Internet Security

VU 188.366

Web Application Security 3

Adrian Dabrowski, Johanna Ullrich, Aljosha Judmayer,
Georg Merzdovnik, and **Christian Kudera**

inetsec@seclab.tuwien.ac.at

Overview

- More on session attacks
  - continuation of last lecture (session highjacking)
- Cross Site Request Forgery
- Man-in-the-Middle attacks against HTTPS
  - https cookie stealing
  - SSL Stripping
  - “Cryptocalypse”
- Live Demo

Session Attacks

What are Session IDs again?

- HTTP is a stateless protocol:
  - it does not “remember” previous requests
- Web applications must create and manage sessions themselves
- Session data is
  - stored at the server
  - associated with a unique Session ID
- After session creation, the client is informed about the session ID
  - The client attaches the session ID to each request

Session Attacks

- Targeted at "stealing" the session ID
- If I know the session ID of a currently logged in user I can impersonate him

Session Attacks

- Targeted at stealing the session ID
- Interception:
  - intercept request or response and extract session ID
- Prediction:
  - predict (or make a few good guesses about) the session ID
- Brute Force:
  - make many guesses about the session ID
- Fixation:
  - make the victim use a certain session ID

NEW!

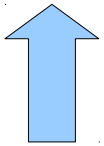
- The first three attacks can be grouped into “Session Hijacking“ attacks
  - we discussed them already

Session Hijacking Attacks

- Interception:
  - based on sniffing traffic
- Prediction/Brute forcing
  - possible if IDs are not random enough

Session Hijacking Attacks

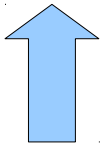
- Interception:
  - based on sniffing traffic
- Prediction/Brute forcing
  - possible if IDs are not random enough



- Prevention:
  - use SSL for each request/response that transports a session ID
  - not only for login!

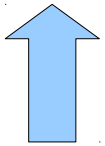
Session Hijacking Attacks

- Interception:
  - based on sniffing traffic



- Prevention:
  - use SSL for each request/response that transports a session ID
  - not only for login!

- Prediction/Brute forcing
  - possible if IDs are not random enough



- Prevention:
  - use large enough, truly random session IDs

Session Fixation Attacks

- So far, we have focused on preventing the attacker from obtaining session credentials
- This approach, however, ignores the possibility of the attacker “issuing” a session ID to the user’s browser
  - The browser then uses a “chosen” session

- In a session fixation attack, the attacker fixes the user’s session ID before the user even logs into the target server
  - What does this mean? The session ID does not have to be stolen

Session Fixation Attacks

- Session management mechanisms can be classified as: *permissive* and *strict*
- ***Permissive*** are those that accept arbitrary session IDs from browser (e.g., Macromedia JRun Server, PHP)
- ***Strict*** are those that only accept sessions that they have created (e.g., MS IIS)

Simple Scenario (permissive session management)

- Suppose we have a bank *online.worldbank.dom*, that uses *permissive* session management
- When the web site is accessed, a session ID is transported via URL parameter sessionid
  - **Attacker picks a sessionid=1234**
  - Attacker sends sessionid to victim and tricks him into clicking on it: http://online.worldbank.dom/login.jsp?sessionid=1234
  - The user clicks on the link and is taken to the banking application login page
  - The web application sees that a session has been assigned and does not issue a new one. Session is bound to new user
  - The user is prompted to log in and provides his credentials
  - Attacker can access victim's account

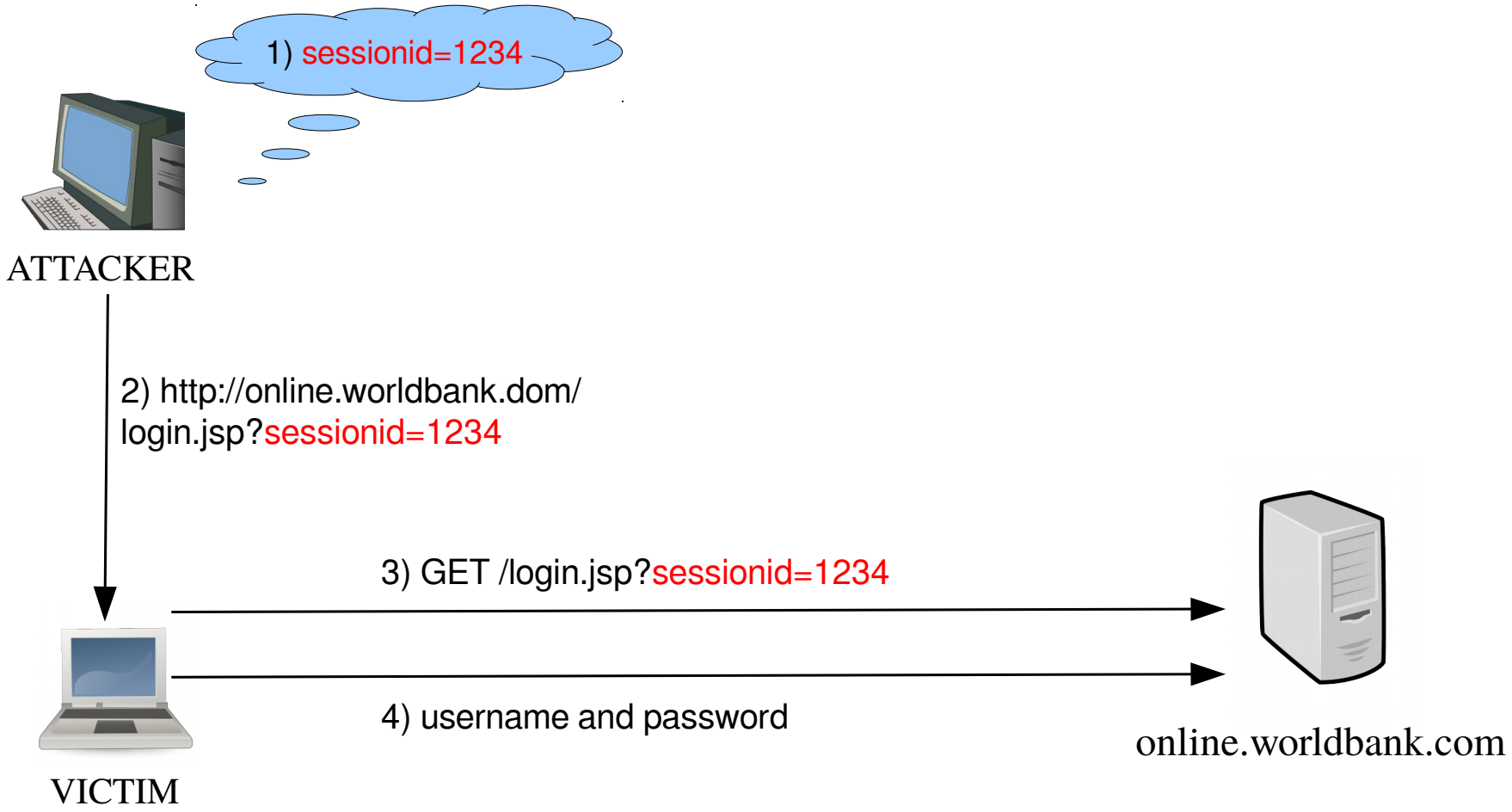
Simple Scenario


VICTIM

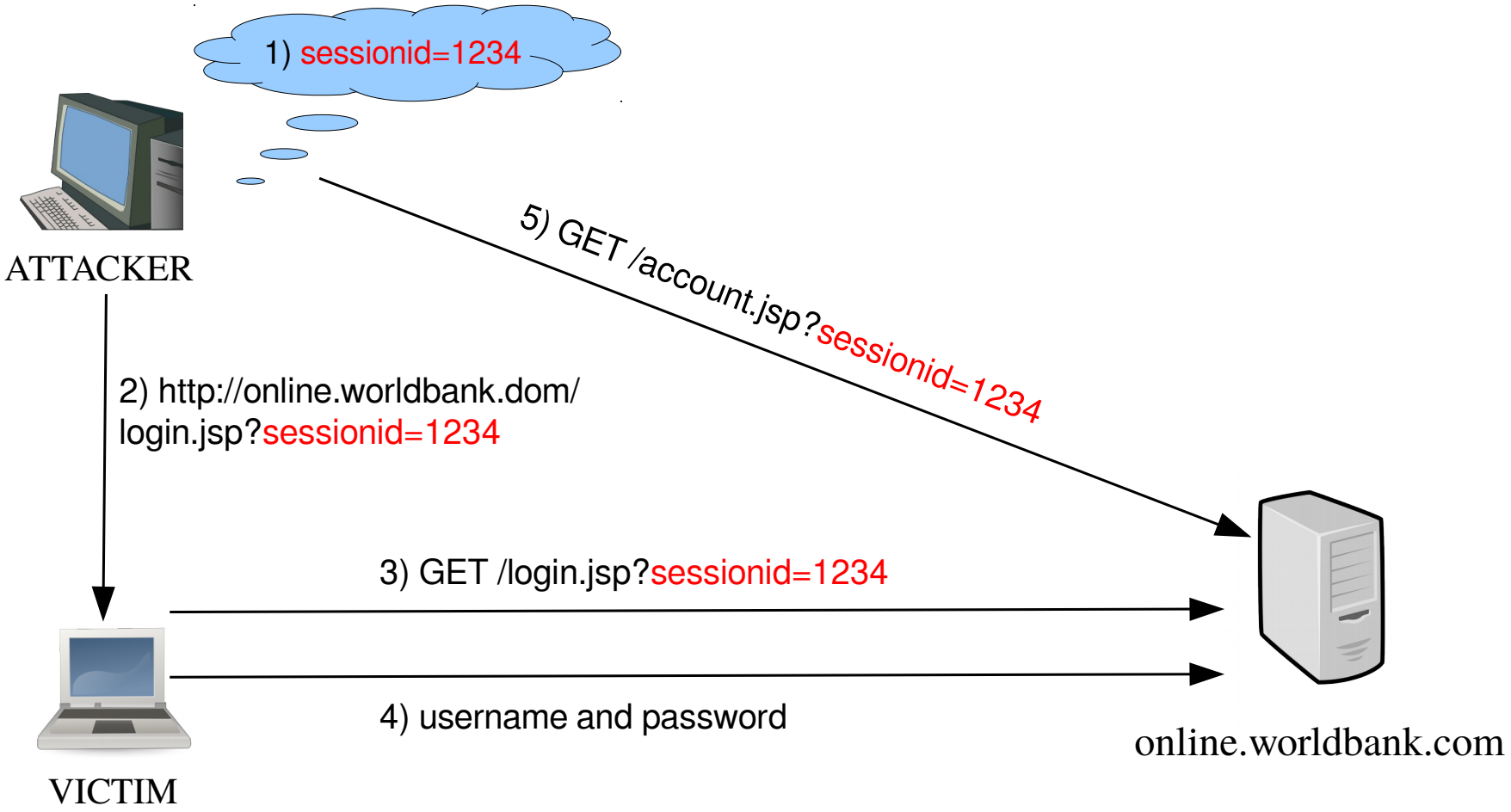

online.worldbank.com

Simple Scenario

Simple Scenario



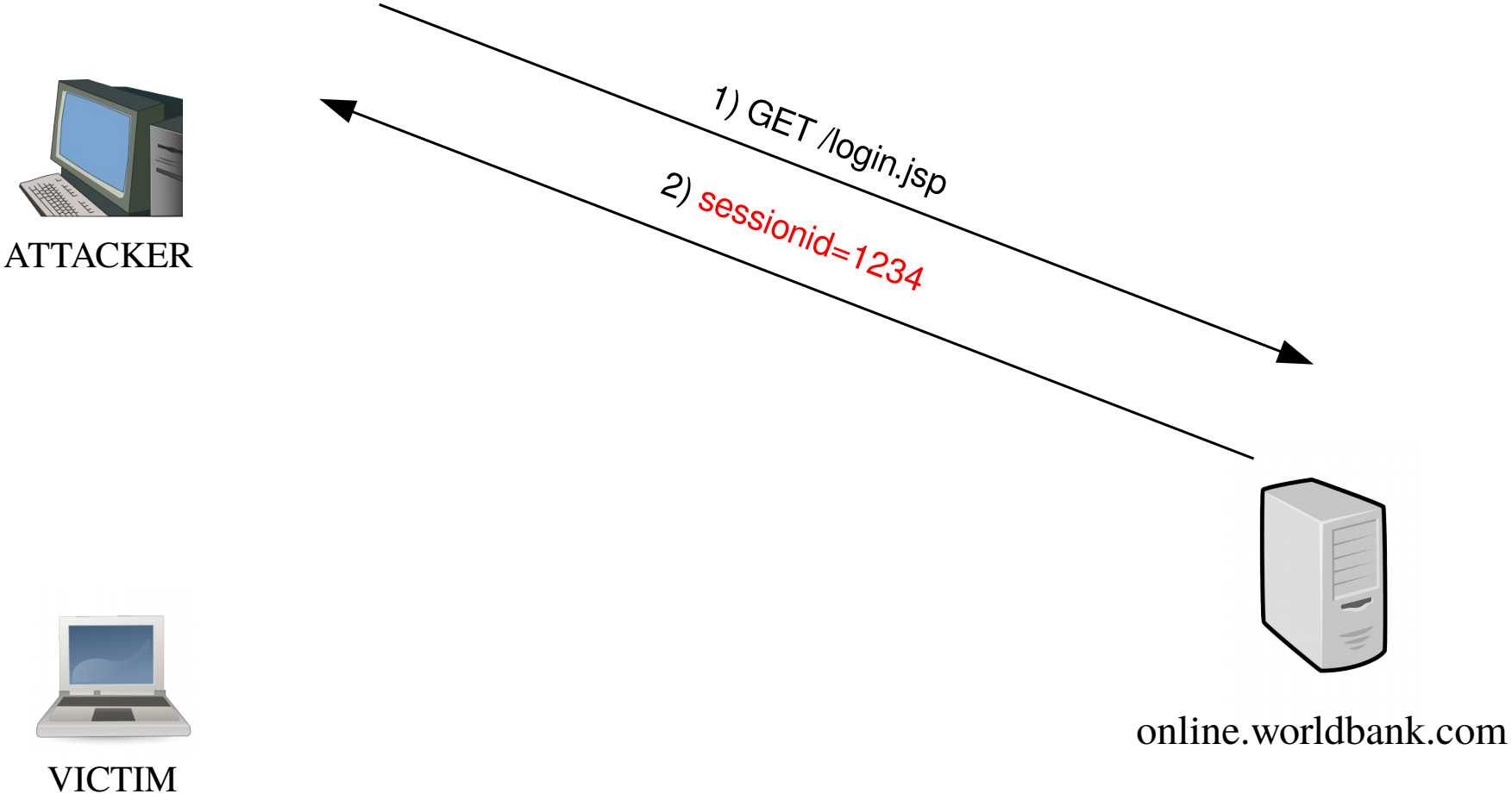
Simple Scenario



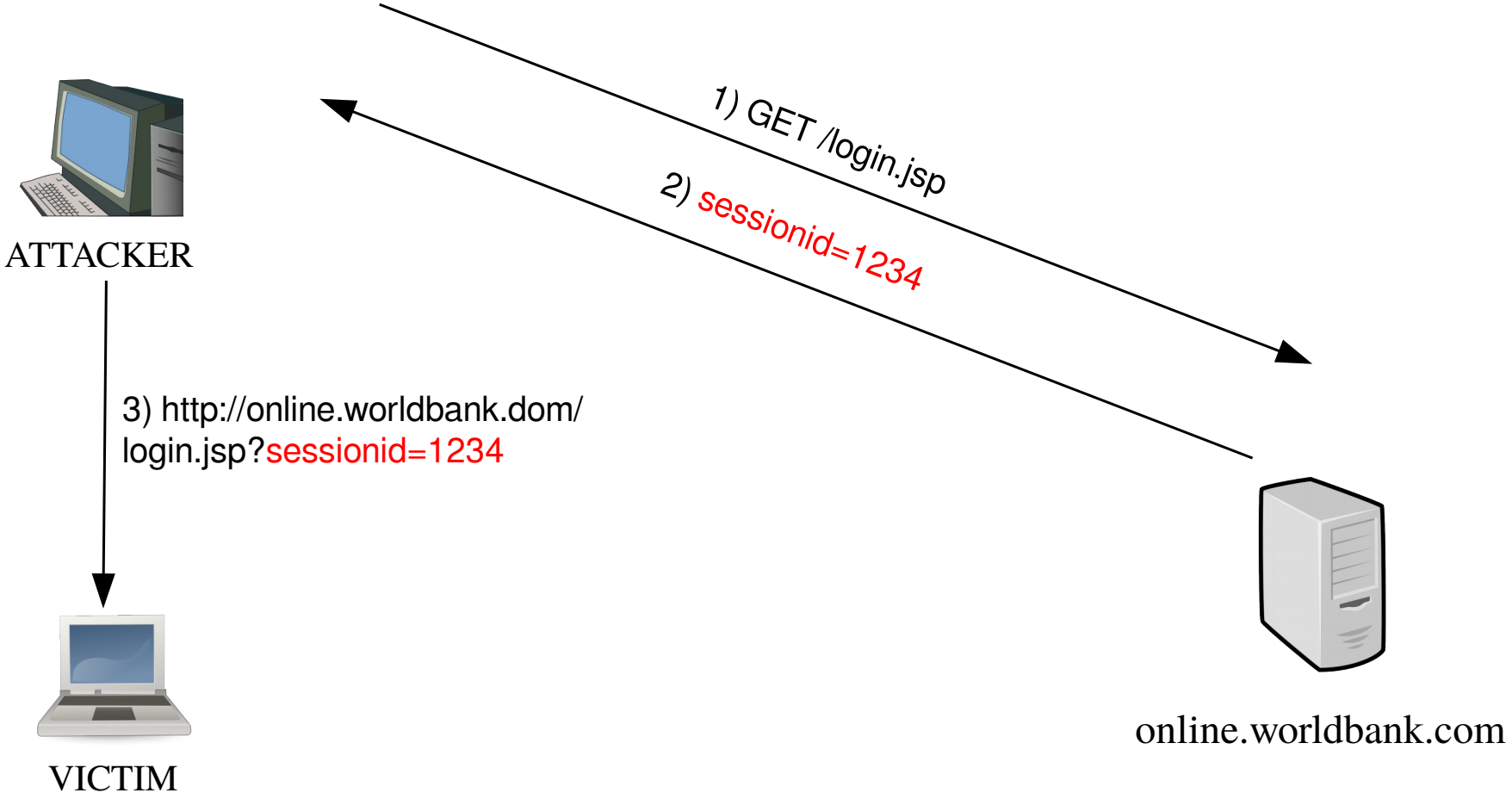
Another Scenario (strict session managment)

- **Strict session** management: the server checks it generated the sessionid
- When the web site is accessed, a session ID is transported via URL parameter sessionid
  - **Attacker sends a request to server**
  - **Server issues attacker a sessionid=1234**
  - **Attacker sends sessionid to victim and tricks him into clicking on it: http://online.worldbank.dom/login.jsp?sessionid=1234**
  - The user clicks on the link and is taken to the banking application login page
  - The web application sees that a session has been assigned and does not issue a new one. Session is bound to new user
  - Attacker can access victim's account

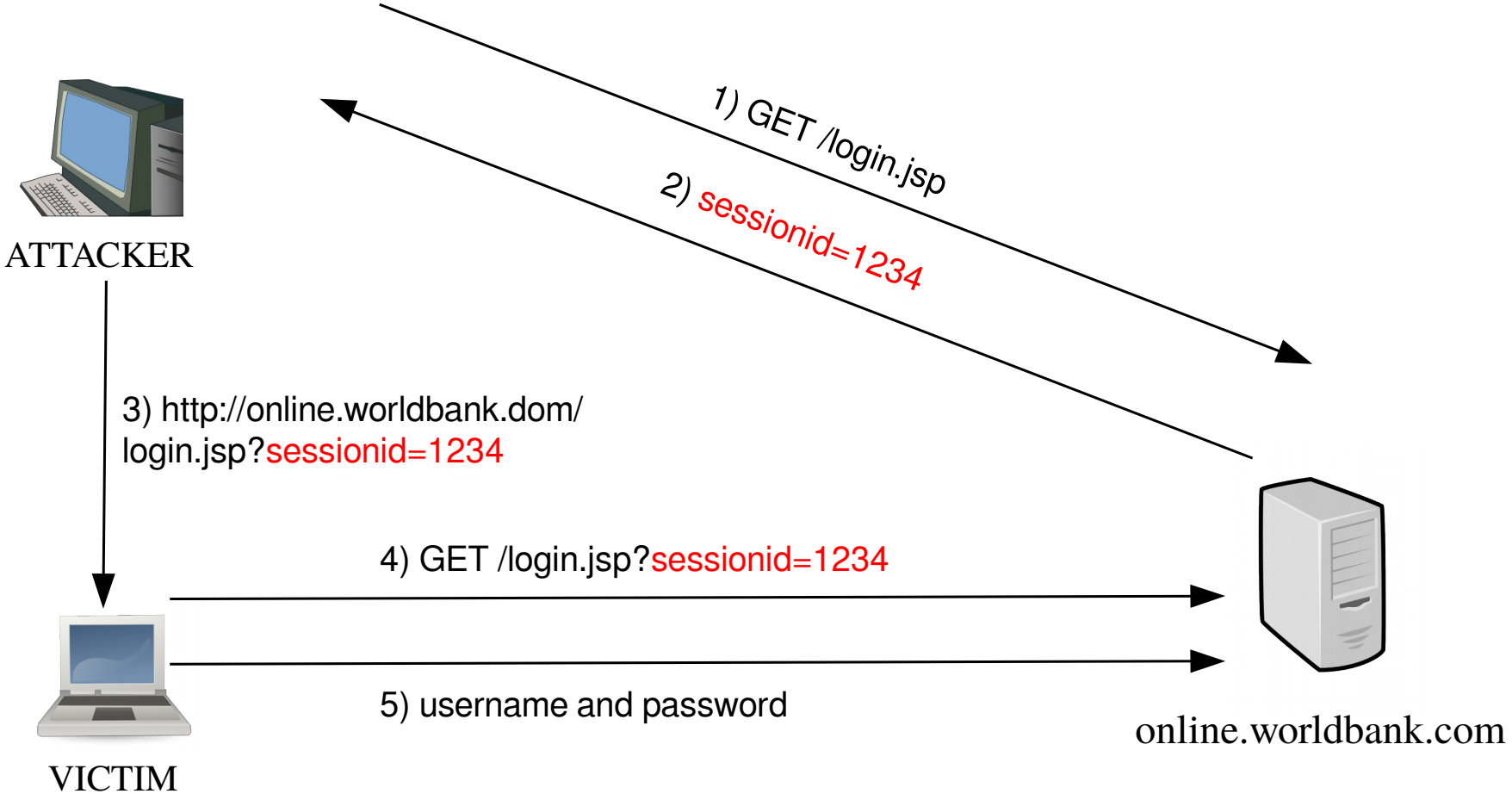
Another Scenario



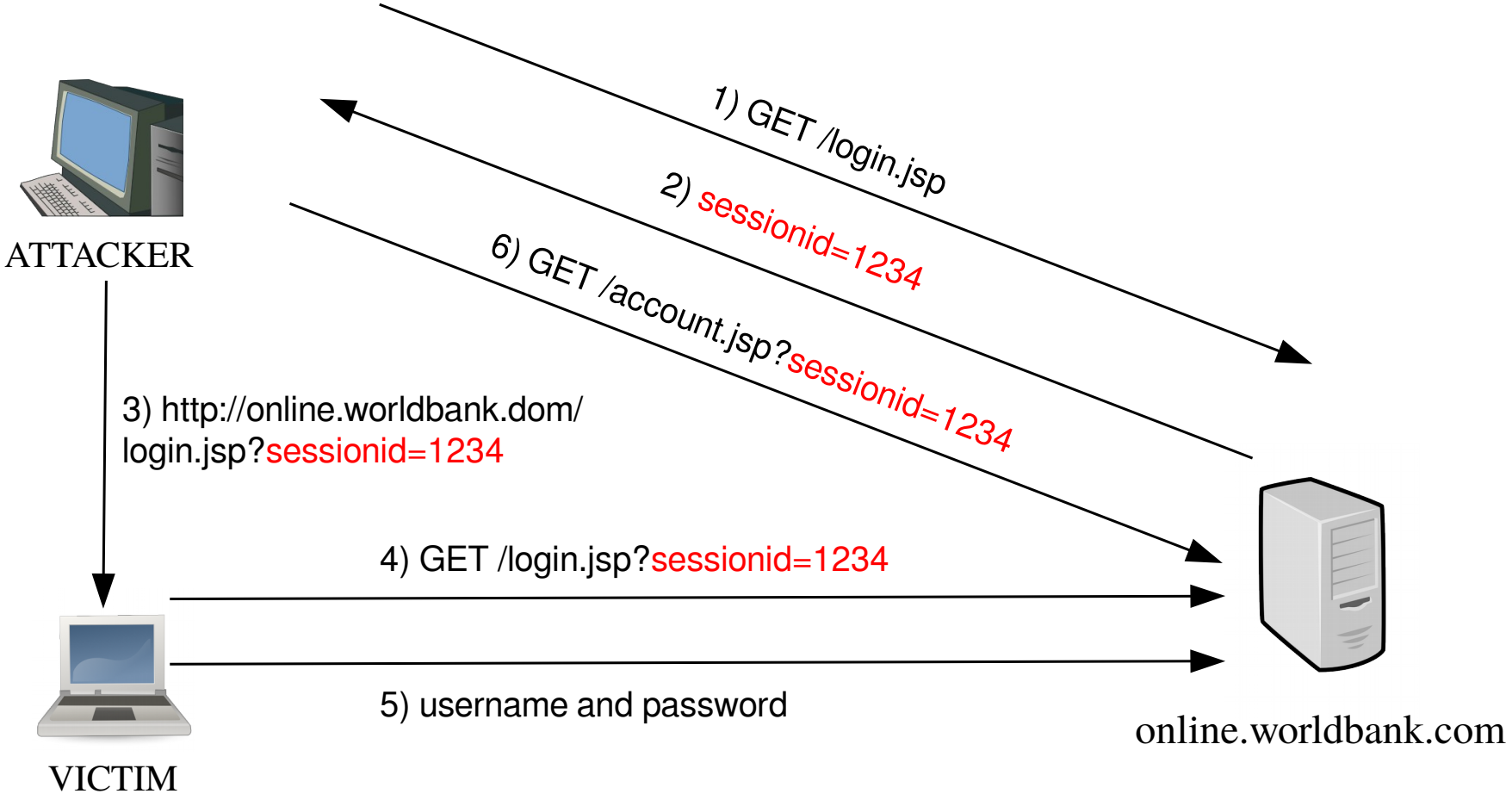
Another Scenario



Another Scenario



Another Scenario



Session Fixation vs Hijacking

- Timing!
- Session Fixation:
  - Victim's browser is attacked before she logs into the target server
- Session Hijacking:
  - Victim's browser is attacked after she logs into the target server

Session Fixation Attacks

- Attacker needs to be able to get a session ID from the server
  - attacker is a legitimate user of the service, OR
  - server assigns session ID before authentication
- The victim needs to be tricked into clicking on link (before session times out)
- Generally, a session fixation attack is a three-step process

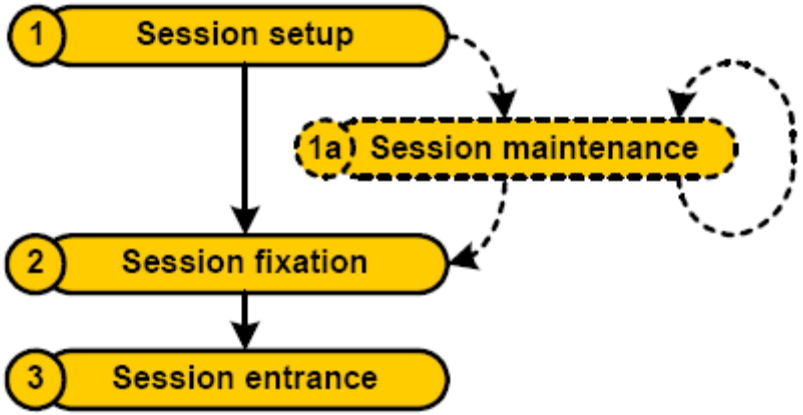
  1) Session setup

  2) Session fixation

  3) Session entrance

Session Fixation Attacks

- In session startup, attacker sets up a so-called “trap session”. Session needs to be kept “alive” by sending requests
- Next, attacker needs to introduce her session ID to the victim’s browser
- Attacker needs to wait until the victim logs in and then enter the victim’s session.



Step 1: Session Setup

- Session management mechanisms can be classified as: permissive and strict
- Against permissive systems: attacker simply picks a session ID
- Against strict systems: attacker sends a request to the server to obtain a session ID
- Permissive systems are easy to attack because the session ID picked by the attacker can be sent to the server at any time; requires no trap session maintenance

Step 2: Session Fixation

- Attacker can use several methods to transport the trap session to the victim's browser
- One way is to trick user into clicking a URL, for example:

  http://online.worldbank.dom/login.jsp?sessionid=1234
- Another possibility is to prepare a login page (e.g., phishing page) where ID is embedded as a hidden from field
- additional method for session fixation attack: cookies

Step 2: Session Fixation

- The attacker needs to install the trap session ID cookie on the victim's browser
- However, browser will only accept cookie assigned either to the issuing server or the issuing server's domain. attacker.com cannot set cookie for worldbank.dom
- Hence, the attacker can
  - exploit an XSS vulnerability in the website or browser to set a cookie
  - cross-subdomain cooking
    - evil.worldbank.com can set cookies for worldbank.com

Session Fixation Countermeasures

- Preventing session fixation is the responsibility of the web application, not the web server
  - Only the web application can implement effective protection. The web server (e.g., Tomcat) only needs to make sure that session IDs cannot be brute-forced or guessed
  - A common-denominator for session fixation attacks is that victim needs to login.
  - Use strict session management policy (server refuses IDs it did not generate)
  - If possible, web application should issue session IDs only after successful authentication
  - alternatively, regenerate the session ID after login

Session Fixation Countermeasures

- Session ID usage should be restricted
  - Bind to IP (network address): usual problems
  - Session ID should be bound to SSL client certificate for highly-critical applications (was the session ID established using certificate?)
  - Session destruction (timeout or logging out)
  - User must have option to log out (destroy current session and any other session)
  - use absolute session timeouts in order to prevent attacker from keeping trap session alive while waiting for victim to log in (can become inconvenient for the user)

Cross Site Request Forgery
CSRF / XSRF

Cross Site Request Forgery

- CSRF: A type of attack that lets attackers send arbitrary HTTP requests on behalf of a victim
- The damage caused by this attack can be severe
  - The attack is not too easy to understand and avoid, and it is likely that many web applications are vulnerable
- Typical scenario: User has established level of privilege with the site
  - Attacker uses this privilege to do “bad” things

Cross Site Request Forgery

- The site is the target of the attack.
- User is the victim and unknowing accomplice.
- The request comes from the victim, hence, it is difficult to identify a CSRF attack.
  - In fact, if you have not taken precautions, chances are very high that your application is vulnerable to CSRF
- Many web developers are still unaware of CSRF, so they take no precautions against it
  - Many widely deployed web applications are vulnerable or have been vulnerable to CSRF

CSRF Example

- Suppose there exists a simple PHP Web application that can be used by (logged in) administraror for creating new users. Here is the form:

```
<form action="create.php" method="POST">
<p>
Username: <input type="text" name="username">
Password: <input type="text" name="password">
<input type="submit" value="Create"/>
</p>
</form>
```

CSRF Example

- … here is the simple PHP application hosted at http://www.victim.com/create.php

```
<?php

session_start();

If (isset($_REQUEST[‘username’] &&
isset($_REQUEST[‘password’]))

{

create_new_user_dude($_REQUEST[‘username’],
$_REQUEST[‘password’]);

}

?>
```

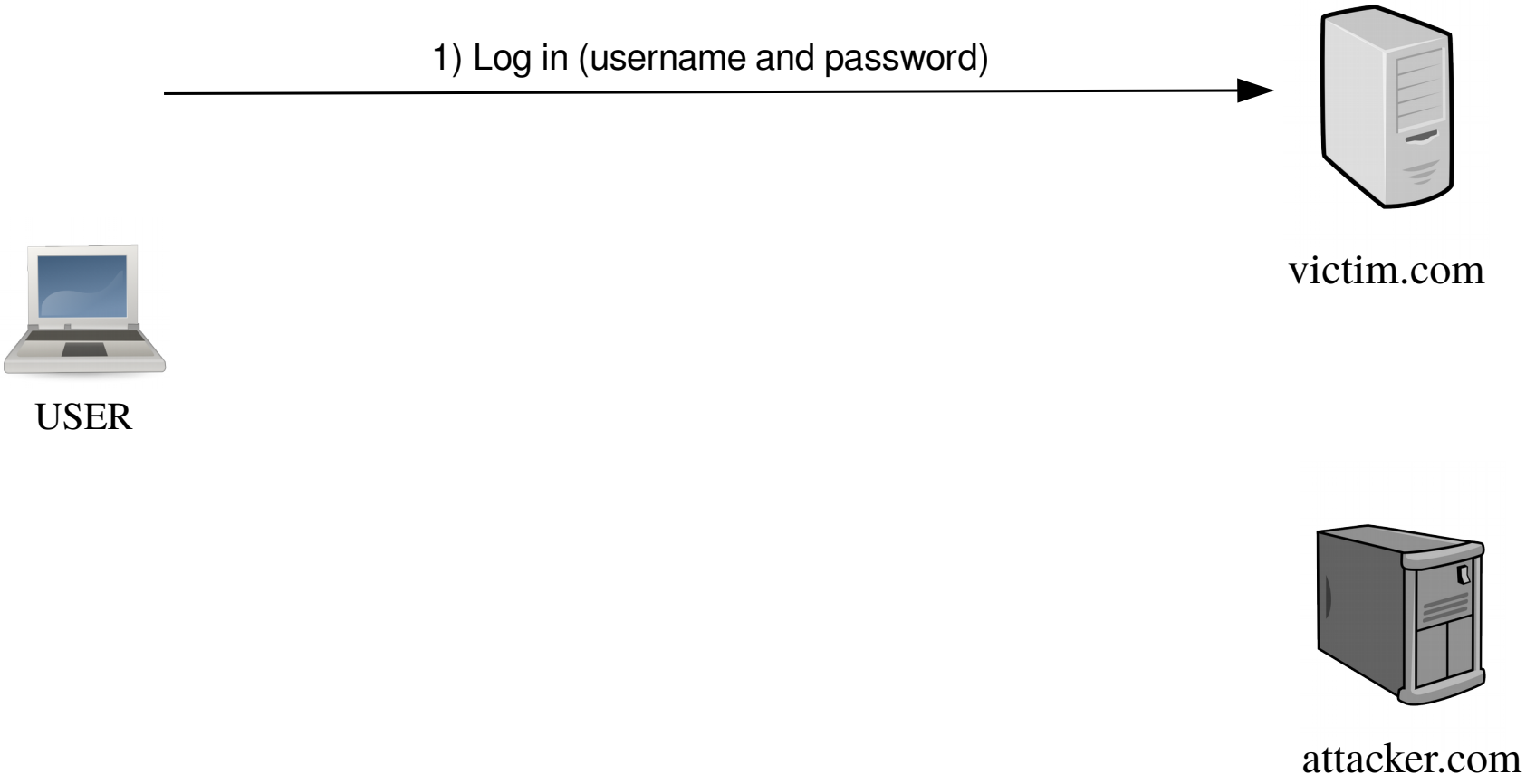
CSRF Example

- What is the problem with this application?
  - Suppose an attacker manages to trick the authenticated user to visit a web page of which she has control
    - Note that visiting is enough!!
  - The “owned” web page has an embedded link such as:

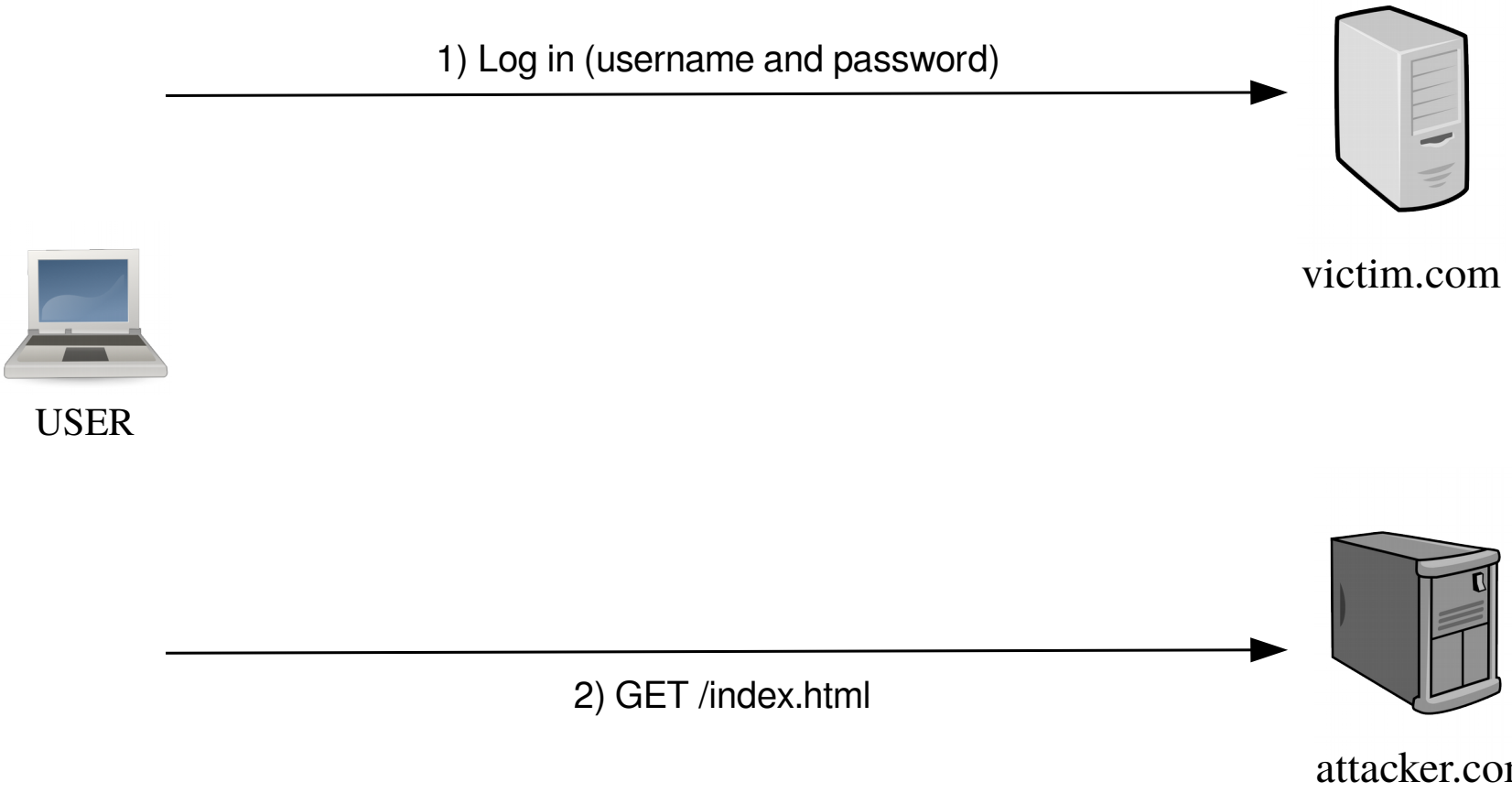
<img src=“http://www.victim.com/create.php?username=badguy&password=nopasswd”>

- The browser tries to download the image from victim.com
  - browser helpfully appends the user's cookies for victim.com

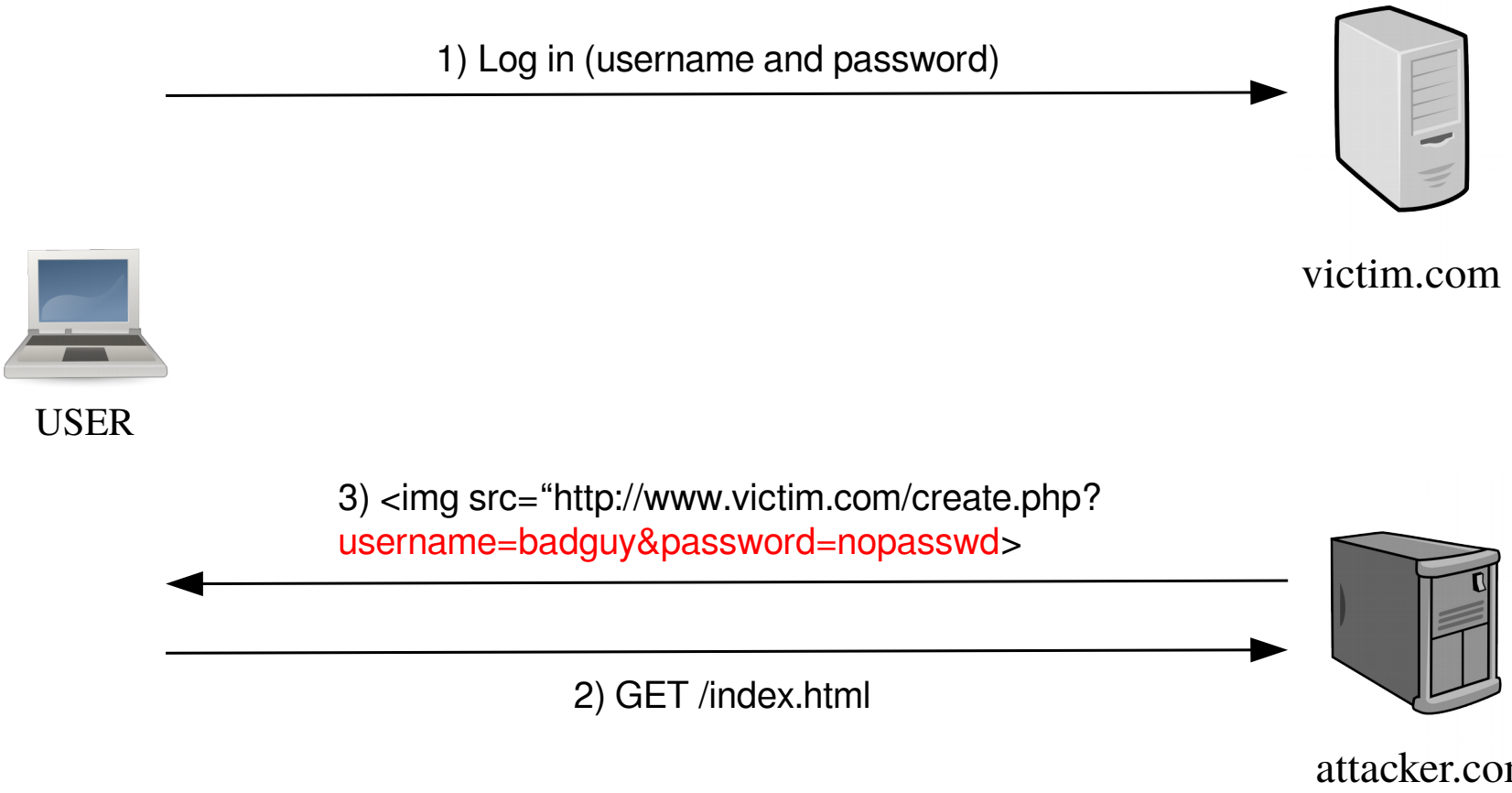
CSRF Example



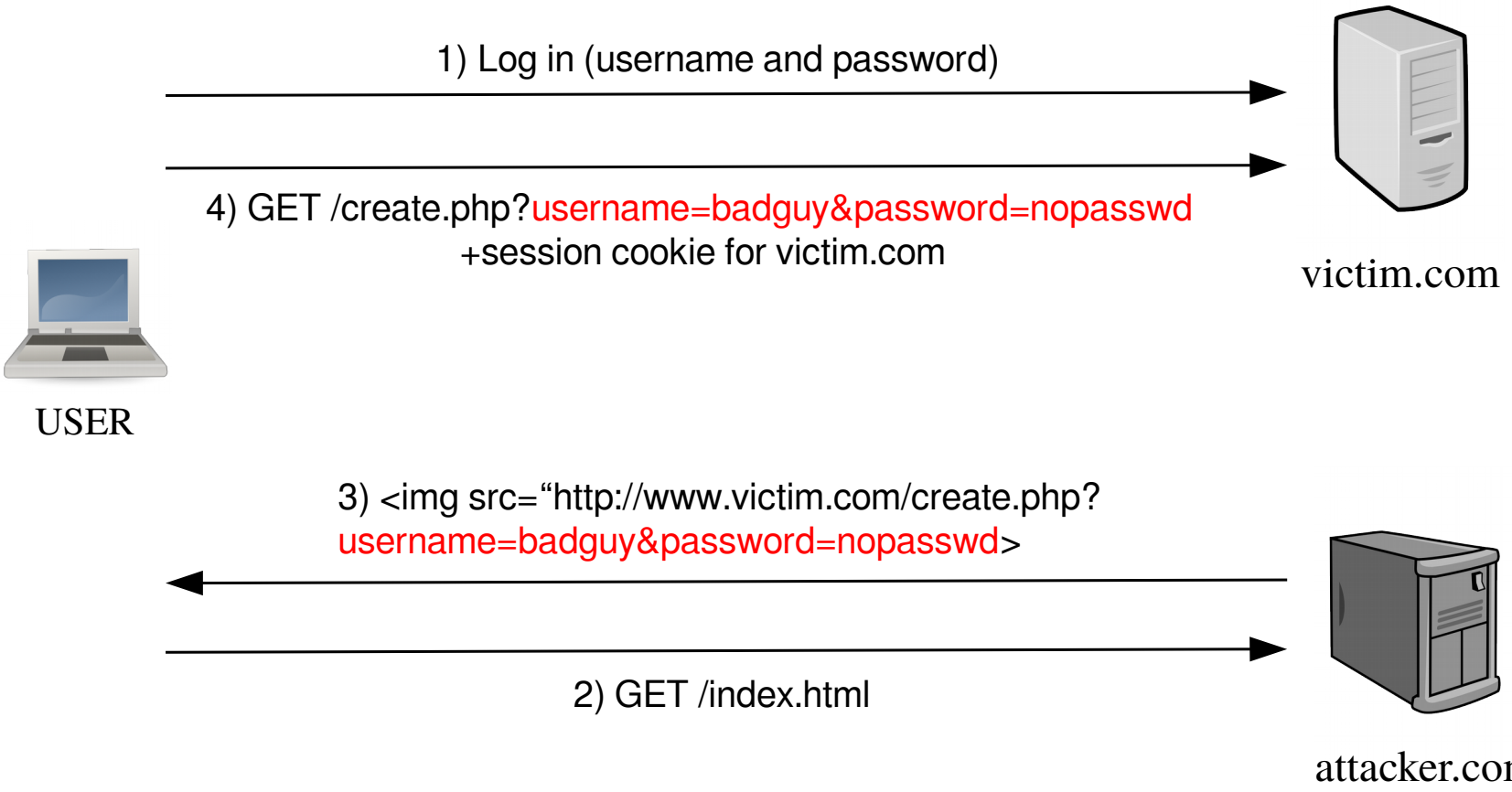
CSRF Example



CSRF Example



CSRF Example



A First Fix

- Once the user visits the page, the URL is fetched and a new user is created. Hence, the web application is compromised
- Why did this error happen? The application used $_REQUEST instead of $_POST
  - It could not distinguish between data sent in the URL and data provided in the form
  - $_REQUEST and allowing GET increases your risk
- General principle of web application design:
  - GET requests should have no side effects

CSRF over POST

- **Using POST requests is not enough to prevent CSRF**

- Attacker can fool user into clicking on submit button of a form on attacker.com
  - form points to victim.com
  - triggers POST request with hidden fields!

- JavaScript can be used to automatically submit a form
  - no interaction needed!

CSRF Countermeasures

- Force the usage of your own forms.
  - Try to identify if the request is coming from your own form

- For example, you could generate a token as part of the form and validate this token upon reception
  - E.g., using unique IDs, MD5 hashes, etc.
  - The token has to be bound to the user session
  - **Cannot be stored in a cookie**
  - You could limit the validity of the token time (e.g., 3 minutes)
  - Attacker cannot steal the token because of SOP

CSRF Vulnerabilities

- Many high profile sites have been vulnerable
- Gmail
  - http://betterexplained.com/articles/gmail-contacts-flaw-overview-and-suggestions/
  - allowed to steal user's contact list
- netflix
  - http://jeremiahgrossman.blogspot.com/2006/10/more-on-netflixs-csrf-advisory.html
  - allowed to change name and address, and order movies
- skype
  - http://www.securescience.net/xss/skype/skype.html
  - allowed to "steal" the user's skype number (impersonate user, receive his calls, use his credit)

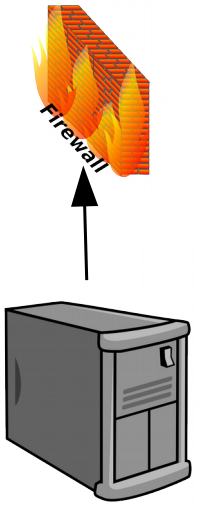
CSRF against home routers

- Home DSL/Cable routers have often been vulnerable
- Typically have weak authentication
  - no password
  - username: admin, password: admin
  - same password for all routers from a provider
    - just google for "$provider $device_name password"
  - most users don't change it
- Protected by firewall
  - can only log in from inside home network
- Can use CSRF to send requests to the router from victim's PC inside the network

CSRF against home routers

DSL router
192.168.0.1

Home User
192.168.0.101



attacker.cm

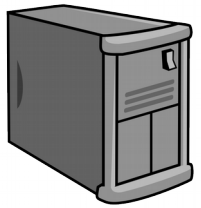
CSRF against home routers

DSL router
192.168.0.1

Home User
192.168.0.101

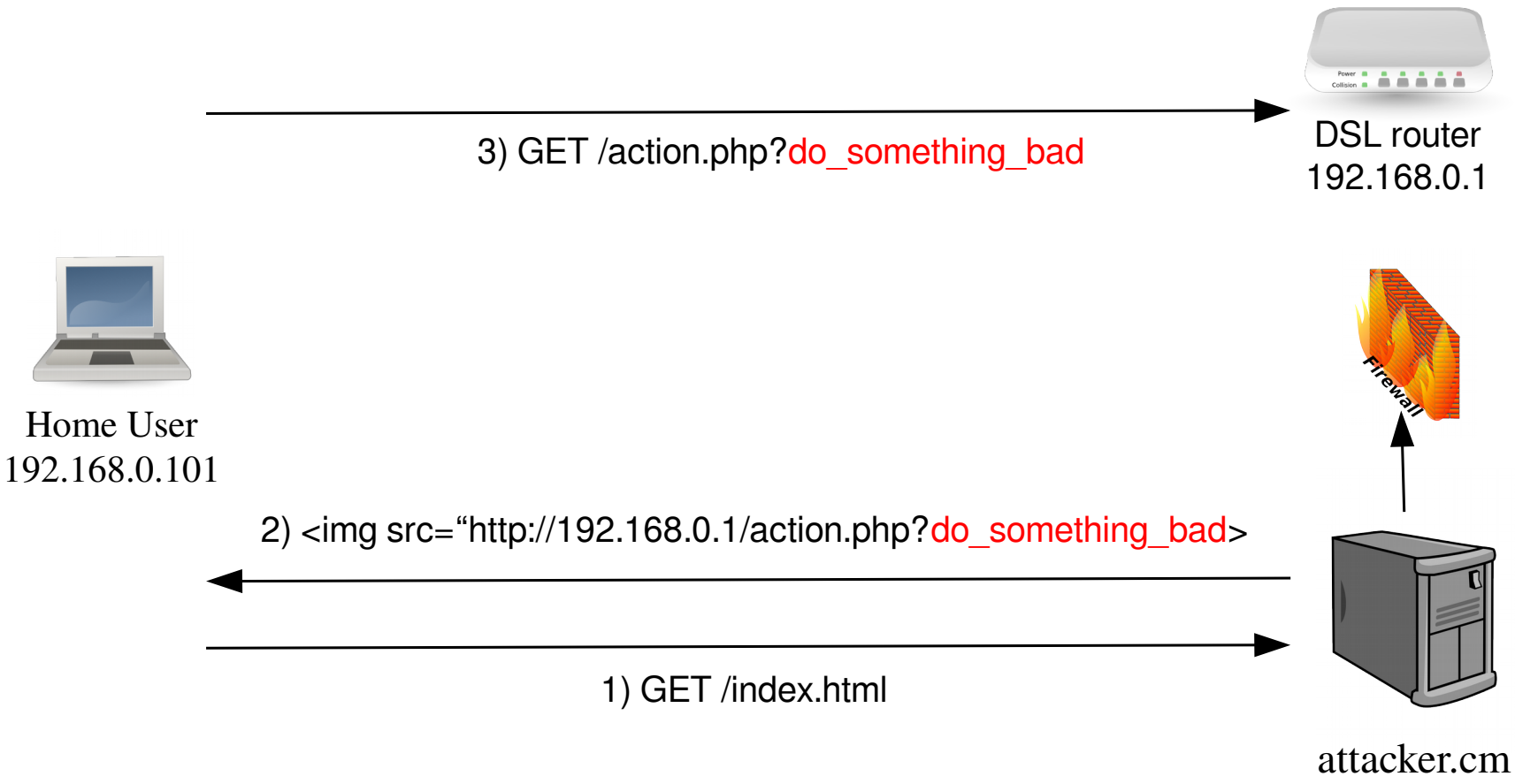
2) <img src="http://192.168.0.1/action.php?do_something_bad>

1) GET /index.html



attacker.cm

CSRF against home routers



CSRF against home routers

- What can the attacker do?
  - example: CSRF in home routers from mexican ISP
  - no password was set by default
  - http://www.securityfocus.com/archive/1/archive/1/476595/100/0/threaded
- Add names to the DNS (216.163.137.3 www.prueba.hkm):
  - http://192.168.1.254/xslt?PAGE=J38_SET&THISPAGE=J38&NEXTPAGE=J38_SET&NAME=www.prueba.hkm&ADDR=216.163.137.3
- Disable Wireless Authentication
  - http://192.168.1.254/xslt?PAGE=C05_POST&THISPAGE=C05&NEXTPAGE=C05_POST&NAME=encrypt_enabled&VALUE=0
- Disable firewall, set new password,...

MITM Attacks against HTTPS

Stealing HTTPS cookies

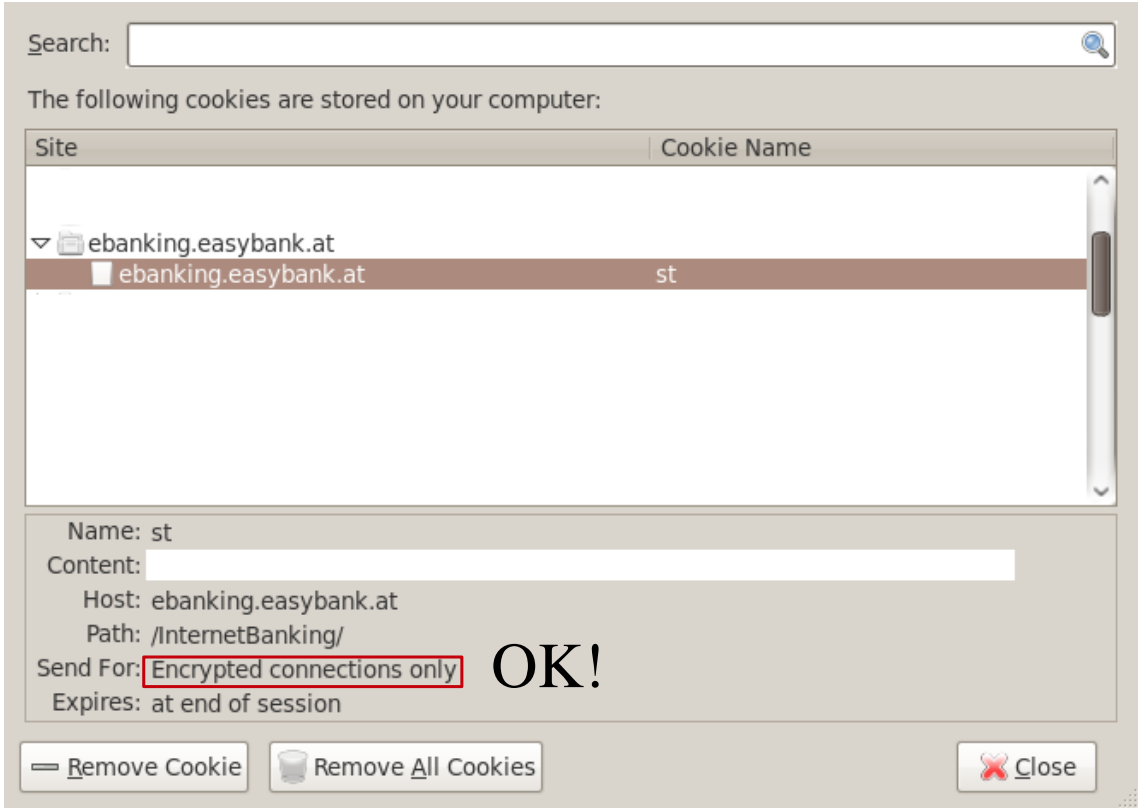
- Man-in-the-middle attack against HTTPS
  - encryption should protect from MITM!
- Cookie management policy is not consistent with Same Origin Policy
  - SOP: in addition to domain, protocol must be the same: http≠https
  - cookies: depends on a secure flag in the cookie
- if secure flag is not set, a cookie set inside an https session can be sent over an http session
  - MITM can use this to steal a session ID that was set inside an HTTPS connection!

Stealing HTTPS cookies

- Attack steps:
  - user logs in to https://mybank.com
  - server sets session ID for mybank.com in cookie
    - encrypted in https
  - user issues an unencrypted HTTP request for something else (http://news.com)
  - attacker (MITM) replies with redirect to http://mybank.com
  - browser follows redirect & sends out cookie with session ID
  - attacker has access to user's session!

Stealing HTTPS cookies

- Easy to check for using your browser
- Check your online banking website, and send them an email if they are vulnerable
- When I first checked the 2 banking websites I use, both were vulnerable



Cookie Attributes

- ```
  HTTP/1.0 200 OK
  Content-type: text/html
  Set-Cookie: name=value
  ```

- Limit to HTTPS:
  ```
  Set-Cookie: name=value; Secure
  ```

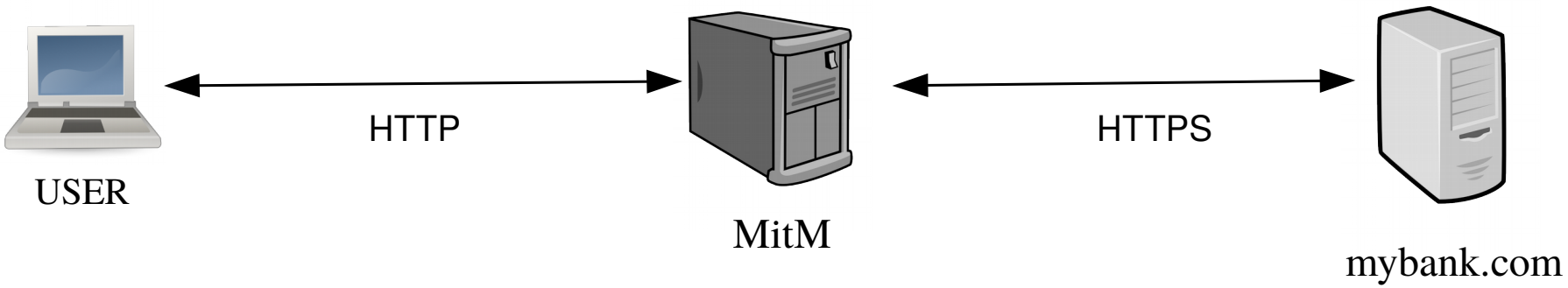
- Limit to real browser requests (not JS, XML-RPC, API...)
  ```
  Set-Cookie: name=value; HttpOnly
  ```
  ```

SSL Stripping

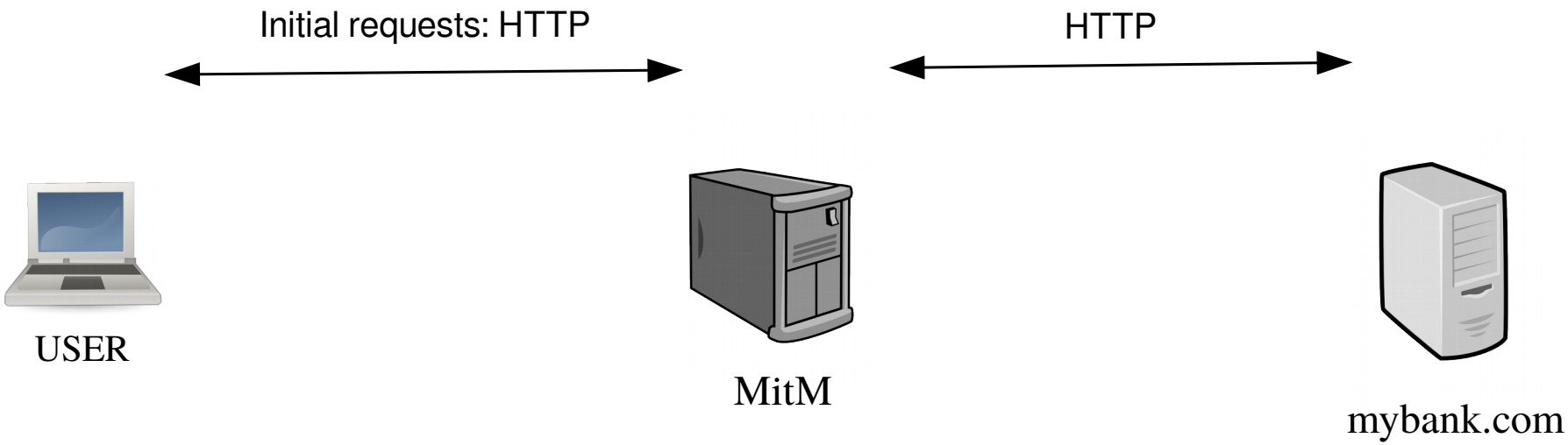
- A more powerful MitM attack against HTTPS
- Old problem:
  - https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf
- Users do not type "https://" in their browser (or http://)
- Most of the time, users reach https through http
  - this first step is unprotected

SSL Stripping

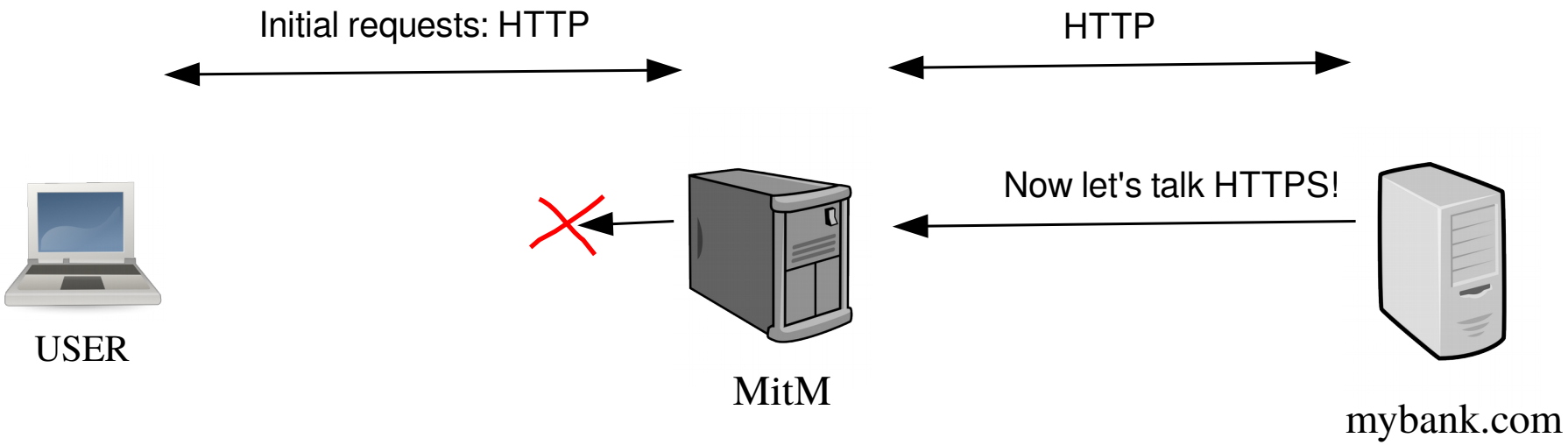
- The basic idea:
  - attacker speaks HTTPS with the server
  - attacker speaks HTTP with the user



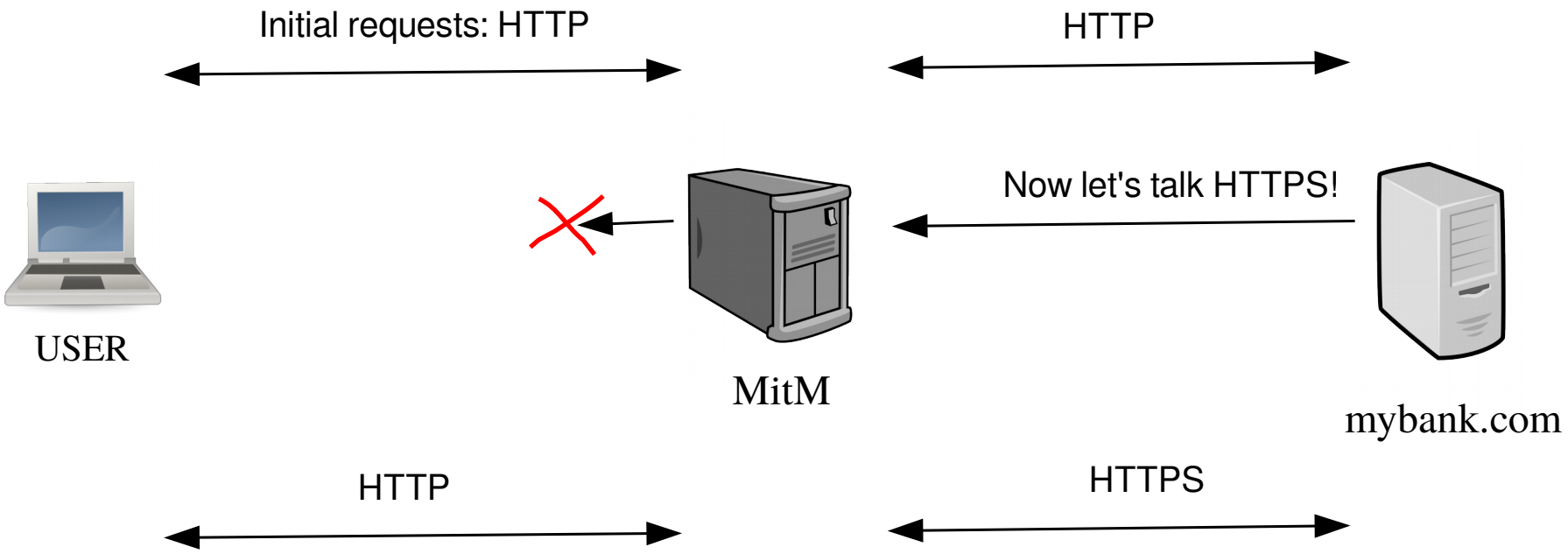
SSL Stripping



SSL Stripping



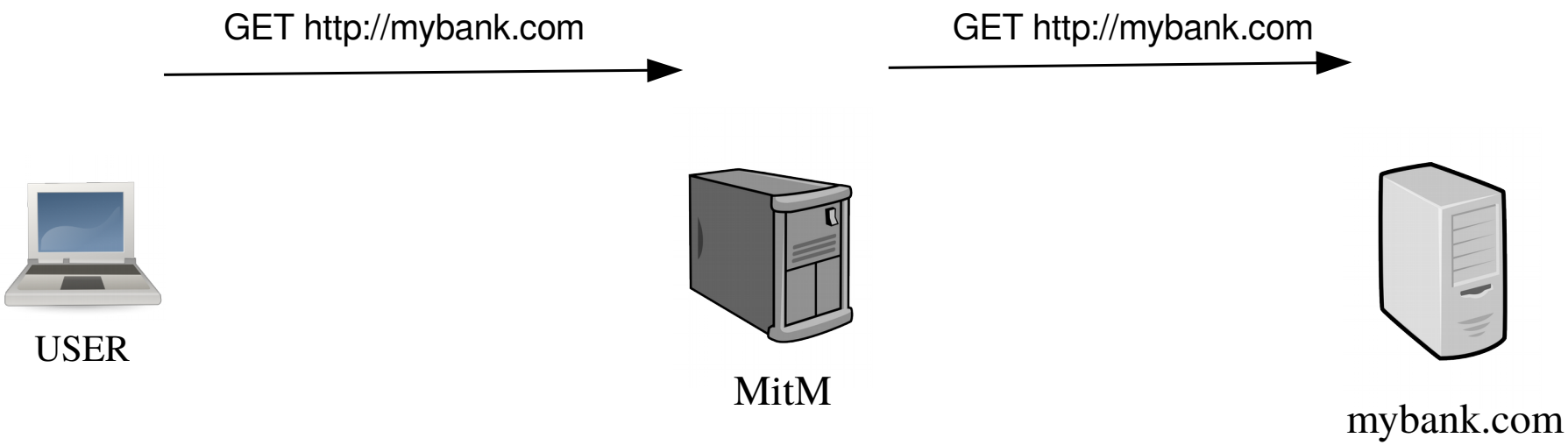
SSL Stripping



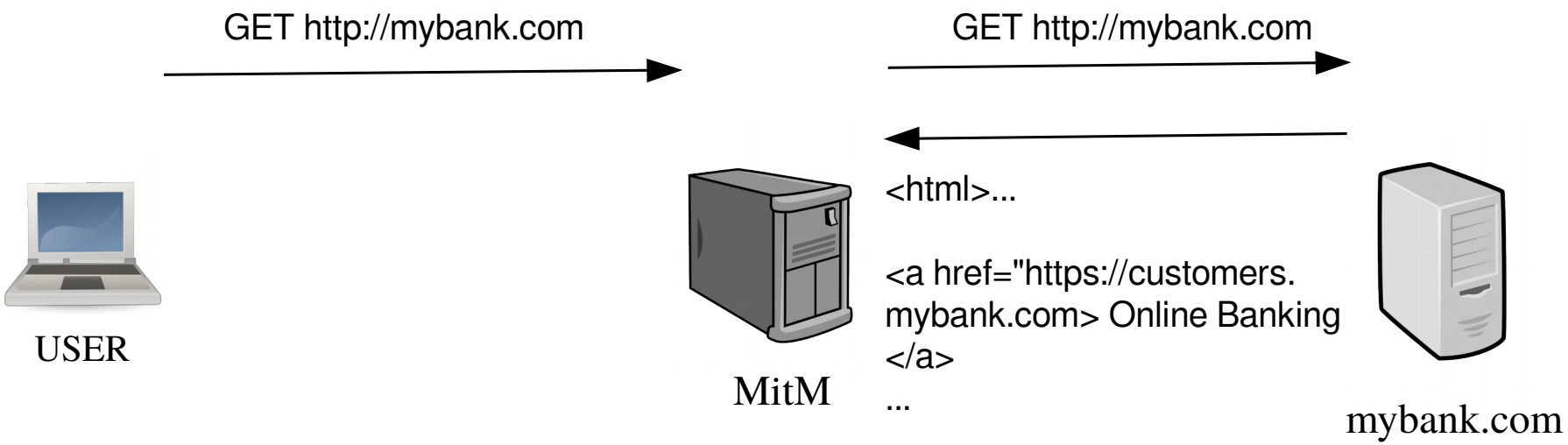
SSL Stripping

- User types www.mybank.com in URL bar
- Before logging in, user would normally be directed to the bank's HTTPS website
  - HTTP 3xx redirect, OR
  - an https link
- MitM attacker can modify the data sent from the server
  - change https:// to http:// in server replies!

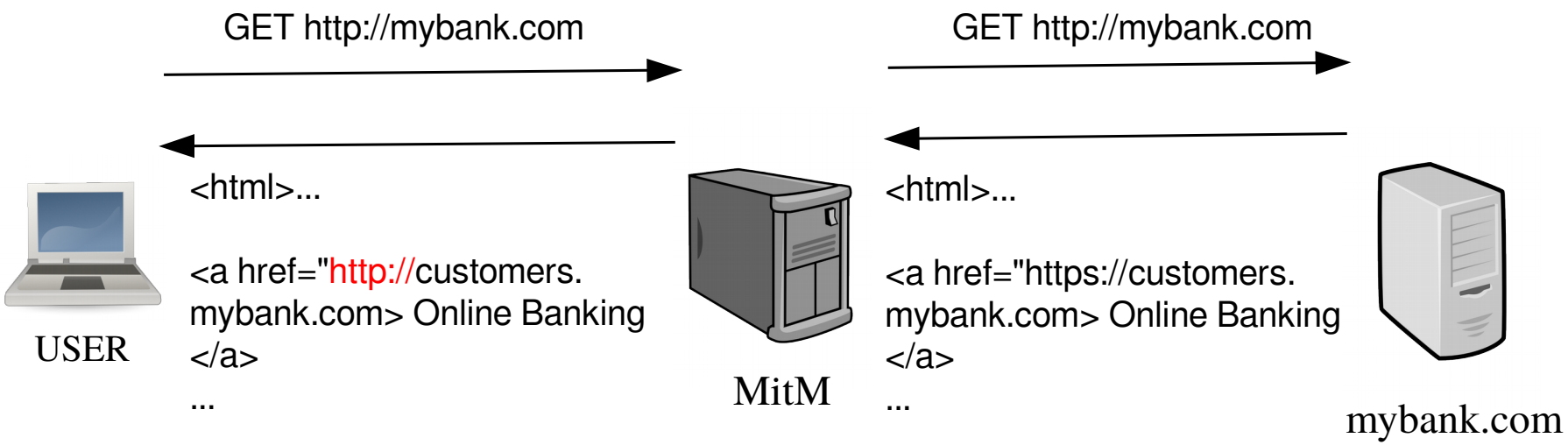
SSL Stripping



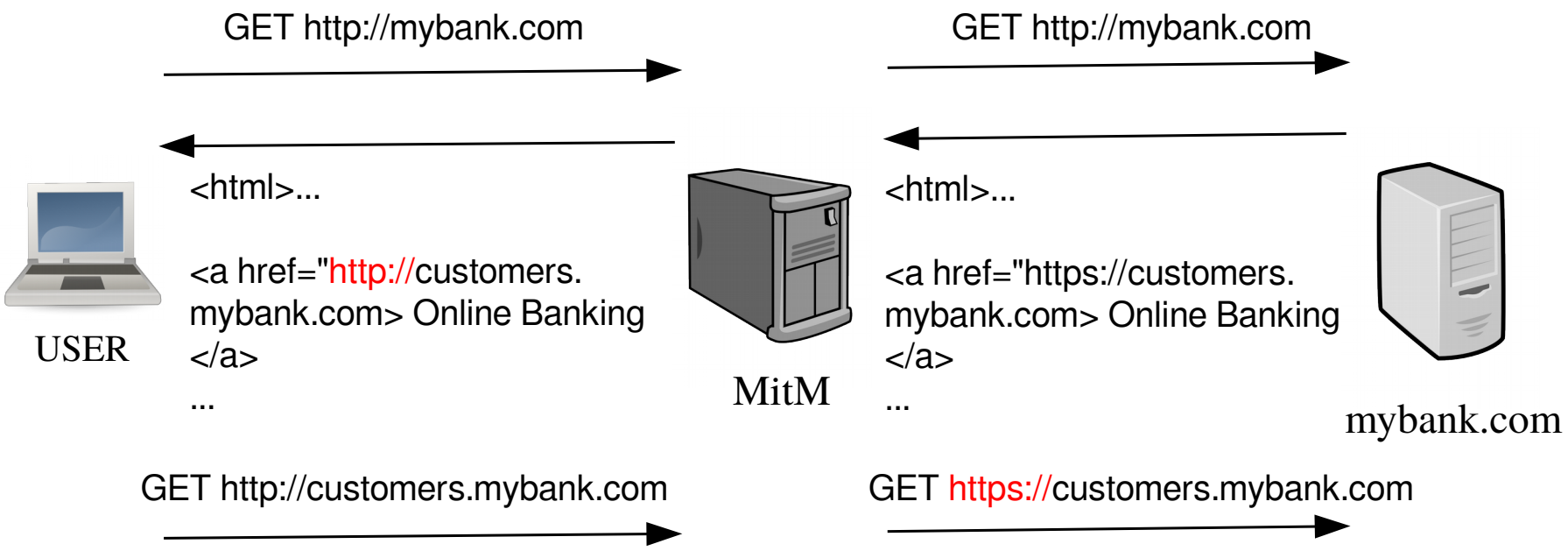
SSL Stripping



SSL Stripping



SSL Stripping

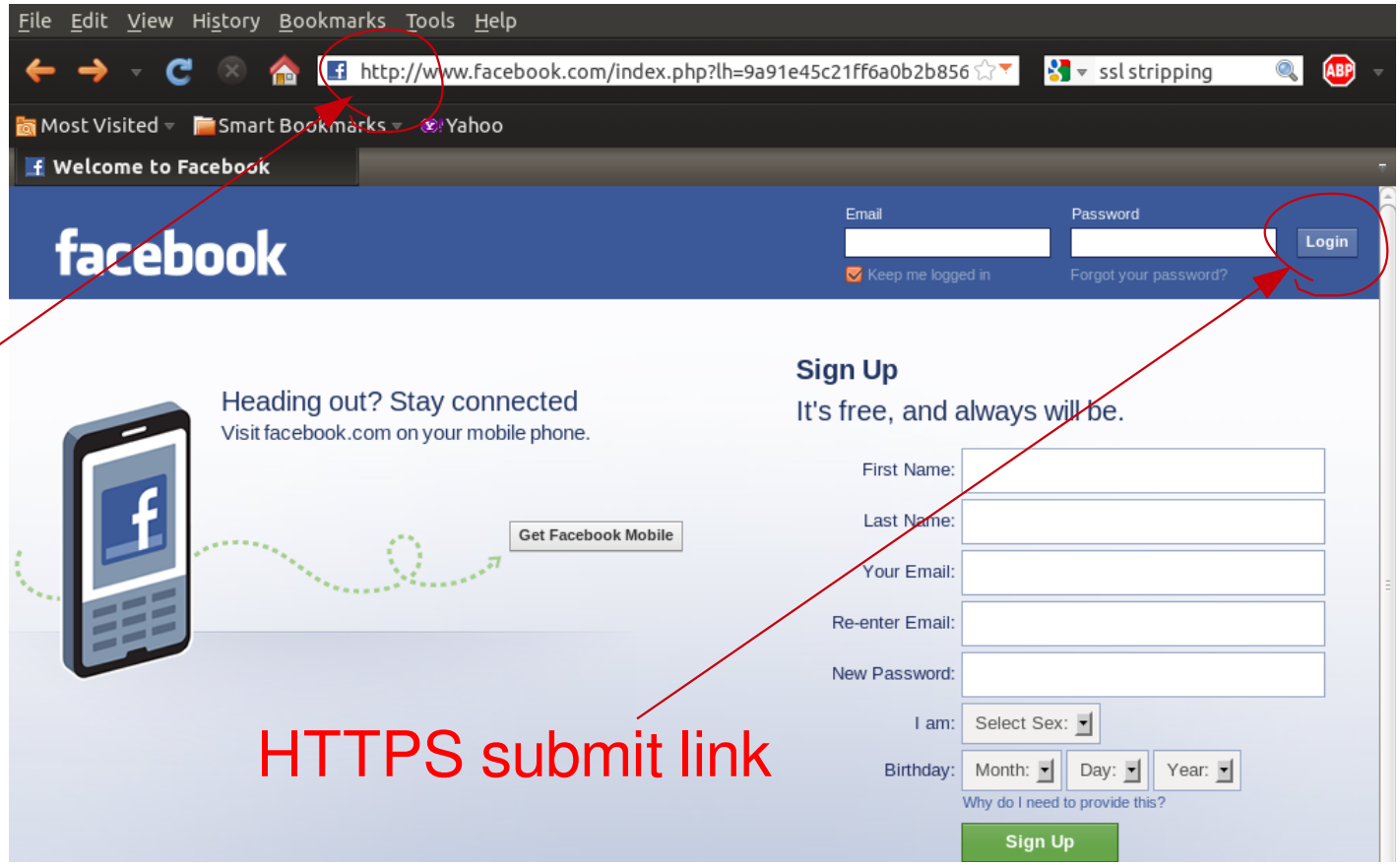


Fooling the user

- User can see http:// in the address bar
  - some users may check for this
  - lock favicon may help fool them 🔒
- Many sites use https only for login
  - to protect username and password
  - does not protect from session hijacking
- SSL stripping in these cases is undetectable
  - unless user looks at html page source

Fooling the user

user never sees https

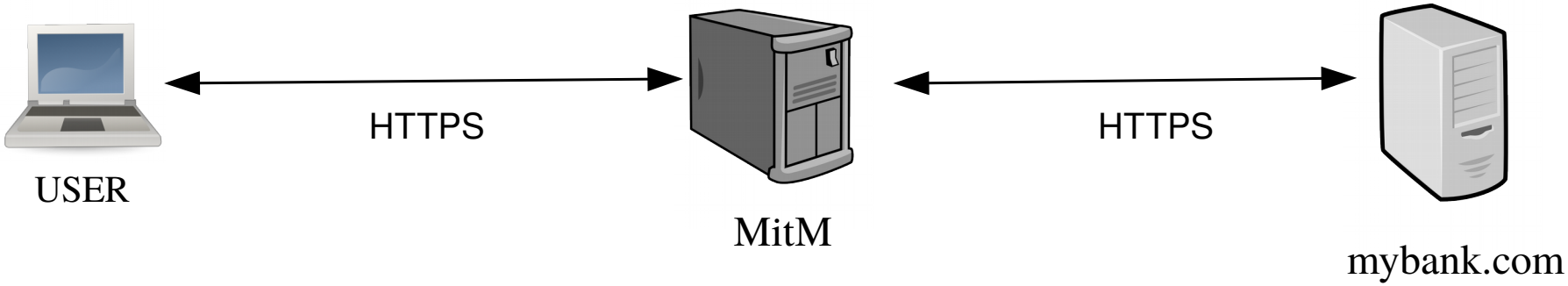


Implementing SSL Stripping

- Modify server replies to replace https links with http
  - in HTTP 3xx redirect replies
  - in HTML page
    - harder if page is dynamically generated with javascript
- Modify client requests
  - remove secure flag on cookies sent by server
    - otherwise user's browser will not send them over HTTP!
- Simplify the task:
  - avoid compression and chunked encoding
  - talk HTTP/1.0 with server
  - Accept-Encoding: identity

SSL Stripping Variant

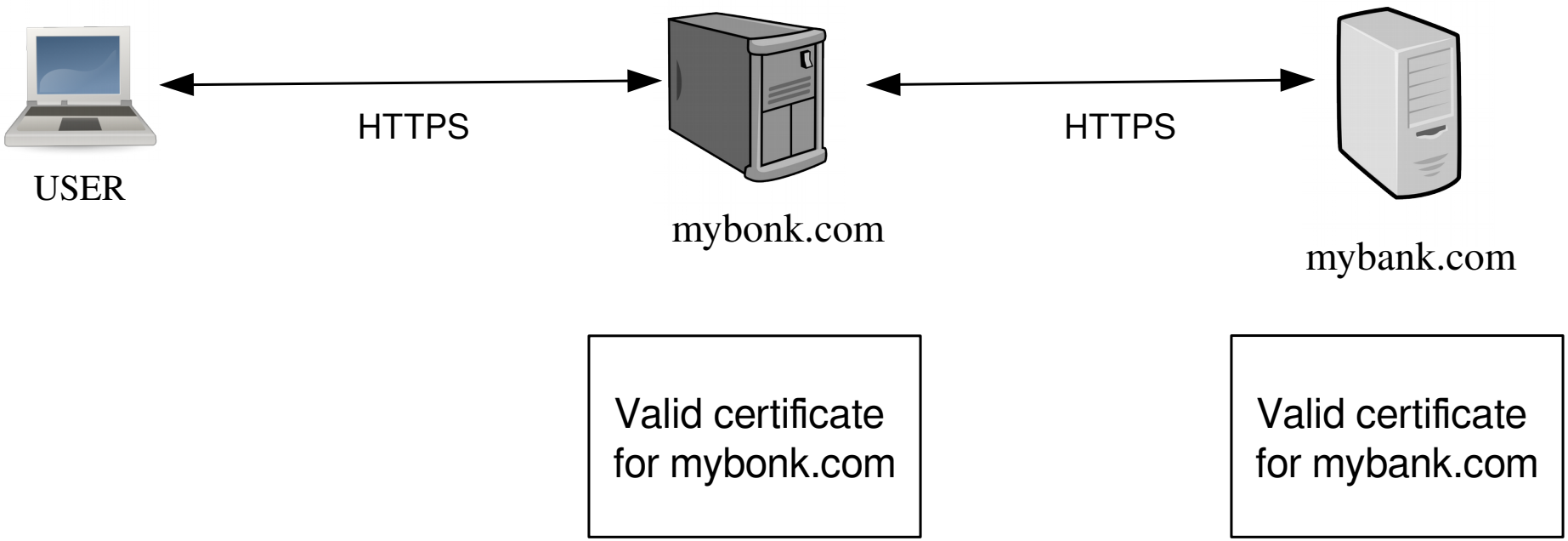
- Instead of stripping SSL, we can have 2 separate HTTPS conversations with user and destination site



SSL Stripping Variant

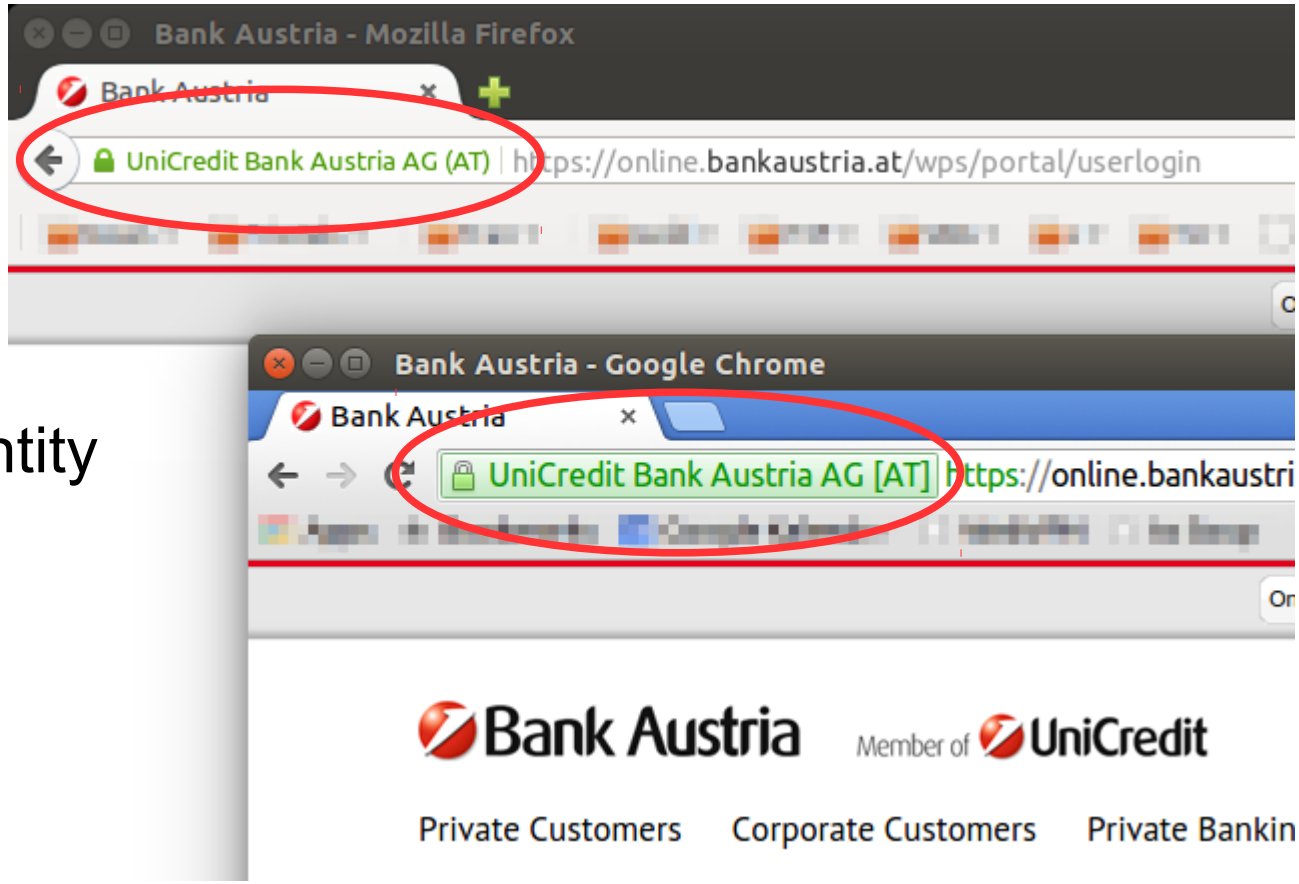
- Option 1: use fake, self-signed certificate for mybank.com
  - browser warning may scare user away!
- Option 2: redirect user to https://evil.com/
  - attacker has a valid certificate for evil.com
  - user may notice wrong domain in address bar!
- Option 3: like 2, but use a domain name that looks similar to mybank.com
  - with internationalized domain names, this is a very powerful attack

SSL Stripping Variant



Extended Validation (EV) Certificates

- Name of Entity shown
- Green Bar



“Cryptocalypse”

- Typical browser has 100+ Root certificates (CA) installed
  - Each CA can delegate to sub-CA (resellers)
  - Cost pressure reduces level of customer verification and security of infrastructure
  - Any of theses sub-CAs can sign certificates for any other domain. (e.g. finanzonline.gv.at)
  - Also used by Deep-Packet-Inspection (DPI)
- Countermeasure
  - CA or certificate pinning
    - Software additionally checks, who issued the certificate

HTTP Strict Transport Security (HSTS)

- `Strict-Transport-Security: max-age=31536000; includeSubDomains;`
- Browser caches HTTPS capability
- HSTS headers over HTTP are ignored

- HTTPS Everywhere
  - Browser plugin that automatically redirects to https://
  - Whitelist based

- Related: HTTP Public Key Pinning (HPKP)
  - Allows Server to pin key/certificate authority

Perfect Forward Secrecy

- Scenario
  - Some attacker sniffs encrypted traffic
  - Obtains private keys afterwards (years later?)
  - Decrypts traffic

- Perfect Forward Secrecy
  - Use special step in initial session setup for key exchange
    - Short term session key is not compromised, even when long term keys are compromised in the future.
  - Optional in Ipsec, SSL/TLS since SSLv3, [Browser comp.!!!]
  - Computational Overhead 15-30%
  - http://www.bettercrypto.org/

HTTP Response Splitting

HTTP Response Splitting

- Fairly interesting web application vulnerability
  - It can be used for XSS, cross user defacement, web cache poisoning, hijacking pages, and browser cache poisoning

- We will discuss a single possible scenario for this attack
  - for more info read Amit Klein's whitepaper:
  - “Divide and Conquer”:HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics

HTTP Response Splitting

- Web cache poisoning
  - deface the version of a web site that is stored in a cache. All users sharing that cache will see the defaced version
  - Attacker fools the proxy server into caching a page that she has control over. The mapping in the cache is messed up
  - Users that would like to access the legitimate web page see the attacker's page
  - Typically, attacker is able to steal cookies, session data, and forge fake login

HTTP Response Splitting

- Web application can be vulnerable if it inserts untrusted content into HTTP response headers
  - web application does not filter \r\n
  - attacker inserts \r\n, followed by more http headers and defaced page content
  - Example of vulnerable application: redirect script

```
<?php
header ("Location: " . $_GET['page']);
?>
```

http://seclab.edu/redirect.php?page=http://dude.com

HTTP Response Splitting

- 1) Request from client 2) Answer from server

```
GET /redirect.php?page=http://dude.com HTTP 1.1\r\n
…
Connection: keep-alive\r\n
\r\n
```

```
HTTP 1.1 302 Moved Temporarily\r\n
Location: http://dude.com
\r\n
<html><head><title>302 Moved Temporarily</title></head>
```

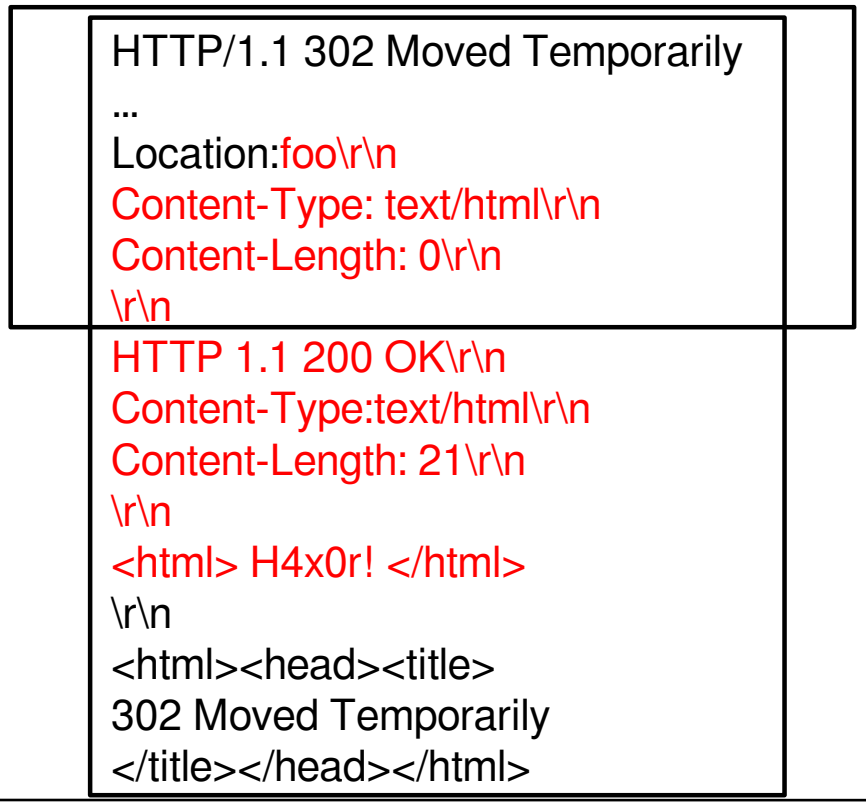
HTTP Response Splitting

- How does the attack work?
  - We insert extra \r\n and inject HTTP headers:
  - http://seclab.edu/redirect.php?page=foo%0d0a...
  - without the URL encoding, here is the parameter:

```
foo\r\n
Content-Type: text/html\r\n
Content-Length: 0\r\n
\r\n
HTTP 1.1 200 OK\r\n
Content-Type:text/html\r\n
Content-Length: \r\n
\r\n
<html> H4x0r! </html>
```

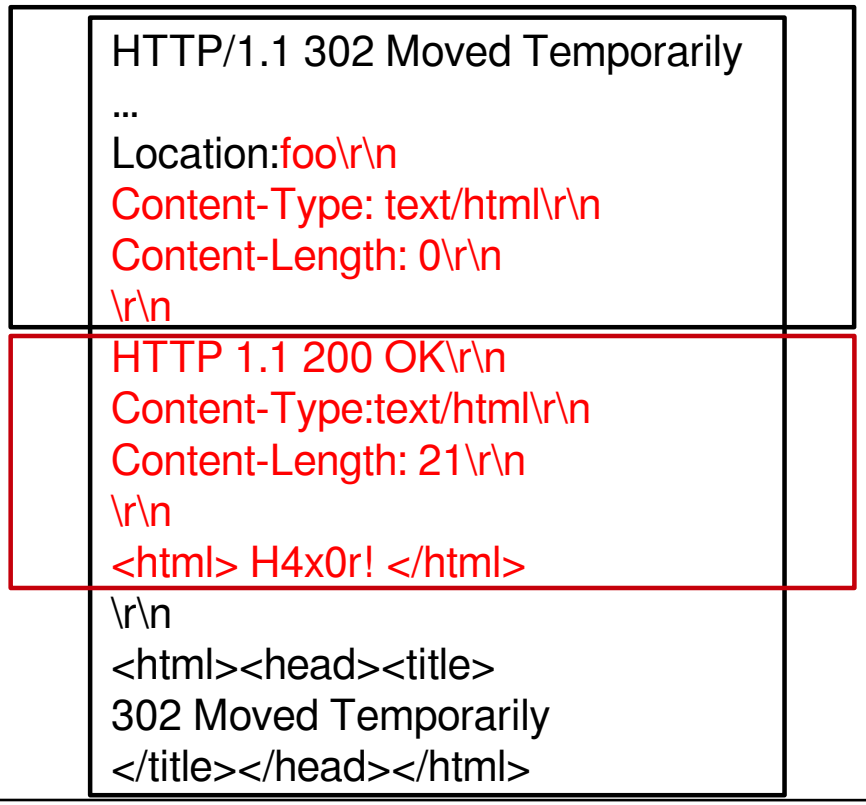
HTTP Response Splitting

- This is what the whole reply now looks like



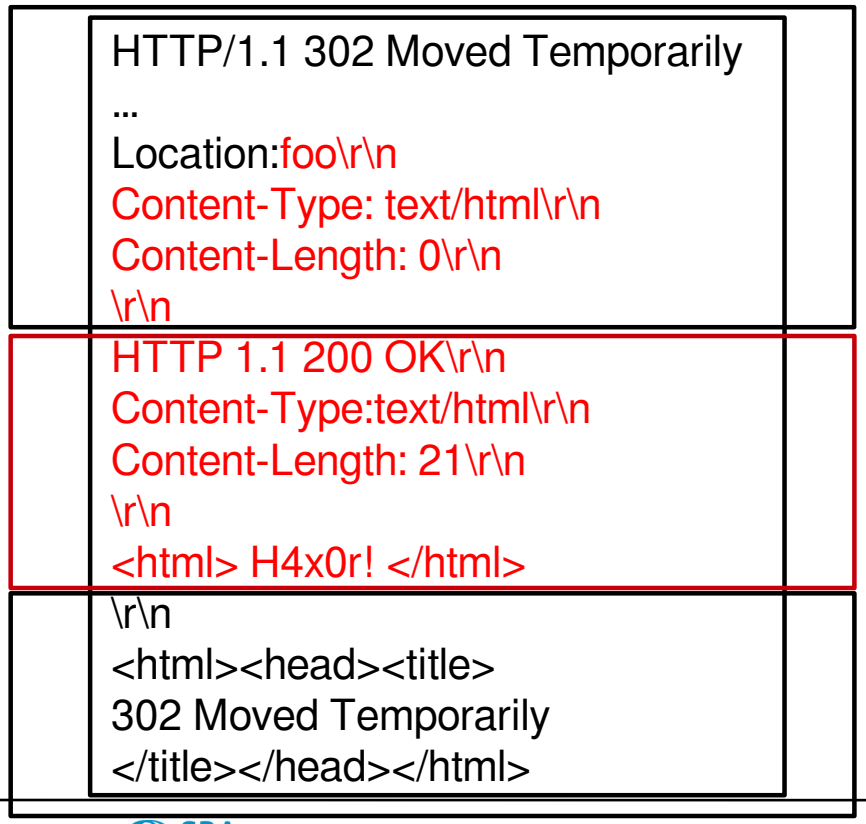
HTTP Response Splitting

- This is what the whole reply now looks like



HTTP Response Splitting

- This is what the whole reply now looks like



HTTP Response Splitting

Client Sends 2 requests:

Caching Proxy associates them with 2 replies:

GET /redirect.php?
page=foo%0d0a...

HTTP/1.1 302
Moved Temporarily

GET /target_page.html

...<html>H4x0r!</html>...

Live Demo

Live Demo

- Burp Suite Community Edition
  - https://portswigger.net/burp/communitydownload
- OWASP Juice Shop Project
  - https://www.owasp.org/index.php/OWASP_Juice_Shop_Project
- OWASP Juice Shop Project: Challenges and Solutions
  - https://bkimminich.gitbooks.io/pwning-owasp-juice-shop/content/part2/

Conclusions

- We covered a range of advanced topics in web security
  - session fixation
  - cross-site request forgery
  - https cookie theft
  - SSL problems
  - “Cryptocalypse”
  - browser history stealing