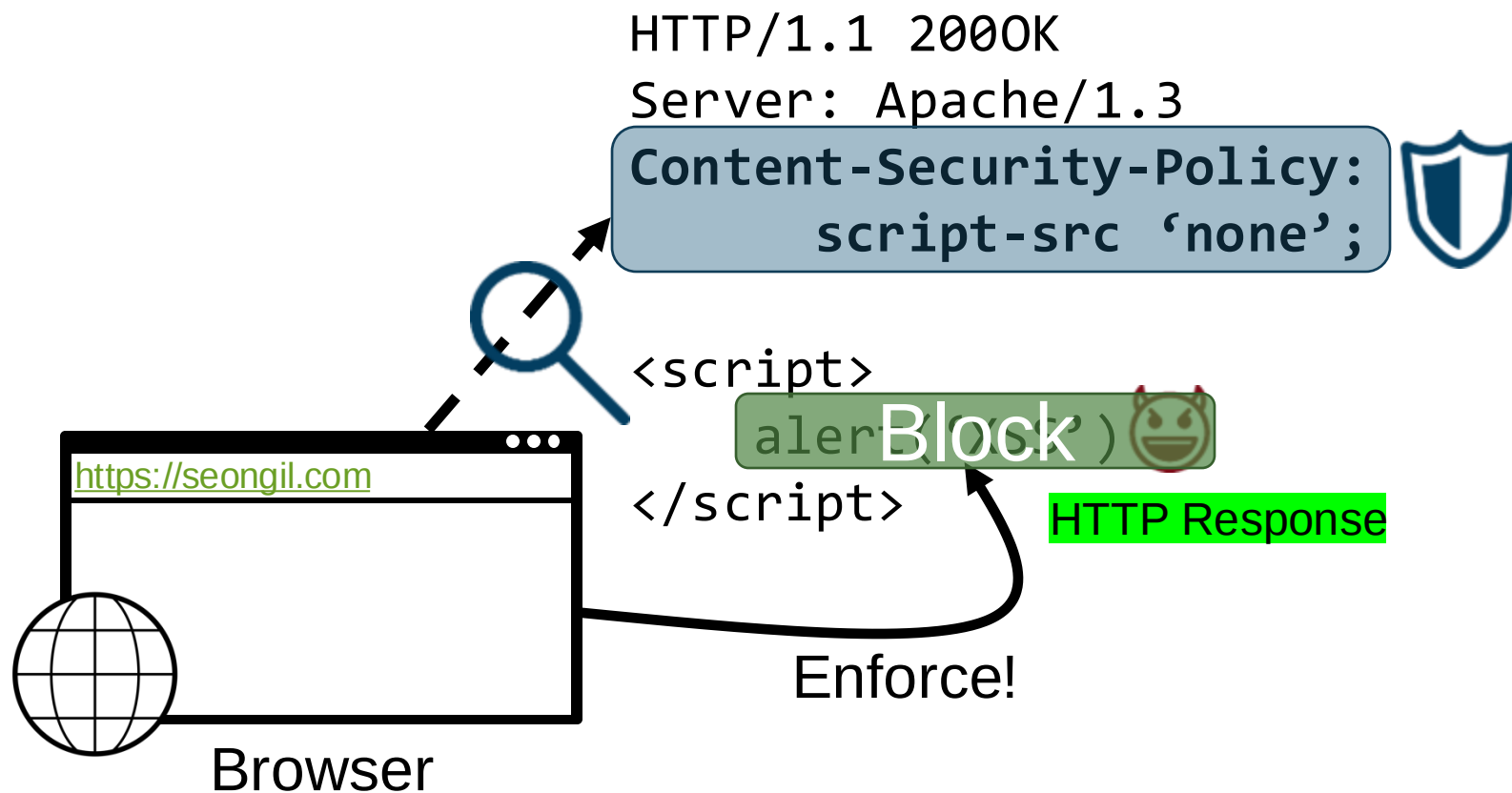
Browser

Servers declare
trusted sources

CSP Workflow



- Explicitly allow resources which are trusted by the developer
 - Servers declare trusted sources



Example Policy on paypal.com



Demo:

<https://www.paypal.com/home>

Content Security Policy (CSP)

14

- Explicitly allow resources which are trusted by the developer
 - Servers declare trusted sources
- Disallow dangerous JS constructs like eval or event handlers
- Delivered as HTTP header or in meta element in page
 - **HTTP header:** Content-Security-Policy: **default-src ...**
 - **Meta element:** `<meta http-equiv="Content-Security-Policy" content="default-src...">`
- **Enforced by the browser (all policies must be satisfied)**
 - Your browser must support CSP for its use
- First candidate recommendation in 2012, currently at Level 3

Browser Support

Chrome

Content-Security-Policy CSP Level 3 - Chrome 59+ Partial Support
 Content-Security-Policy CSP Level 2 - Chrome 40+ Full Support Since January 2015
 Content-Security-Policy CSP 1.0 - Chrome 25+
 X-Webkit-CSP **Deprecated** - Chrome 14-24

Safari

Content-Security-Policy CSP Level 3 - Safari 15.4+ Partial Support
 Content-Security-Policy CSP Level 2 - Safari 10+
 Content-Security-Policy CSP 1.0 - Safari 7+
 X-Webkit-CSP **Deprecated** - Safari 6

Firefox

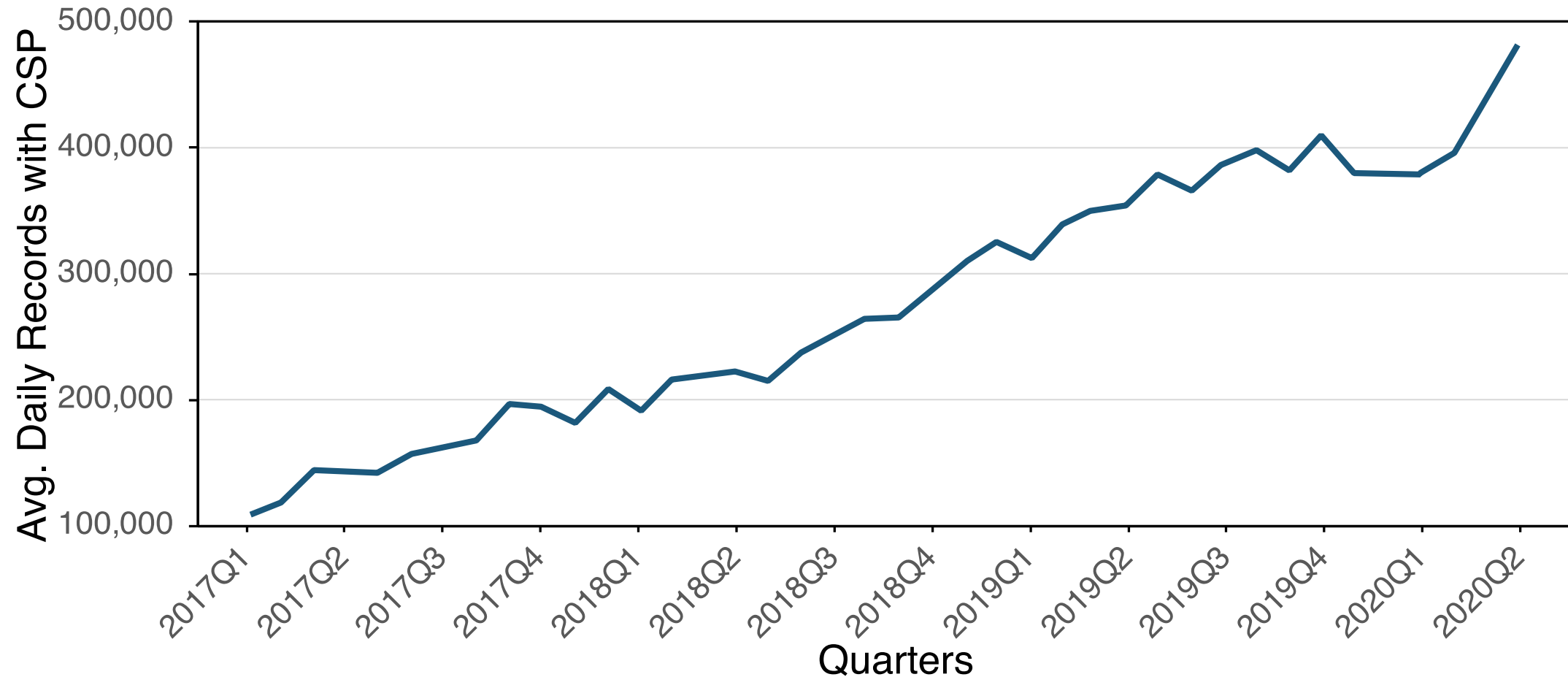
Content-Security-Policy CSP Level 3 - Firefox 58+ Partial Support
 Content-Security-Policy CSP Level 2 - Firefox 31+ *Partial Support* since July 2014
 Content-Security-Policy CSP 1.0 - Firefox 23+ Full Support
 X-Content-Security-Policy **Deprecated** - Firefox 4-22

Edge

Content-Security-Policy CSP Level 3 - Edge 79+ Partial Support
 Content-Security-Policy CSP Level 2 - Edge 15+ Partial, 76+ Full
 Content-Security-Policy CSP 1.0 - Edge 12+

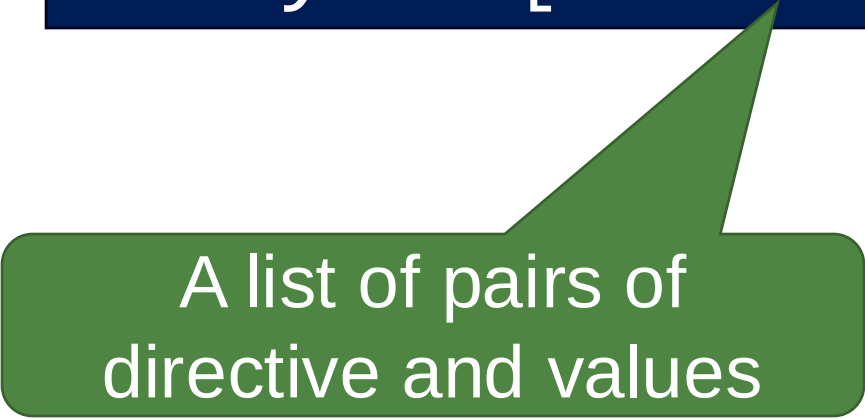
CSP Popularity

16



Format of CSP

```
Policy := [directive [value1];]...
```



A list of pairs of
directive and values

CSP Level 1 – Controlling Scripting Resources 18

Policy := [directive [value1];]...

- ✓ **Directive:** `script-src`
 - Specifically controls where scripts can be loaded from
 - **If provided, inline scripts and eval will not be allowed**
- ✓ **Value:** Many different ways to control sources
 - **'none'** – no scripts can be included from any host
 - **'self'** – only own origin
 - **https://domain.com** – allow the script from this origin
 - **https://*.domain.com** – any subdomain of domain.com, any script on them
 - **https:** – any origin delivered via HTTPs
 - **'unsafe-inline' / 'unsafe-eval'** – reenables inline handlers and eval

CSP Level 1 Demo

19



Demo:

https://websec-lab.github.io/courses/2025s-cse467/demo/csp_demo.html

CSP Level 1 – Example

CSP for website <https://example.com>:

```
script-src 'self' https://unist.ac.kr;
```

Executes scripts only
from the same origin and
<https://unist.ac.kr>

HTML	Execution Allowed?
<code><script src = “https://unist.ac.kr/myscript.js”></script></code>	✓
<code><script src = “https://example.com/stuff.js”></script></code>	✓
<code><script>alert(1)</script> // inline script</code>	✗
<code><script src = “https://ad.com/someads.js”></script></code>	✗

CSP Level 1 – Controlling Additional Resources

21

- **img-src, style-src, font-src, object-src, media-src**
 - Controls non-scripting resources: images, CSS, fonts, objects, audio/video
- **frame-src**
 - Controls from which origins frames may be added to a page
- **connect-src**
 - Controls XMLHttpRequest, WebSockets (and other) connection targets
- **default-src**
 - Serves as fallback for all fetch directives (all of the above)
 - **Only used when specific directive is absent**

CSP Level 1 – Limitations

- If our goal is to allow scripts from own origin and inline scripts
 - Solution: `script-src 'self' 'unsafe-inline'`
- **Problem:** bypasses literally any protection
 - Attacker can inject inline JavaScript
- **One possible solution:** removing inline script and converting it into an external script

For each inline script...



- Proposed improvement in CSP Level 2: **nonces and hashes**

CSP Level 2 – Nonces and Hashes



- Allows every inline script adds nonce property
 - script-src 'nonce-\$value' 'self'
- Allows inline scripts based on their SHA hash (SHA256, SHA384, or SHA512)
 - script-src 'sha256-\$hash' 'self'

CSP Level 2 – Example

26

```
script-src 'self' https://cdn.example.org  
'nonce-d90e0153c074f6c3fcf53' 'sha256-  
5bf5c8f91b8c6adde74da363ac497d5ac19e4595fe39cbdda22cec8445d3814c'
```

```
<script>  
alert("My hash is correct")  
</script>
```

```
<script>  
alert("incorrect")  
</script>
```

SHA256 hash value:
5bf5c8f91b8c6adde74da363ac497d5ac19
e4595fe39cbdda22cec8445d3814c

CSP Level 2 – Example

27

```
script-src 'self' https://cdn.example.org  
'nonce-d90e0153c074f6c3fcf53' 'sha256-  
5bf5c8f91b8c6adde74da363ac497d5ac19e4595fe39cbdda22cec8445d3814c'
```



```
<script>  
alert("My hash is correct")  
</script>
```

SHA256 matches
value of CSP header



```
<script>  
alert("incorrect")  
</script>
```

SHA256 does not
match

CSP Level 2 – Example

28

```
script-src 'self' https://cdn.example.org  
'nonce-d90e0153c074f6c3fcf53' 'sha256-  
5bf5c8f91b8c6adde74da363ac497d5ac19e4595fe39cbdda22cec8445d3814c'
```

```
<script nonce="d90e0153c074f6c3fcf53">  
alert("It's all good")  
</script>
```



Script nonce matches
CSP header

```
<script nonce="nocluehackplz">  
alert("I will not work")  
</script>
```



Script nonce does not
match CSP header

CSP Level 2 – Limitations

```
script-src 'self' https://cdn.example.org  
'nonce-d90e0153c074f6c3fcf53' 'sha256-  
5bf5c8f91b8c6adde74da363ac497d5ac19e4595fe39cbdda22cec8445d3814c'
```

```
<script nonce="d90e0153c074f6c3fcf53">  
  script=document.createElement("script");  
  script.src = "http://ad.com/ad.js";  
  document.body.appendChild(script);  
</script>
```

Add new script element
without nonce

Does this script work under a
nonce-based policy?

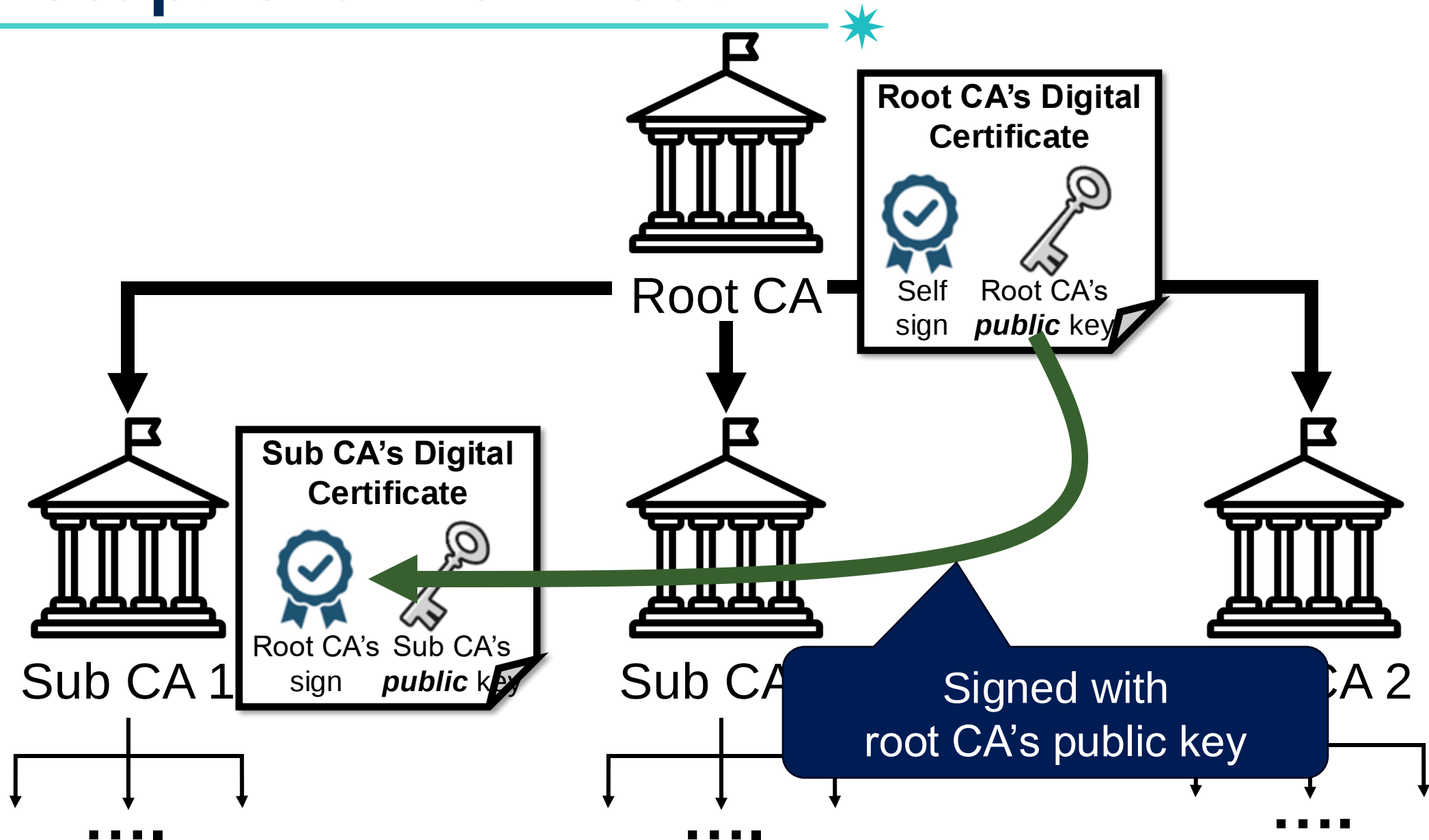
No!

Changes from Level 2 to Level 3: Strict-dynamic

- Additional changes: add **strict-dynamic**
 - Allows adding scripts programmatically, eases CSP deployment in, e.g., ad scenario
- Mostly due to dynamic ADs
 - 1st page load: script from ads.com → fancy-cars.com
 - 2nd page load: script from ads.com → cheap-ads.net → dealsdeals.biz
- **Idea: propagate trust**
 - If we trust ads.com, let's also trust whoever ads.com load script from

Recap: Chain of Trust

31



CSP Level 3 – strict-dynamic Example

32

```
script-src 'self' https://cdn.example.org  
'nonce-d90e0153c074f6c3fcf53' 'strict-dynamic'
```

```
<script nonce="d90e0153c074f6c3fcf53">  
  script=document.createElement("script");  
  script.src = "http://ad.com/ad.js";  
  document.body.appendChild(script);  
</script>
```

We trust this script

Propagate trust: we also trust this script, so we allow it to execute

CSP – Bypasses

33



- Does not stop XSS, tries to mitigate its effects
- Problem #1: User input at the trusted script

Problem #1: User Input

```
script-src 'nonce-random123' 'strict-dynamic';
```

- What if the injection happens directly at nonced script elements?

```
<script nonce="random123">  
  script=document.createElement("script");  
  script.src = user_input + "valid.js";  
  document.body.appendChild(script);  
</script>
```

Problem #1: User Input

```
script-src 'nonce-random123' 'strict-dynamic';
```

- What if the injection happens directly at nonced script elements?

```
<script nonce="random123">  
  script=document.createElement("script");  
  script.src = user_input + "valid.js";  
  document.body.appendChild(script);  
</script>
```

Executes on the
target origin

Attacker can inject
attacker.com

CSP – Bypasses



- Does not stop XSS, tries to mitigate its effects
- Problem #1: User input at the trusted script
- Problem #2: Developer's mistake/misconfiguration

Problem #2: Developer's Mistake/Misconfiguration 27

- Developer's *mistake*

```
default-src 'self'
```

- Typo in the first directive leads to the default-src directive being missing from the policy (Content Security Problems?, **CCS'16**)

- Developer's *misconfiguration*

```
default-src 'unsafe-inline' *
```

- Defining CSP is hard!
- Many website developers just allow all of the inline script and all hosts
- **94.72% of all website bypassible (e.g., misconfigured their CSP)**
(CSP Is Dead, Long Live CSP!, **CCS'2016**)

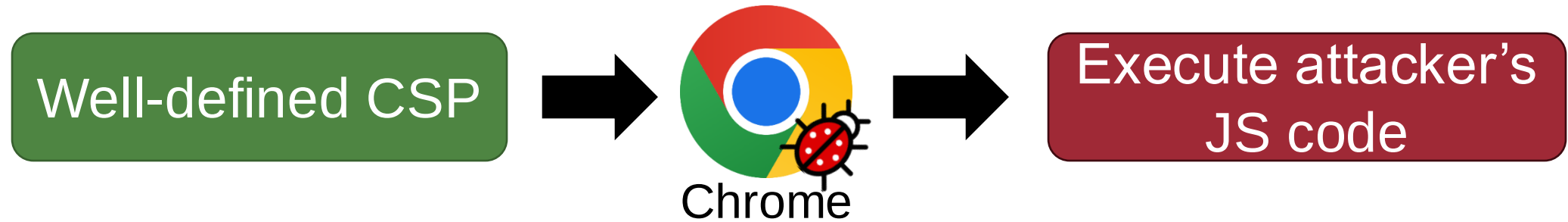
CSP – Bypasses



- Does not stop XSS, tries to mitigate its effects
- Problem #1: User input at the trusted script
- Problem #2: Developer's mistake/misconfiguration
- Problem #3: Browser bugs (CSP enforcement bugs)

Problem #3: Browser Bugs

- NOTE: CSP is enforced by the browser



```
<?php
  header("HTTP/: 100");
  header("Content-Security-Policy: default-src 'self'")
?>
<script>alert(1)</script>
```

Expected behavior:
Not execute JS code



Execute
JS code
(due to 100
status code)



Safari



Not execute
JS code



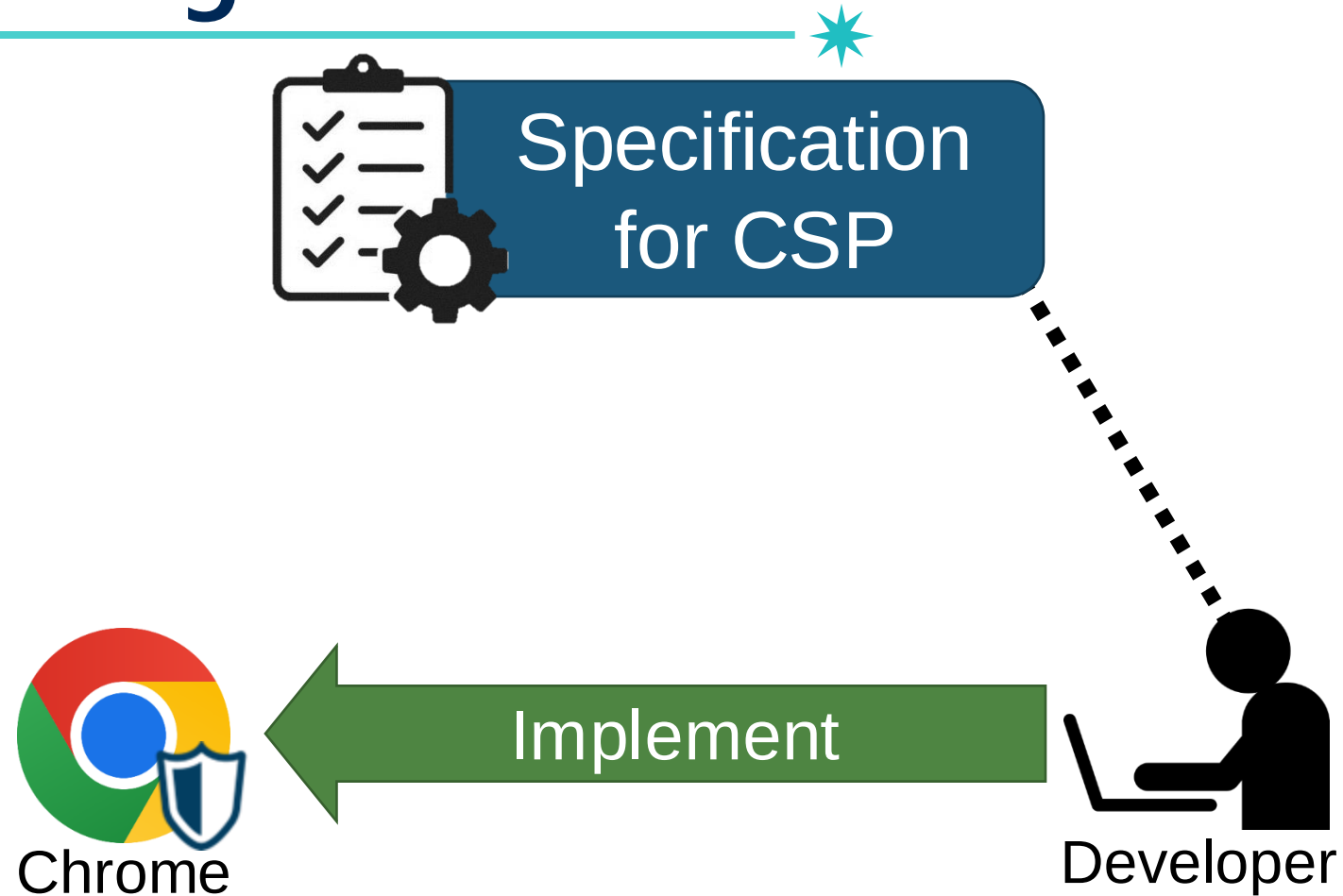
Firefox



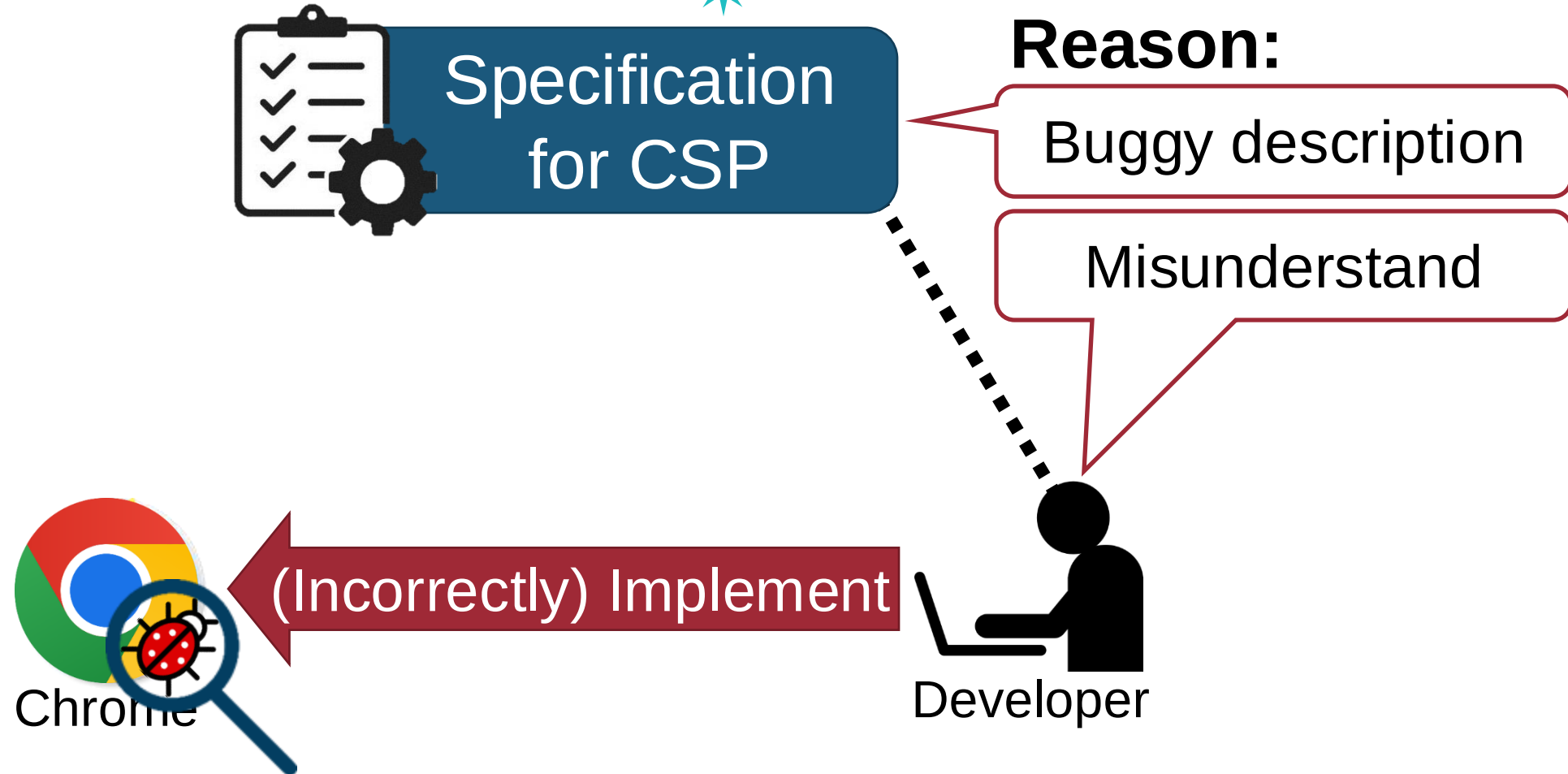
Not execute
JS code

**Research Question:
How to find
CSP-related browser bugs?**

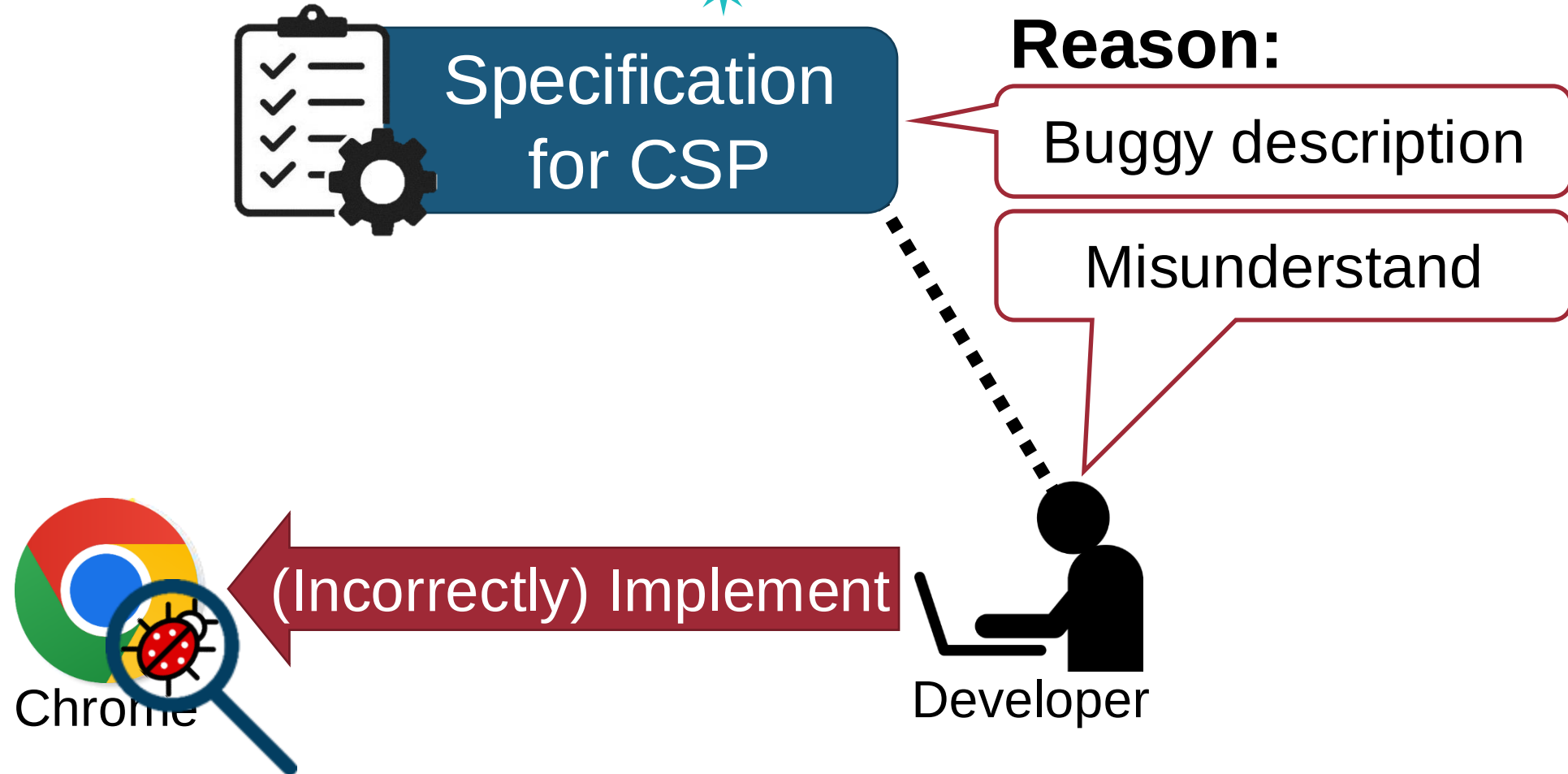
(FYI) Finding CSP Enforcement



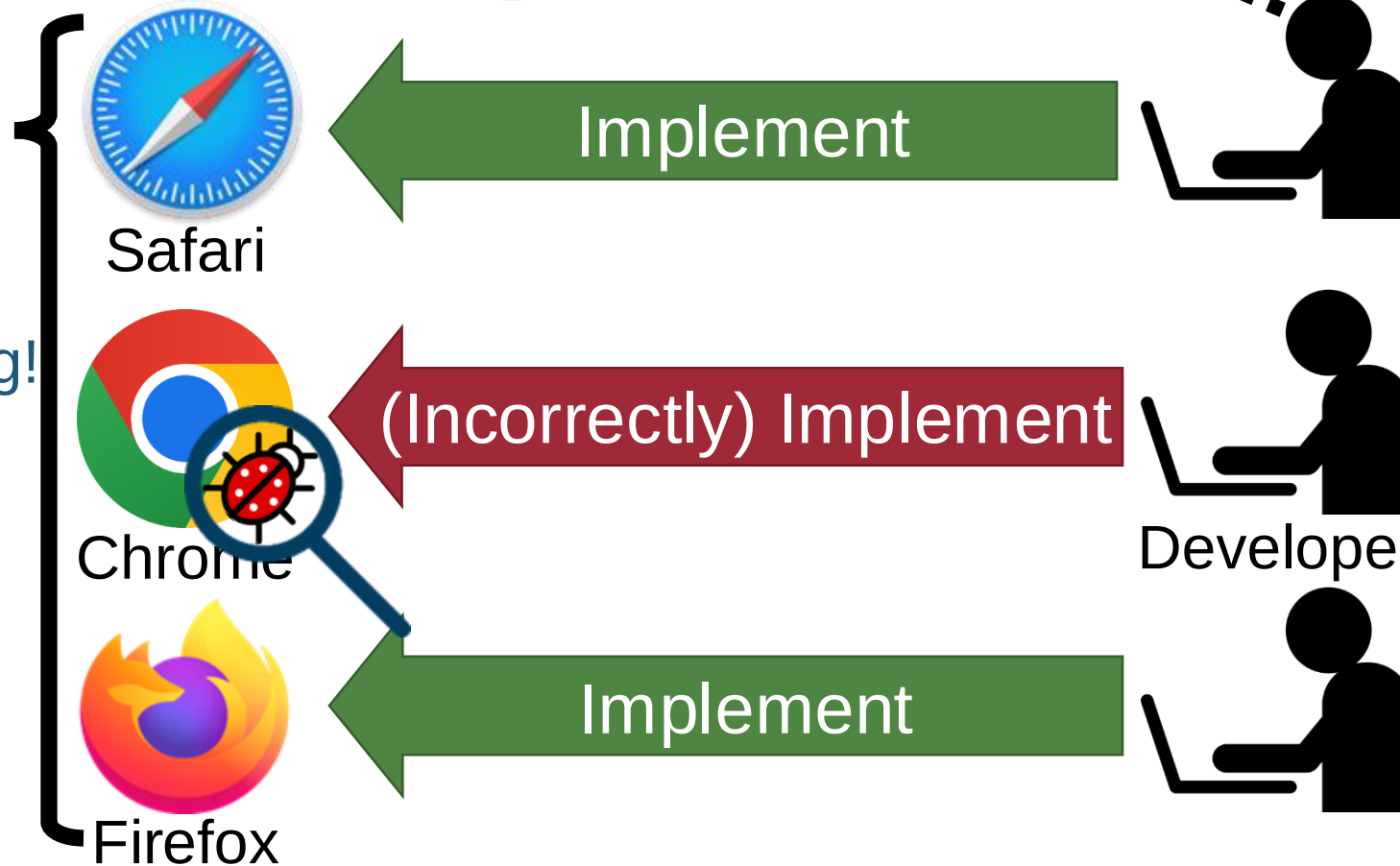
(FYI) Finding CSP Enforcement



(FYI) Finding CSP Enforcement

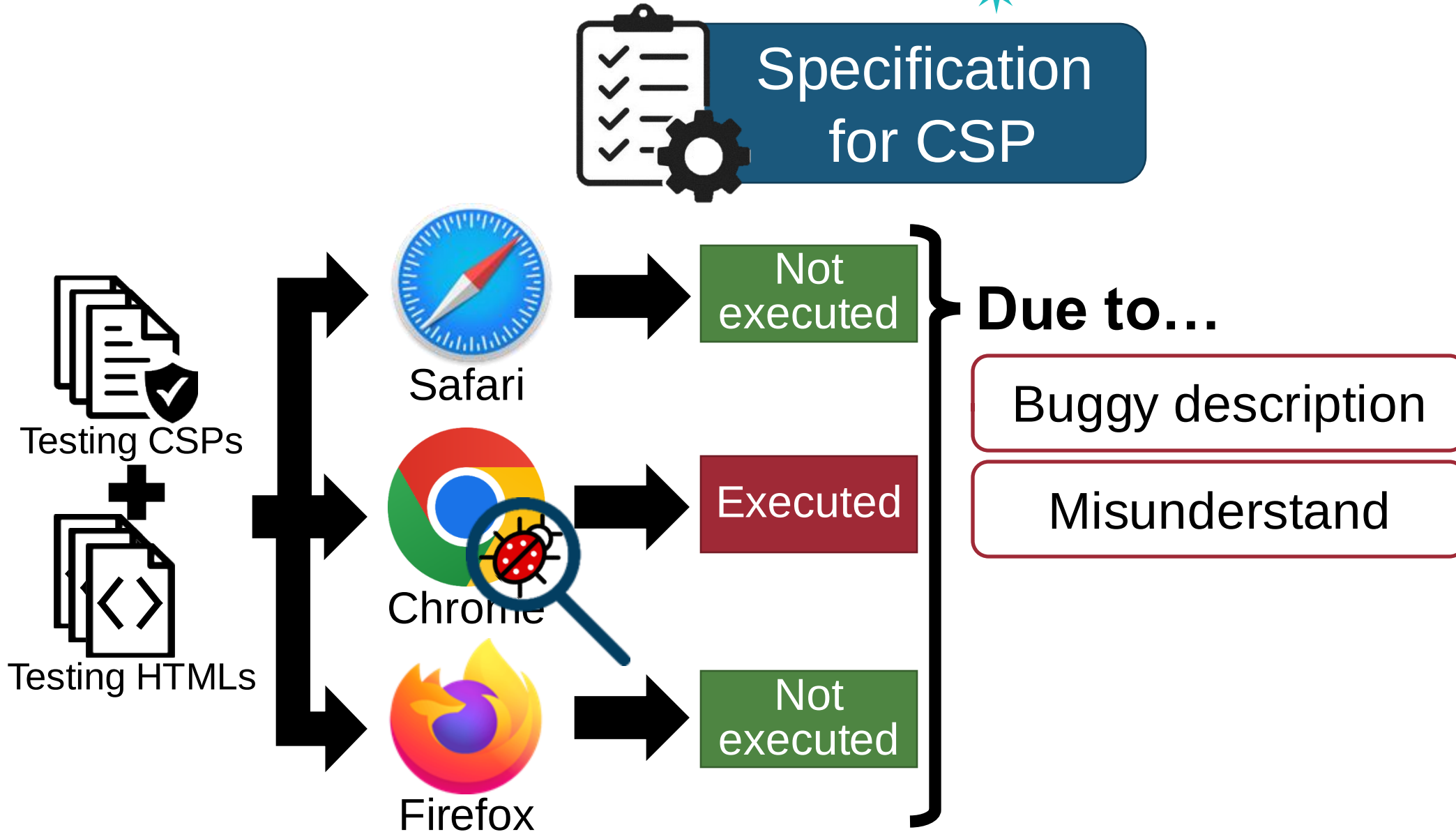


(FYI) Finding CSP Enforcement

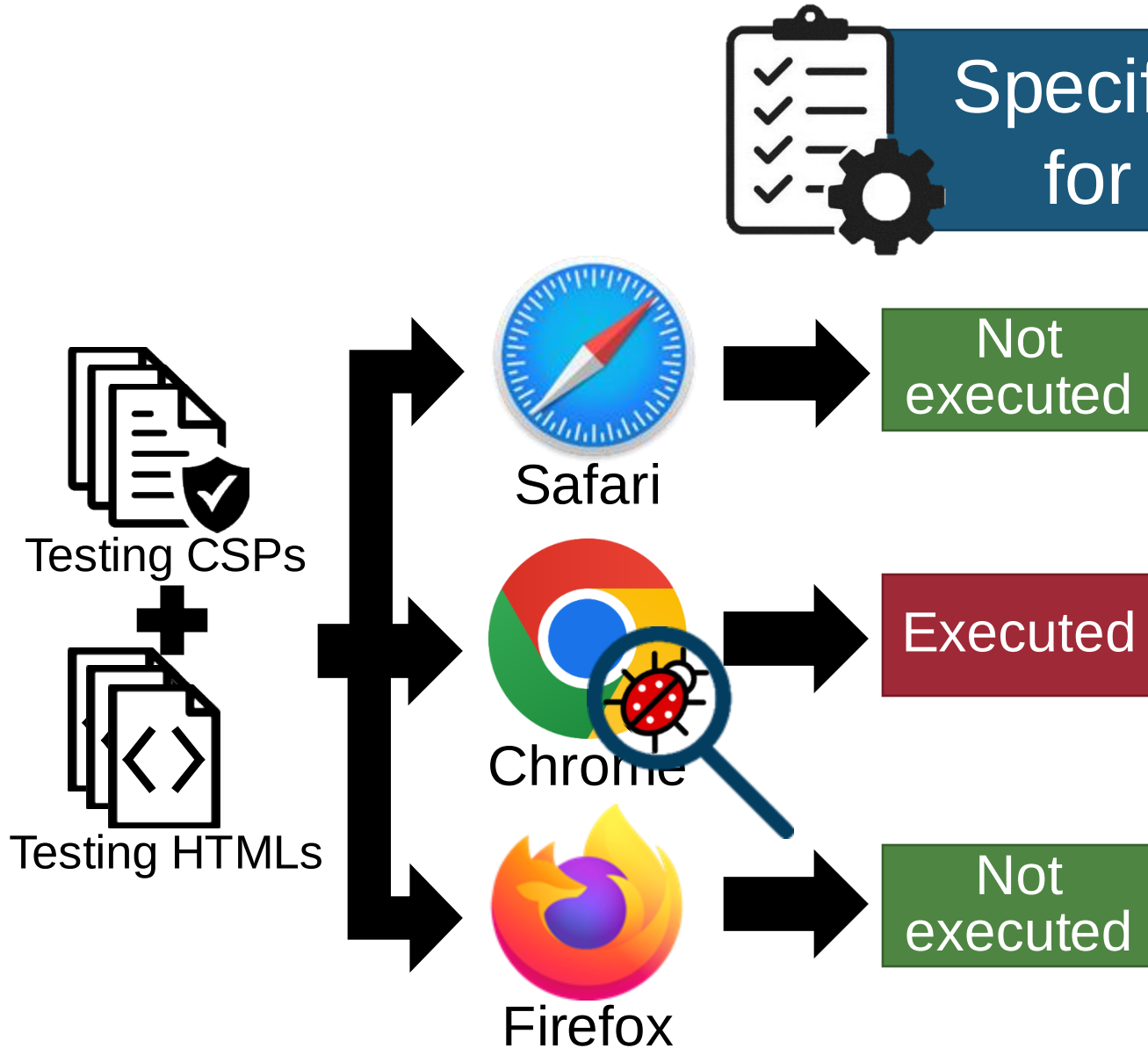


Our approach:
Differential testing!

(FYI) Finding CSP Enforcement



(FYI) Finding CSP Enforcement



Result

- First testing tool for finding CSP enforcement bugs
- Found **27 browser bugs** in Chrome, Safari, and Firefox
- ...and **three description bugs**
- 23 bugs have been patched
- Published in *NDSS'23*

Recent Studies



- DiffCSP, ***NDSS '23***
- 12 angry developers, ***CCS '21***
- Complex security policy?, ***NDSS '20***
- CSP is dead, long live CSP!, ***CCS '16***
- Reining in the web with CSP, ***WWW '10***
- CCSP, ***USENIX Security '17***
- CSPAutoGen, ***CCS '16***

Cross-Site Request Forgery (CSRF)

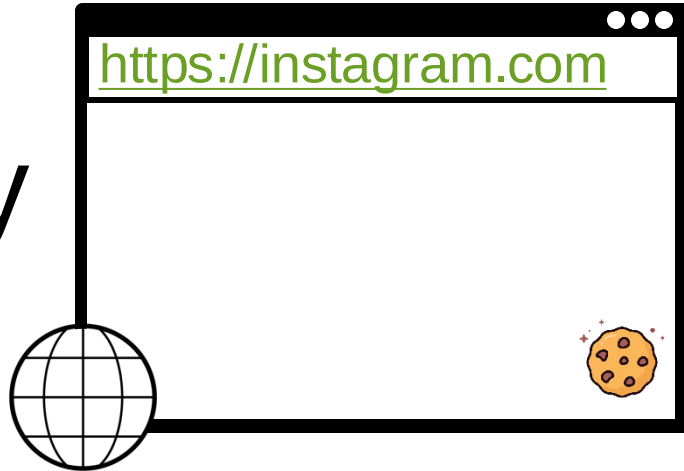
Recap: Cookies



49



1st try



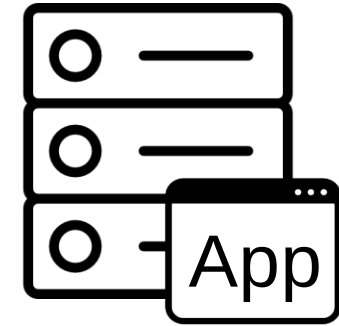
Username: Alice
Password: 1234



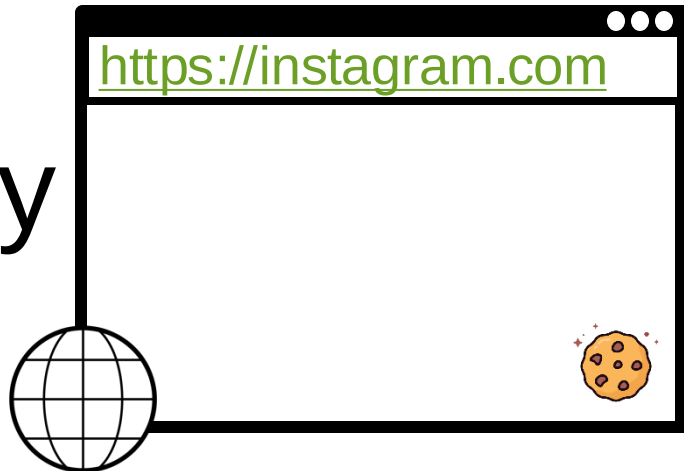
Set-Cookie: SSID:4AEBRE42



instagram.com
web server



2nd try

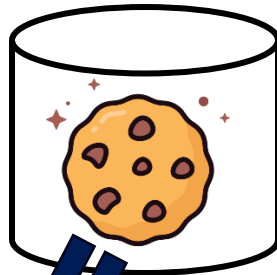
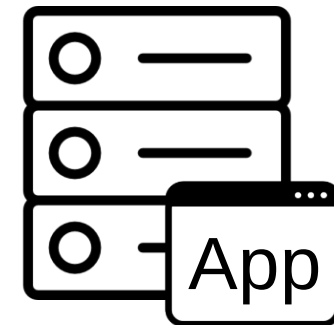


Hey Instagram,
show me my profile
Cookie: SSID:4AEBRE42



Hello Alice!
Here is your profile

instagram.com
web server



?

Recap: Cookies



- Store a server-created file (cookie) in the browser
- Examples
 - Authentication (log in)
 - Personalization (language preference, shopping cart)
 - User tracking

**Q. What if an attacker tricks the user to do unwanted actions?
(e.g., send money to the attacker)**

Regular Website Usage



<https://bank.com/transfer.php>

Destination account:

Amount:

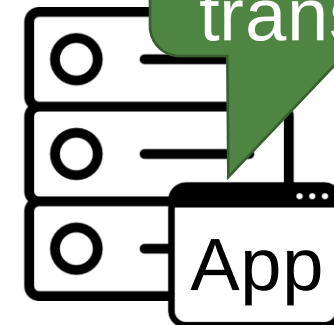
```
<form method="POST"
target="https://bank.com/transfer.php">
  <input type="text" name="to">
  <input type="text" name="amount">
  <input type="submit">
</form>
```



User

To: 123-456-789, Amount: \$50 

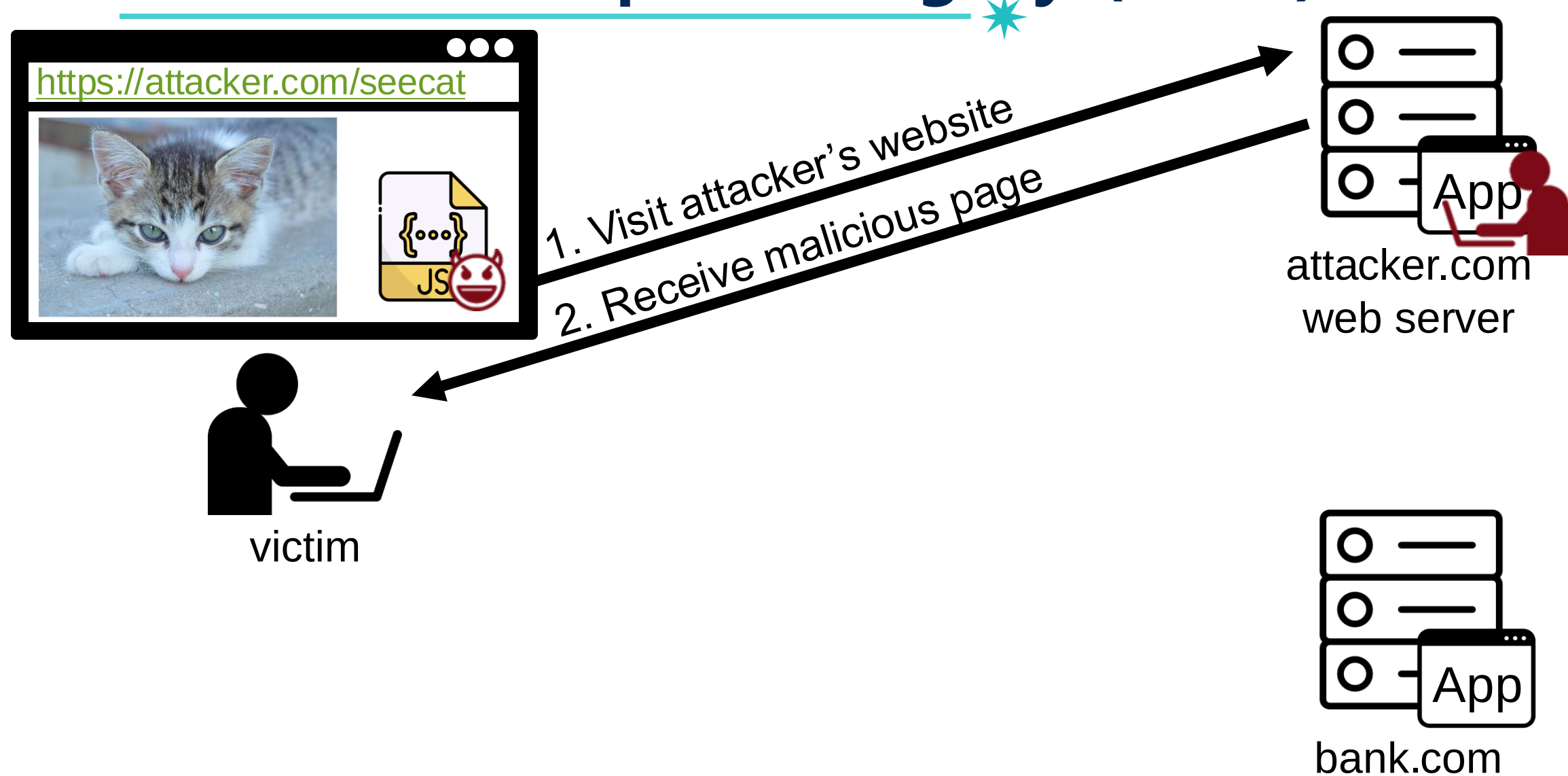
Response: "Transfer OK"



bank.com

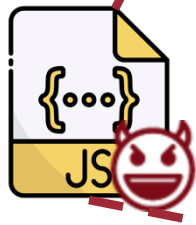
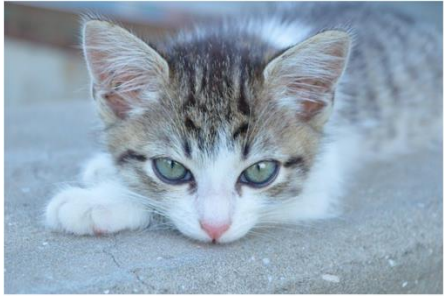
Processing
transaction

Cross-Site Request Forgery (CSRF)



Cross-Site Request Forgery (CSRF)

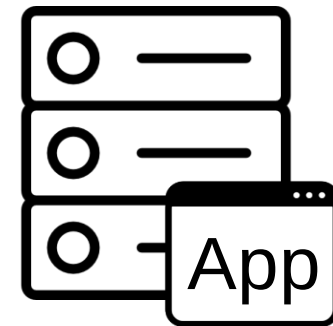
<https://attacker.com/seecat>



```
<form method="POST" id="transfer"
target="https://bank.com/transfer.php">
  <input type="hidden" name="to" value="attacker_account">
  <input type="hidden" name="amount" value="1000000">
</form>
<script>
  transfer.submit();
</script>
```



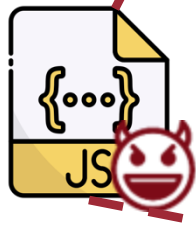
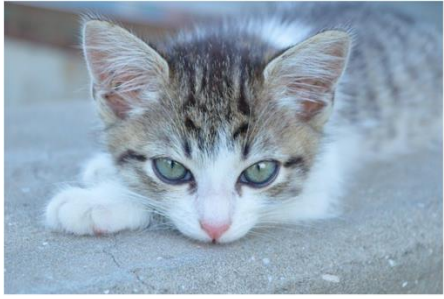
victim



bank.com

Cross-Site Request Forgery (CSRF)

<https://attacker.com/seecat>



```
<form method="POST" id="transfer"
target="https://bank.com/transfer.php">
  <input type="hidden" name="to" value="attacker_account">
  <input type="hidden" name="amount" value="1000000">
</form>
<script>
  transfer.submit();
</script>
```

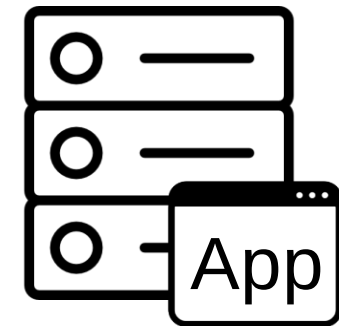
Attacker can control
these values

Send request!

(No user involvement required)



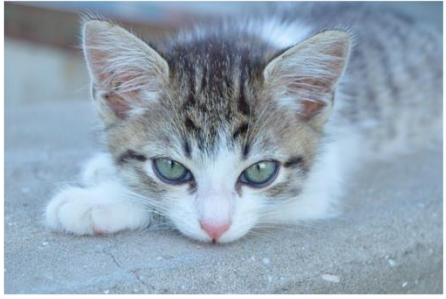
victim



bank.com

Cross-Site Request Forgery (CSRF)

<https://attacker.com/seecat>

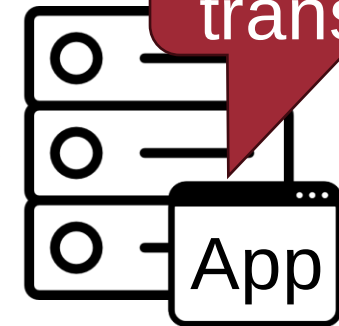


```
<form method="POST" id="transfer"
target="https://bank.com/transfer.php">
  <input type="hidden" name="to" value="attacker_account">
  <input type="hidden" name="amount" value="1000000">
</form>
<script>
  transfer.submit();
</script>
```



victim

To: attacker_account, Amount: \$1,000,000



bank.com

Processing transaction

Cross-Site Request Forgery

Cross-Site Request Forgery (CSRF)

- Force a user to execute unwanted actions (e.g., changing state) on an **authenticated** web application
- Attack works for GET ...
 - Invisible images, hidden iframes, css files, scripts, ...

```
  
</img>
```

Cross-Site Request Forgery (CSRF)

- Force a user to execute unwanted actions (e.g., changing state) on an **authenticated** web application
- Attack works for GET ...
 - Invisible images, hidden iframes, css files, scripts, ...

```
  
</img>
```

Browser send request on behalf of the user (victim)

The image is not visible, but the request goes out

Cross-Site Request Forgery (CSRF)

- Force a user to execute unwanted actions (e.g., changing state) on an **authenticated** web application
- Attack works for GET ...
 - Invisible images, hidden iframes, css files, scripts, ...

```

</img>
```

- and POST

```
<form method="POST" action="https://bank.com/transfer.php" id="transfer">
  <input type="hidden" name="act-to" value="attacker_account">
  <input type="hidden" name="amount" value="100000">
</form>
<script>
  transfer.submit()
</script>
```

Cross-Site Request Forgery (CSRF)

- Force a user to execute unwanted actions (e.g., changing state) on an **authenticated** web application

Very important point:
A web page can **send information to any site!**

- and POST

```
<form method="POST" action="https://bank.com/transfer.php" id="transfer">
  <input type="hidden" name="act-to" value="attacker_account">
  <input type="hidden" name="amount" value="100000">
</form>
<script>
  transfer.submit()
</script>
```

SOP Does Not Control Sending



- SOP violation? Nope!
- Same origin policy (SOP) controls access to DOM
- **Active content (scripts) can send a request anywhere!**
 - No user involvement required

CSRF on Netflix 2006

61



CSRF on Netflix 2006



- CSRF vulnerabilities at Netflix allowed the attacker to do:
 - Add movies to your rental queue
 - Add a movie to the top of your rental queue
 - Change the name and address of a victim's account
 - Change the email and password on a victim's account

```
  
</img>
```

How to Defense CSRF Attacks?

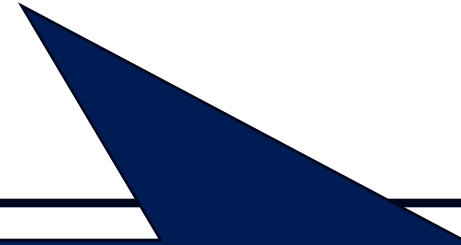
63

- Referrer checking: “where is this request coming from?”
 - Accept requests only if their referrer is the same as the server (e.g., *.bank.com)

Recap: Referrer Header



```
GET /cse467.html HTTP/1.1
Host: websec-lab.com
Accept-Language: en
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;)
Referer: http://google.com
```



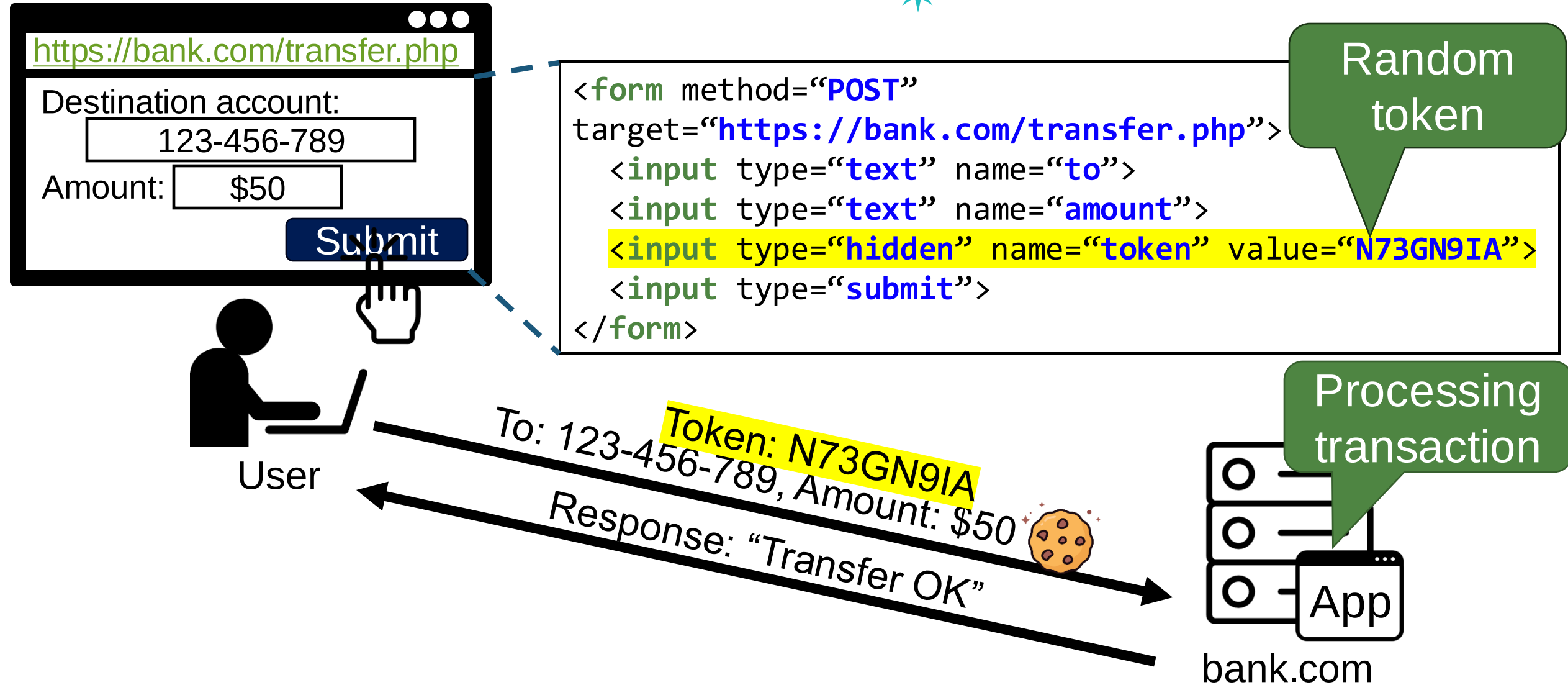
Contain the address from which a resource has been requested

How to Defense CSRF Attacks?

65

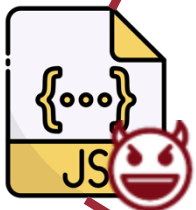
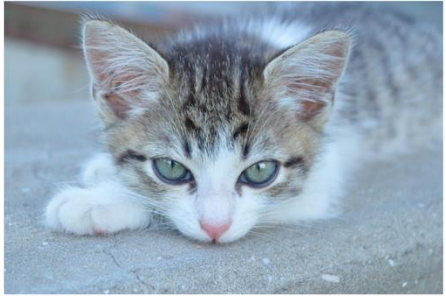
- Referrer checking: “where is this request coming from?”
 - Accept requests only if their referrer is the same as the server (e.g., *.bank.com)
- Secret validation token
 - For each session, a fresh secret token is generated by the server
 - Send requests with the token
 - Accept requests only if the token is valid

Secret Validation Token: Regular Usage



Secret Validation Token: Preventing CSRF 67

<https://attacker.com/seecat>



```
<form method="POST" id="transfer"
target="https://bank.com/transfer.php">
  <input type="hidden" name="to" value="attacker_account">
  <input type="hidden" name="amount" value="1000000">
  <input type="hidden" name="token" value="noclude">
</form>
<script>
  transfer.submit();
</script>
```



victim

To: attacker_account, Amount: \$1,000,000



bank.com

Invalid token:
No transaction

How to Defense CSRF Attacks?

68

- Referrer checking: “where is this request coming from?”
 - Accept requests only if their referrer is the same as the server (e.g., *.bank.com)
- Secret validation token
 - For each session, a fresh secret token is generated by the server
 - Send requests with the token
 - Accept requests only if the token is valid
- SameSite Cookies

Same-Site Cookies



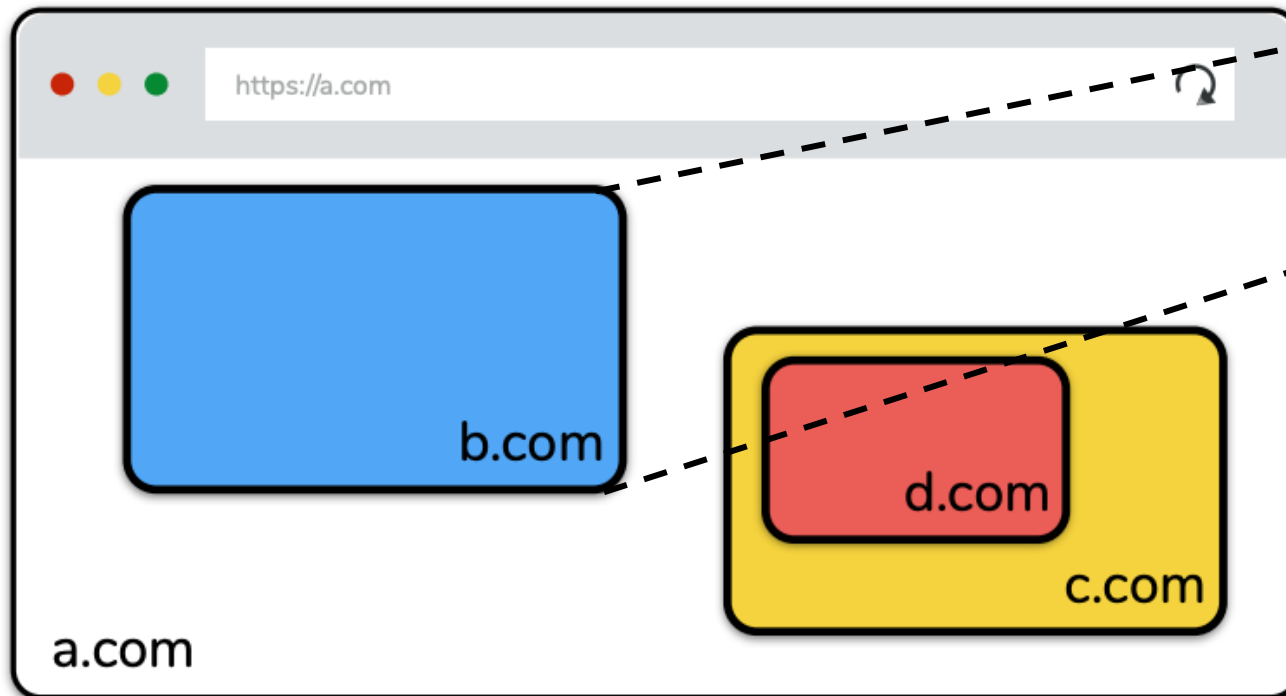
- Two modes
 - Strict: browser will **NEVER** send cookies with cross-origin request

```
Set-Cookie: session=0F8tgd0hi9ynR1M9wa30Da; SameSite=Strict
```
 - Lax: browser will send the cookie in cross-site requests, but only if both of the following conditions are met:
 - The request uses safe requests (e.g., GET)
 - The request resulted from a top-level navigation by the user, such as clicking on a link
- Until May 2018, only supported by Chrome and Opera
- Since Chrome 80, defaults to SameSite=lax

Clickjacking

Recap: Browser Execution Model

- Windows may contain frames from different sources
 - Frame**: rigid visible division
 - iFrame**: floating inline frame



```
<iframe src="b.com">  
</iframe>
```

Framing other Websites

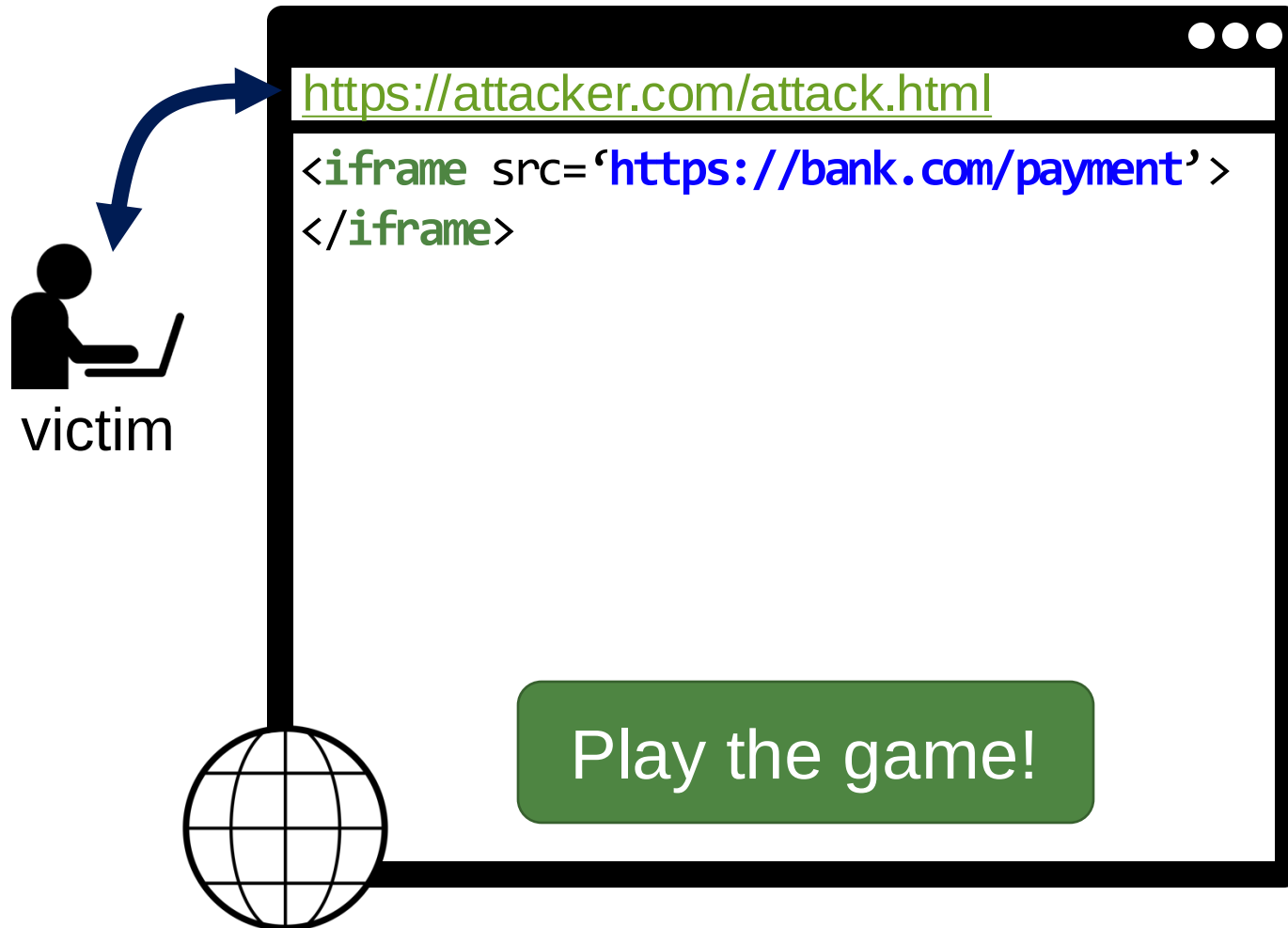


- HTML supports framing of other (cross-origin sites)
 - E.g., iframes
 - Very useful feature for advertisement, like buttons, ...
- Embedding site controls most of the frame's properties
 - How large the frame should be
 - Where the frame is displayed
 - How opaque the frame should be....

What could go wrong?

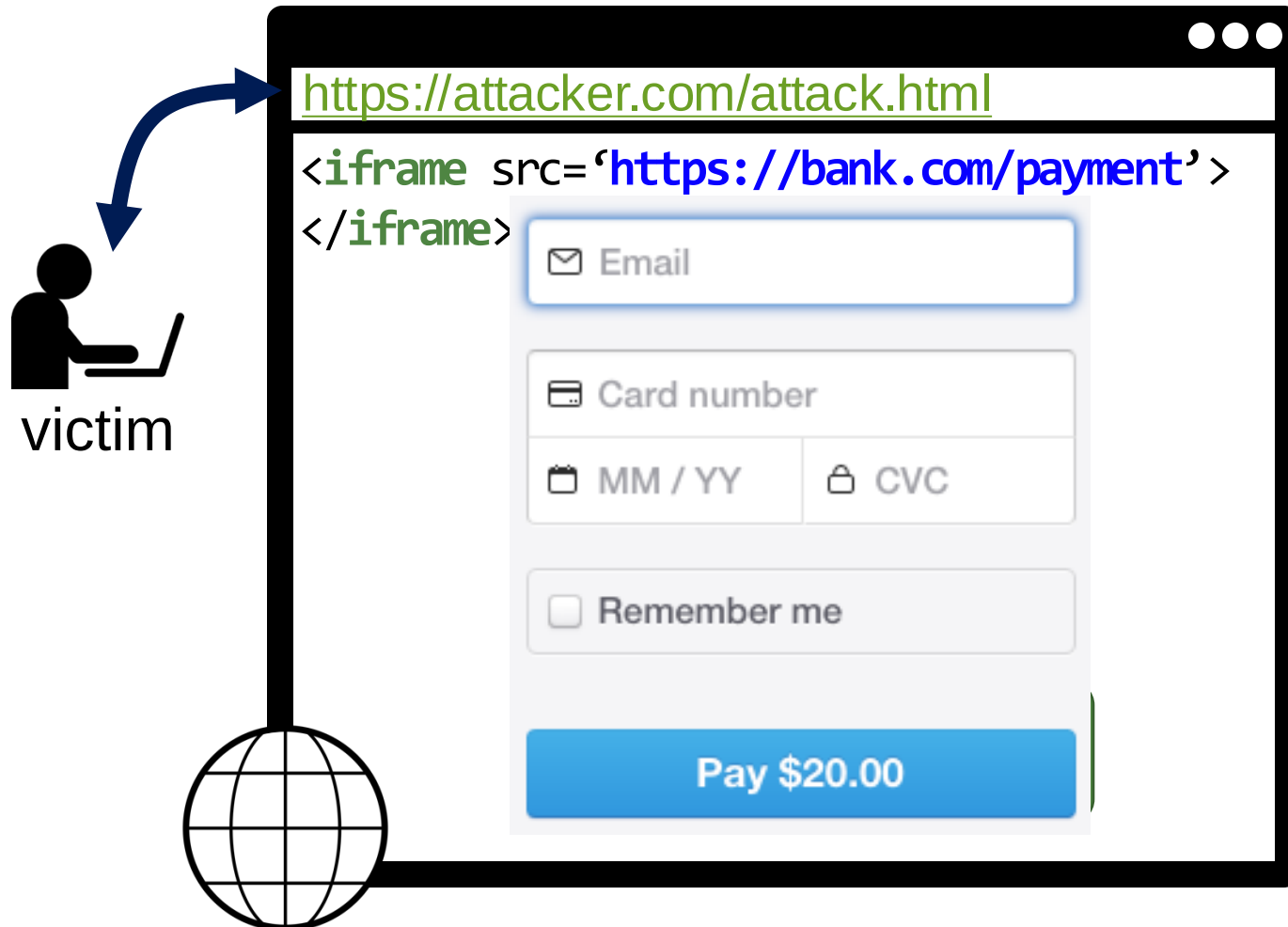
Clickjacking (UI Redressing)

- Attacker **overlays transparent or opaque frames** to trick a user into clicking on a button or link on another page



Clickjacking (UI Redressing)

- Attacker **overlays transparent or opaque frames** to trick a user into clicking on a button or link on another page



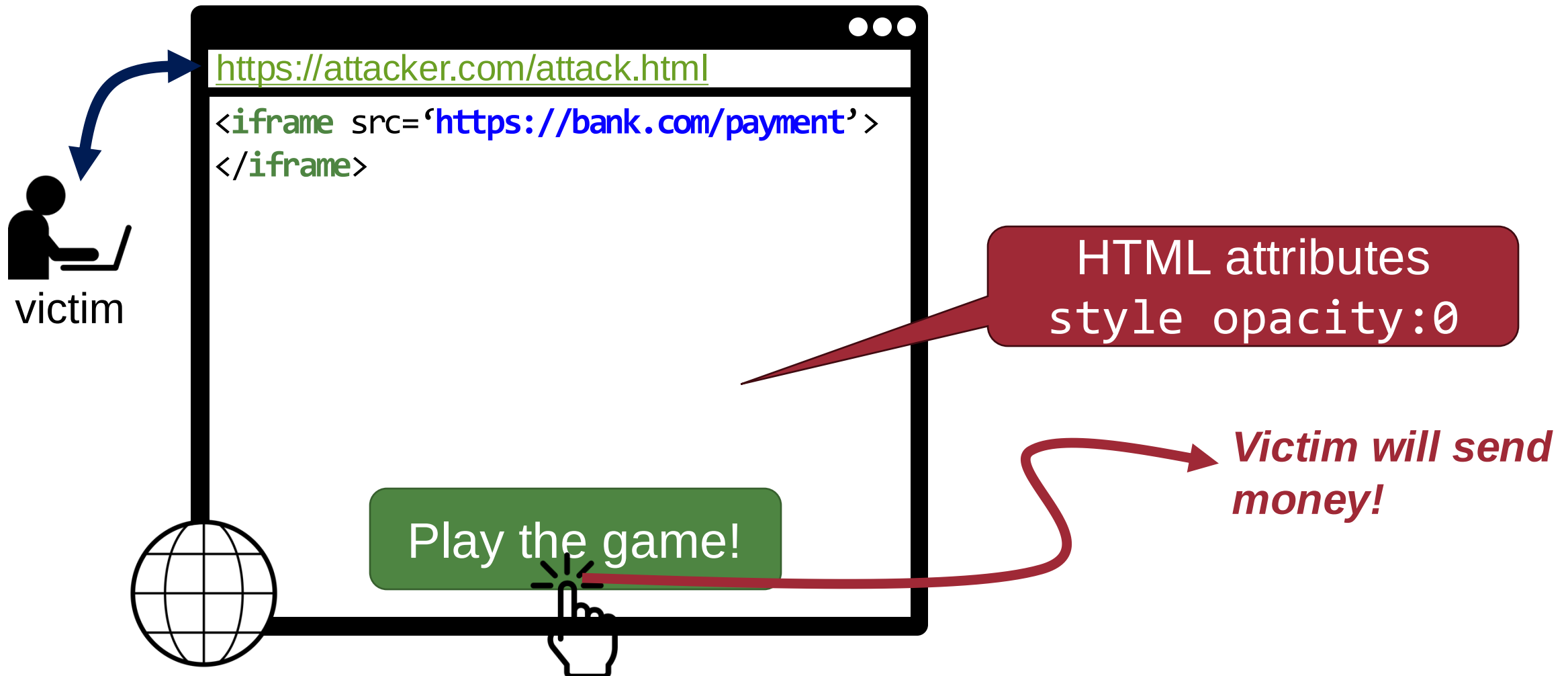
Clickjacking (UI Redressing)

- Attacker **overlays transparent or opaque frames** to trick a user into clicking on a button or link on another page



Clickjacking (UI Redressing)

- Attacker **overlays transparent or opaque frames** to trick a user into clicking on a button or link on another page



Clickjacking Demo

77



- Demo: https://websec-lab.github.io/courses/2025s-cse467/demo/clickjacking_demo1.html



Clickjacking – Hiding the Target Element 78

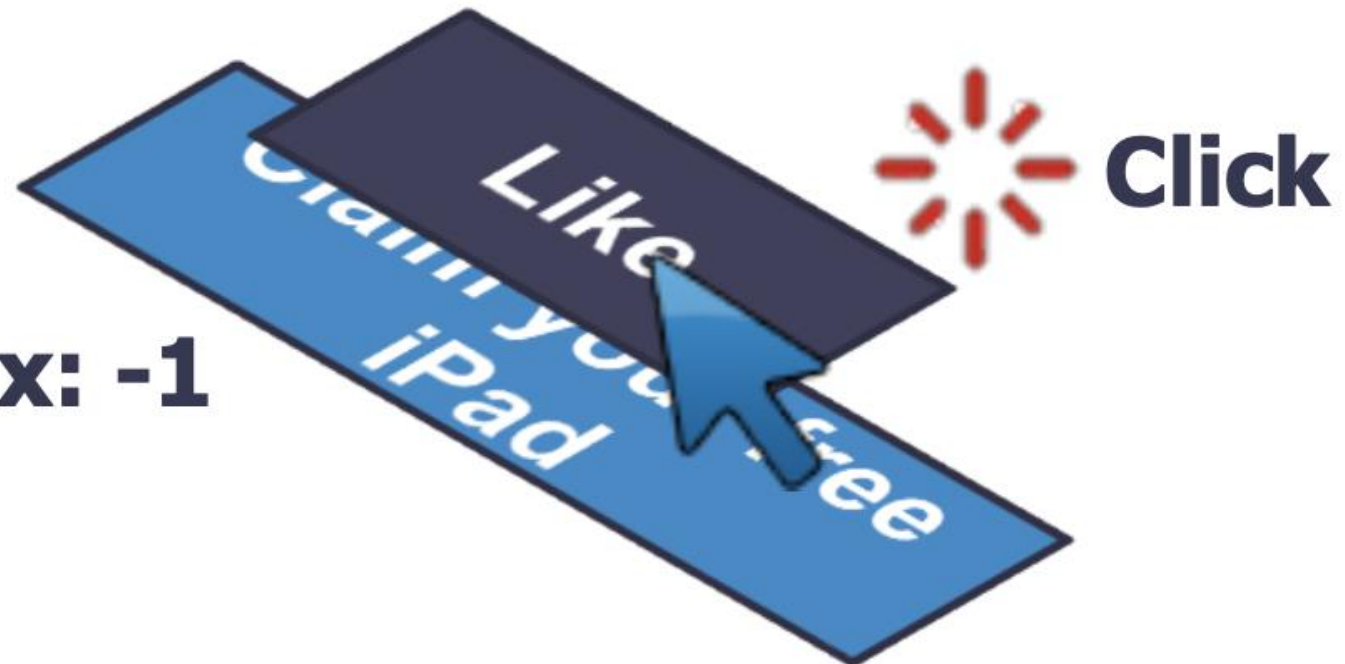
- Use CSS opacity property and z-index property

Make other element float under the target element

opacity: 0.1

Hide target element

z-index: -1



Clickjacking - Fake Cursors

- Use CSS cursor property and JavaScript to simulate a fake cursor icon on the screen

Real cursor icon



Fake cursor icon



Clickjacking - Fake Cursors

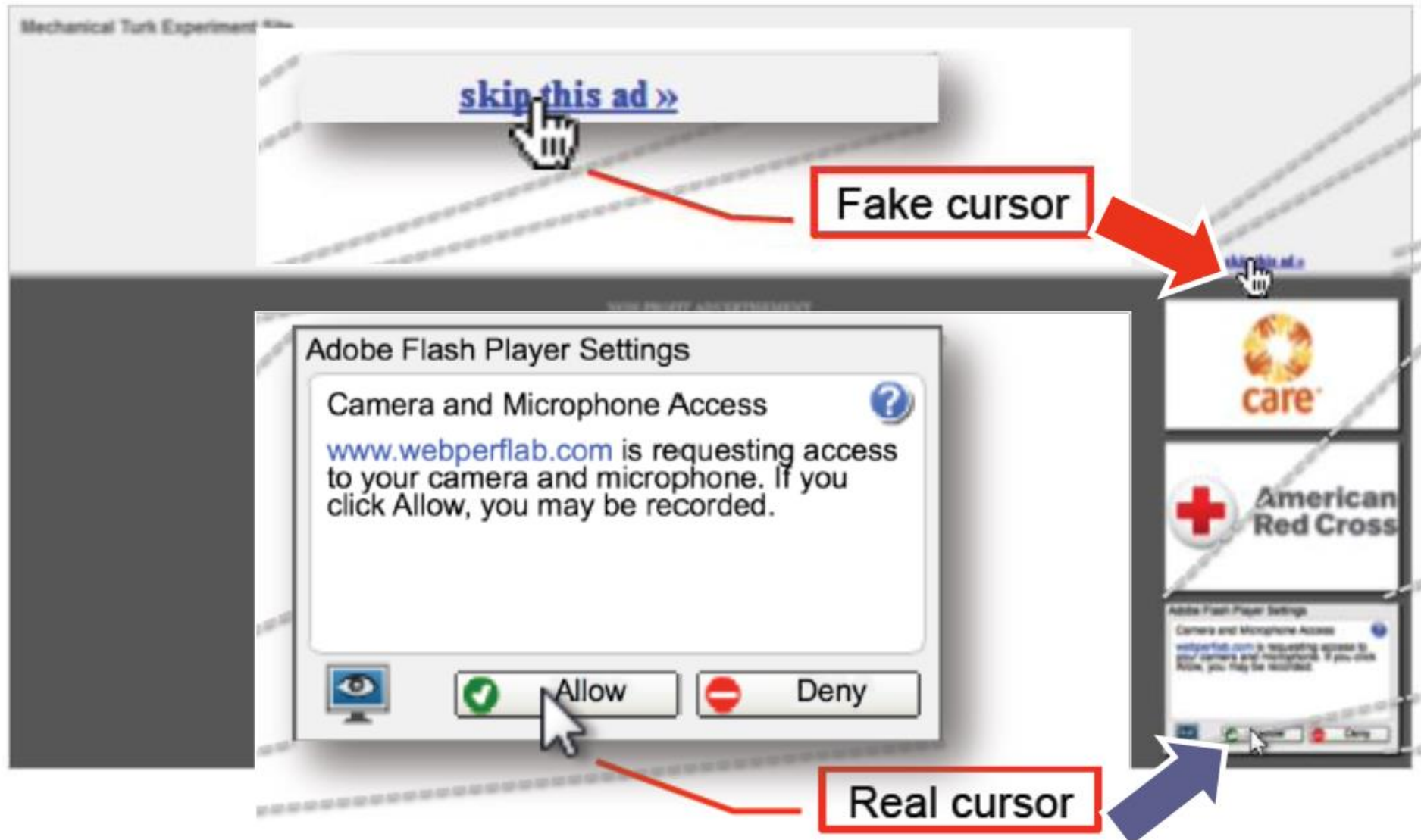
- Use CSS cursor property and JavaScript to simulate a fake cursor icon on the screen

Hide real cursor icon
by using `cursor: none`

Fake cursor icon



Cursor Spoofing



How to Prevent Clickjacking?



82

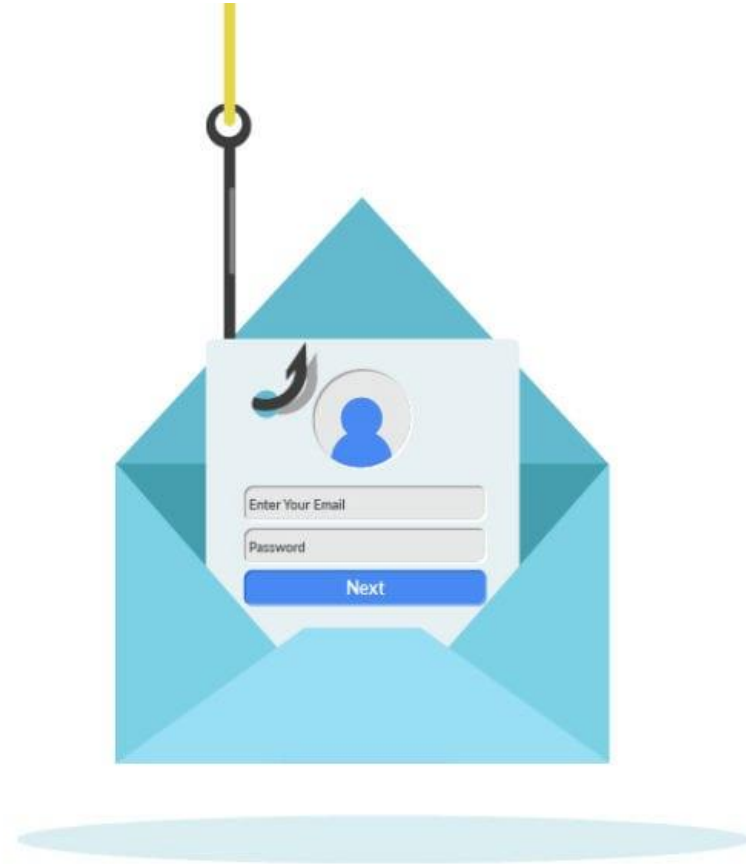
- Frame busting
 - Make sure that my website is not loaded in an enclosing frame

```
if (top !== self)
  top.location = self.location
```

JS

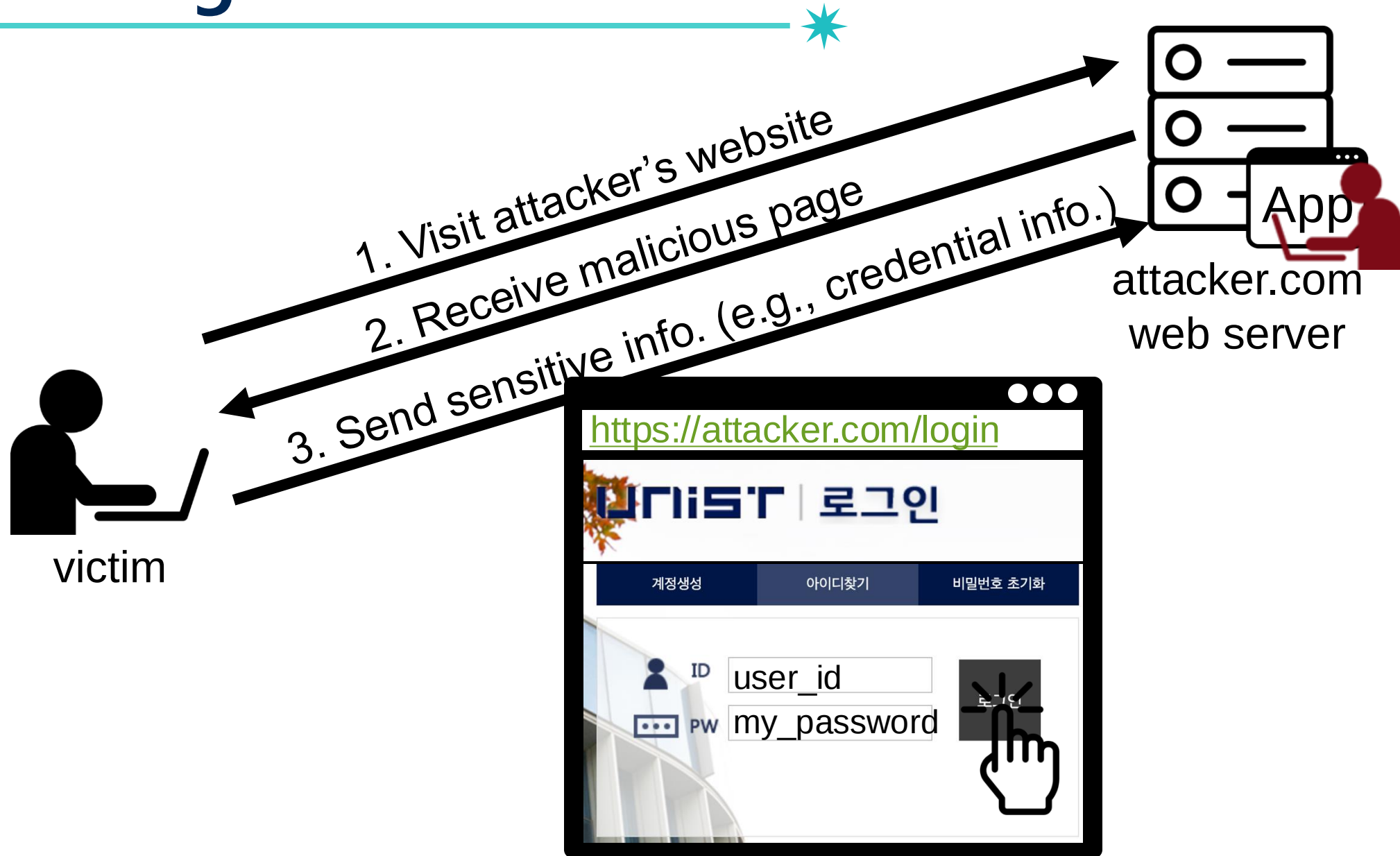
- Use CSP's frame-ancestors
 - Determine whether my website may be embedded in another site
 - 'none': denies from any host
 - 'self': allows only from same origin
 - `http://example.org`: allows specific origin

Phishing



Phishing

84



Phishing



- Disguising as a trustworthy entity, and obtain private information
 - Login credentials
 - Financial records
- Links to phishing webpages dispatched to victims through email or SMS
- According to a report from the FBI, it received 800,944 reports of phishing, with losses exceeding \$10.3 billion in 2022

Outlook



Sign in

to continue to Outlook

Email address, phone number, or Skype

No account? [Create one!](#)

[Can't access your account?](#)

[Sign-in options](#)

Next

Phishing

From: apple,Inc <Update.account.confirmed@altervista.org>
To:
Sent: Thursday, April 24, 2014 12:35 PM
Subject: Update your Account information !



Dear iTunes Customer!

Your itunes account has been frozen because we are unable to validate your account information. Once you have updated your account records, we will try again to validate your information and your account suspension will be lifted. This will help protect your account in the future. This process does not take more than 3 minutes. To proceed to confirm your account details please click on the link below and follow the instructions.

[Get Started](#) ▼

If you need our Help left by clicking the Help link located in the upper right-hand corner of any Apple page. .

Sincerely,

Apple Inc

Please do not reply to this email. We are unable to respond to inquiries sent to this address. For immediate answers to your questions, visit our Help left by clicking "Help" at the top of any Apple page.

Copyright © 2014 Apple Inc. All rights reserved. Apple is located at 2211 N. First St., San Jose, CA 95131.

Typical Properties of Spoofed Sites



- Attackers manually copy/recreate web content from target website
 - Show logos found on the honest site
- Have suspicious URLs: mostly, being camouflaged as a URL that looks familiar to people
 - E.g., umist.ac.kr
- Ask for user input
 - Debit card number, username, password, ...
- Phishing content served from attacker-owned web server
 - Or a compromised web server

Safe to Type Your Password?

89



Spear Phishing



- Phishing attempts directed at specific individuals
- This can increase the likelihood of success, as the sender appears more credible and informed



Spear Phishing

91



From: UDEL HR <hremployeepayroll@udel.edu>

Date: August 13, 2015 at 12:48:29 PM EDT

To: <[REDACTED]>

Subject: Your August 2015 Paycheck



Hello,

We assessed the 2015 payment structure as provided for under the terms of employment and discovered that you are due for a salary raise starting August 2015.

Your salary raise documents are enclosed below:

[Access the documents here](#)

Faithfully

Human Resources

University of Delaware

Spear Phishing

92



Mrs. Füsun Tümsavaş <jsc7339@gmail.com>

09-12 (화) , 오전 1:01

Seongil Wi <seongil.wi@unist.ac.kr> ✉

↩ 전체 회신

지운 편지함

Hello Seongil Wi,

I am contacting you for the receipt of the sum of US\$9,500,000.00 (Nine Million Five Hundred United State **Dollars**) only.

Please Let me know if you are interested,

Regards,

Mrs. Füsun Tümsavaş

How to Detect Phishing?



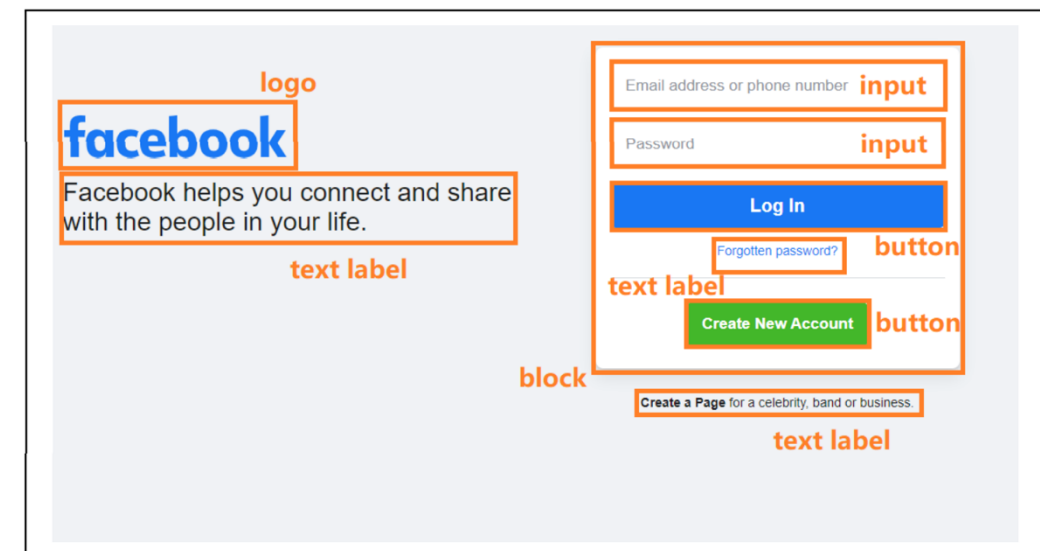
- Crowdsourcing, Blacklisting
 - lists reported phishing URLs
 - E.g., <https://openphish.com/>

Phishing URL	Targeted Brand	Time
https://diepost-zoll-ch.com/steps/	Generic/Spear Phishing	06:24:20
https://257.nhksf.com/	Tencent	06:18:49
https://ffspind7my.terbaru-2023.com/vhsfhqpdhdxih1	Garena	06:17:46
https://web.telegram.data-bees.cn/	Telegram	06:17:26
http://wantlengtime.com/	WhatsApp	06:16:08
http://manualmetarestore-39f.pages.dev/	Crypto/Wallet	06:15:36

How to Detect Phishing?

- Crowdsourcing, Blacklisting
 - lists reported phishing URLs
 - E.g., <https://openphish.com/>
- URL-based pattern detection
 - E.g., A URL is phishy if its length ≥ 76
 - E.g., Brand name modification with '-'
 - youtube-x.com
- Content-based pattern detection

- Require human intervention and verification
- Phishers are starting to use one-time URLs



Conclusion



- Content Security Policy (CSP)
 - Allow resources which are trusted by the developer
- Cross-Site Request Forgery (CSRF)
 - Force a user to execute unwanted actions
- Clickjacking
 - Attackers overlays opaque frames to trick a user
- Phishing
 - Disguising as a trustworthy entity, and obtain private information

Question?