

Internet Security 2

(aka Advanced InetSec)

Web Security 3

Markus Huber
SBA Research

inetsec@seclab.tuwien.ac.at

News from the Lab

*Int. Secure Systems Lab
Technical University Vienna*

- Challenge 4 / Simple Worm for another week
 - More than 10 people have already completed it
 - And the winner is: BERNHARD "Mind Mole" URBAN
- Next lecture: 15th of December on DRM

Overview

*Int. Secure Systems Lab
Technical University Vienna*

- More on session attacks
 - continuation of last year's lecture (session hijacking)
- Cross Site Request Forgery
- Man-in-the-Middle attacks against HTTPS
 - https cookie stealing
 - SSL Stripping
- Browser history stealing (historical)
- Attacks exploiting the caching infrastructure

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

*Int. Secure Systems Lab
Technical University Vienna*

- 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
- the first three attacks can be grouped into “Session Hijacking“ attacks
 - we discussed them in inetsec1

NEW!

Session Hijacking Attacks

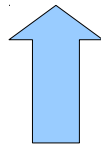
*Int. Secure Systems Lab
Technical University Vienna*

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

Session Hijacking Attacks

*Int. Secure Systems Lab
Technical University Vienna*

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

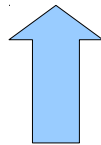


- Prevention:
 - use SSL for each request/response that transports a session ID
 - not only for login!

Session Hijacking Attacks

*Int. Secure Systems Lab
Technical University Vienna*

- 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!



- 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)

Int. Secure Systems Lab
Technical University Vienna

- 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

*Int. Secure Systems Lab
Technical University Vienna*



ATTACKER



VICTIM



online.worldbank.com

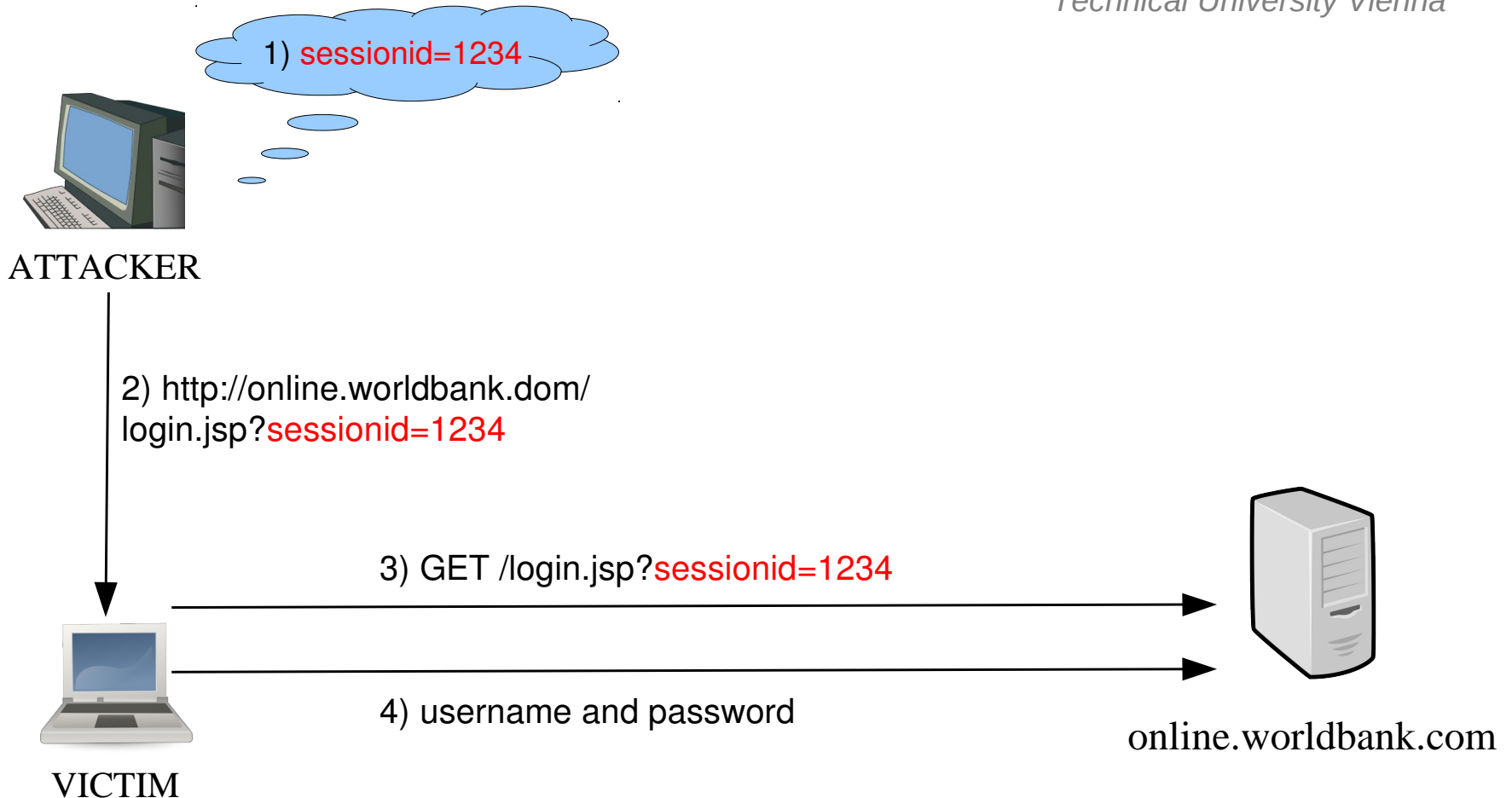
Simple Scenario

Int. Secure Systems Lab
Technical University Vienna



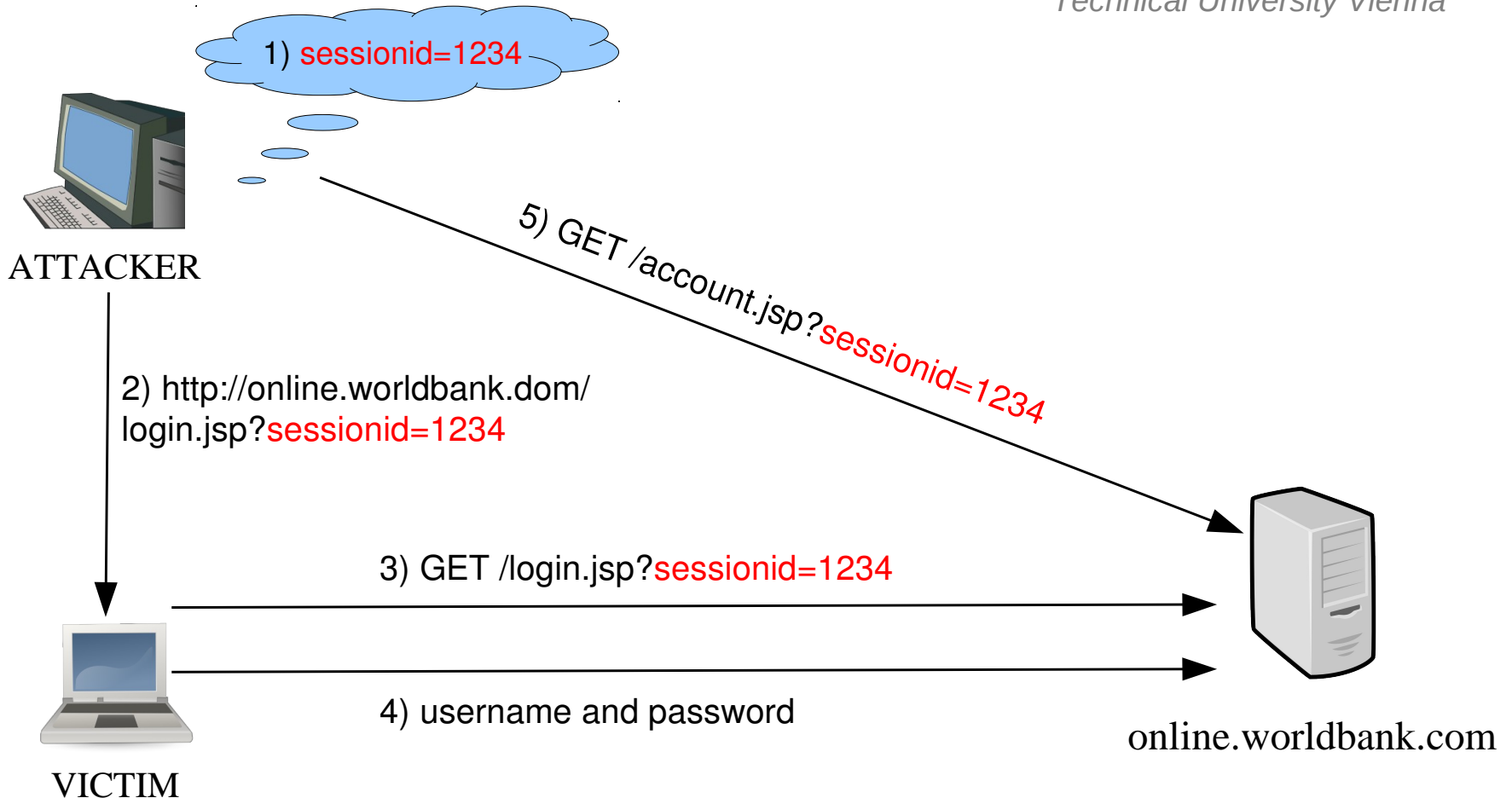
Simple Scenario

Int. Secure Systems Lab
Technical University Vienna



Simple Scenario

Int. Secure Systems Lab
Technical University Vienna



Another Scenario (strict session management)

Int. Secure Systems Lab
Technical University Vienna

- 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

Int. Secure Systems Lab
Technical University Vienna



ATTACKER

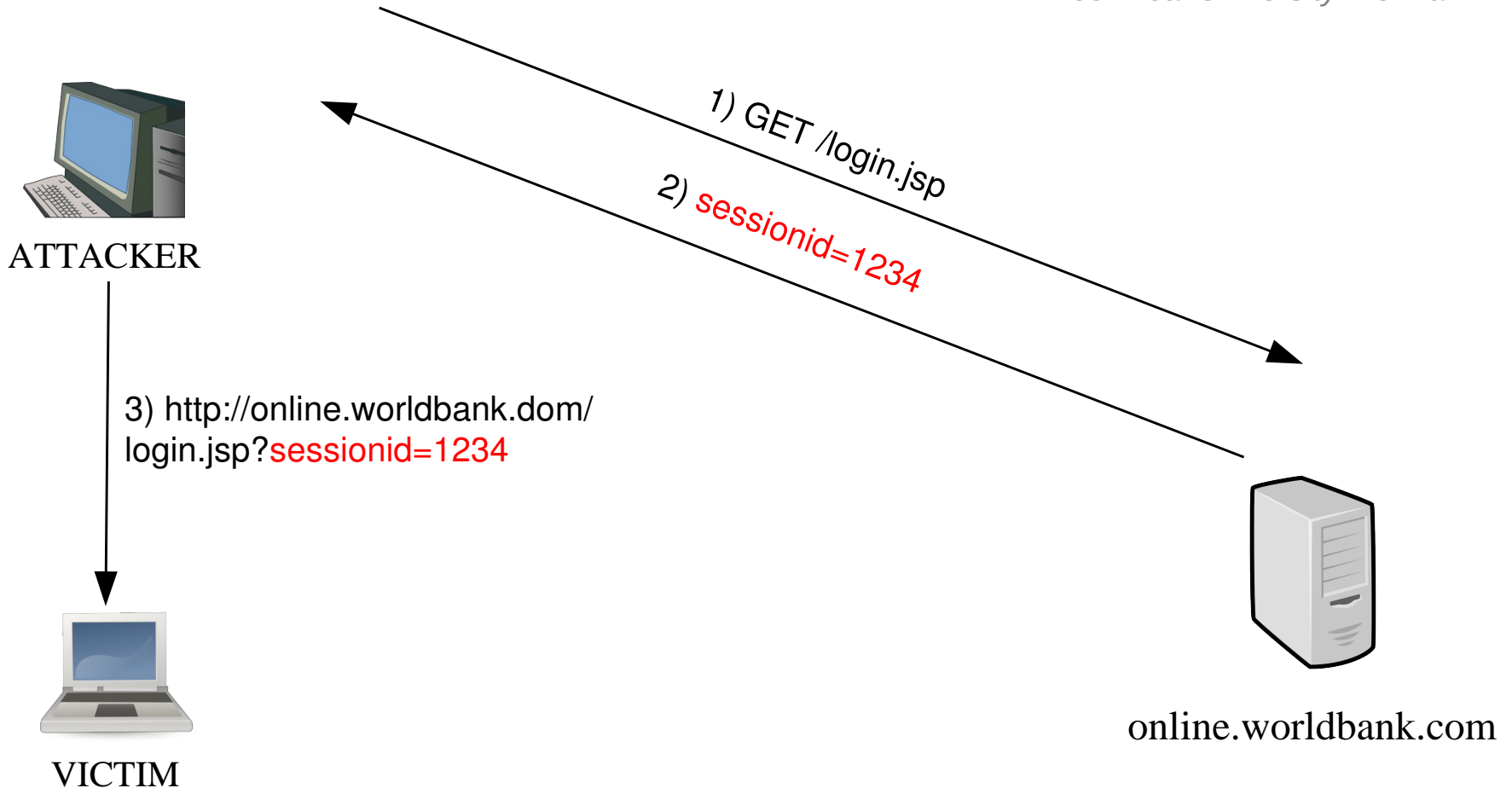


VICTIM

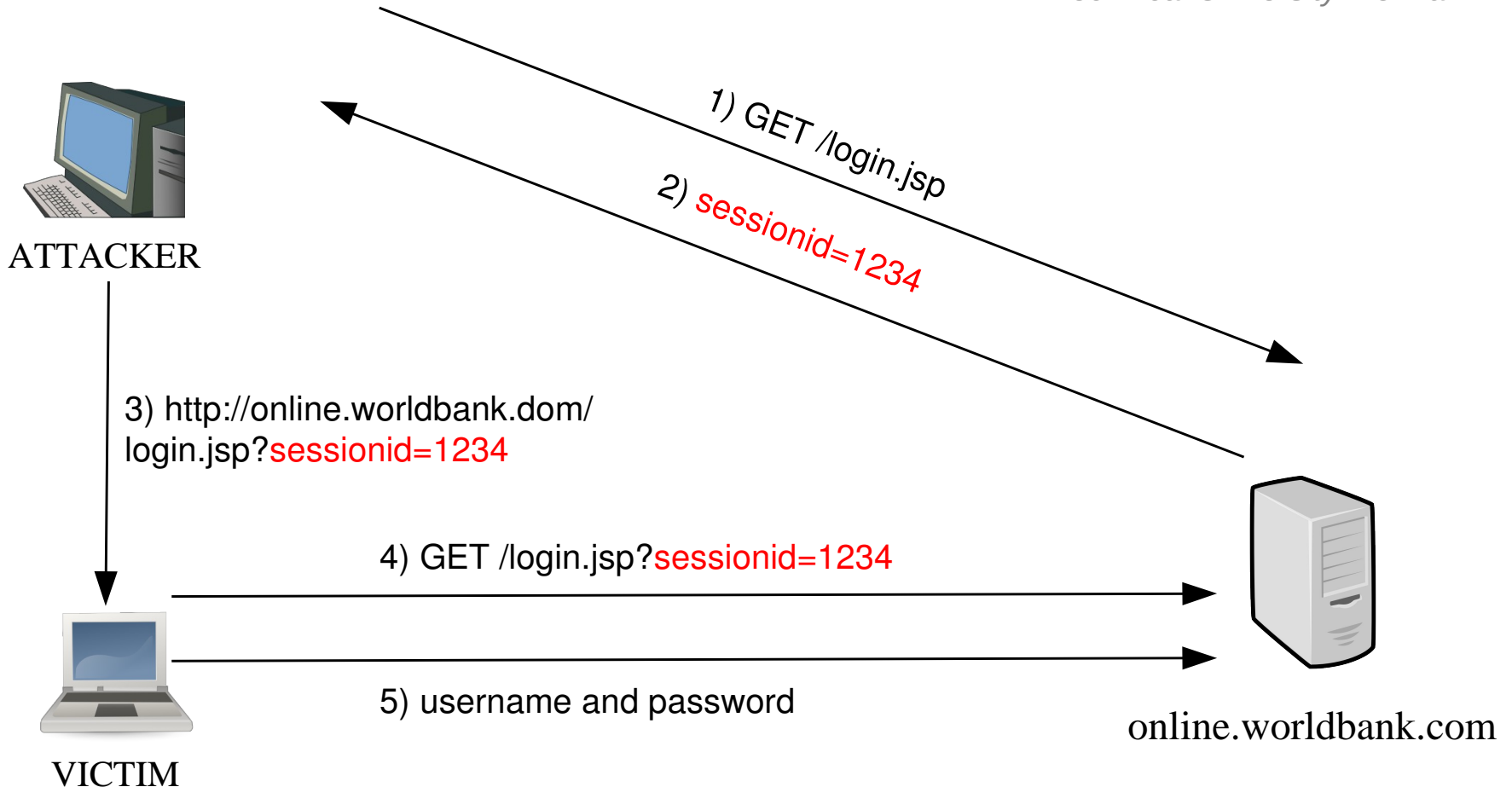


online.worldbank.com

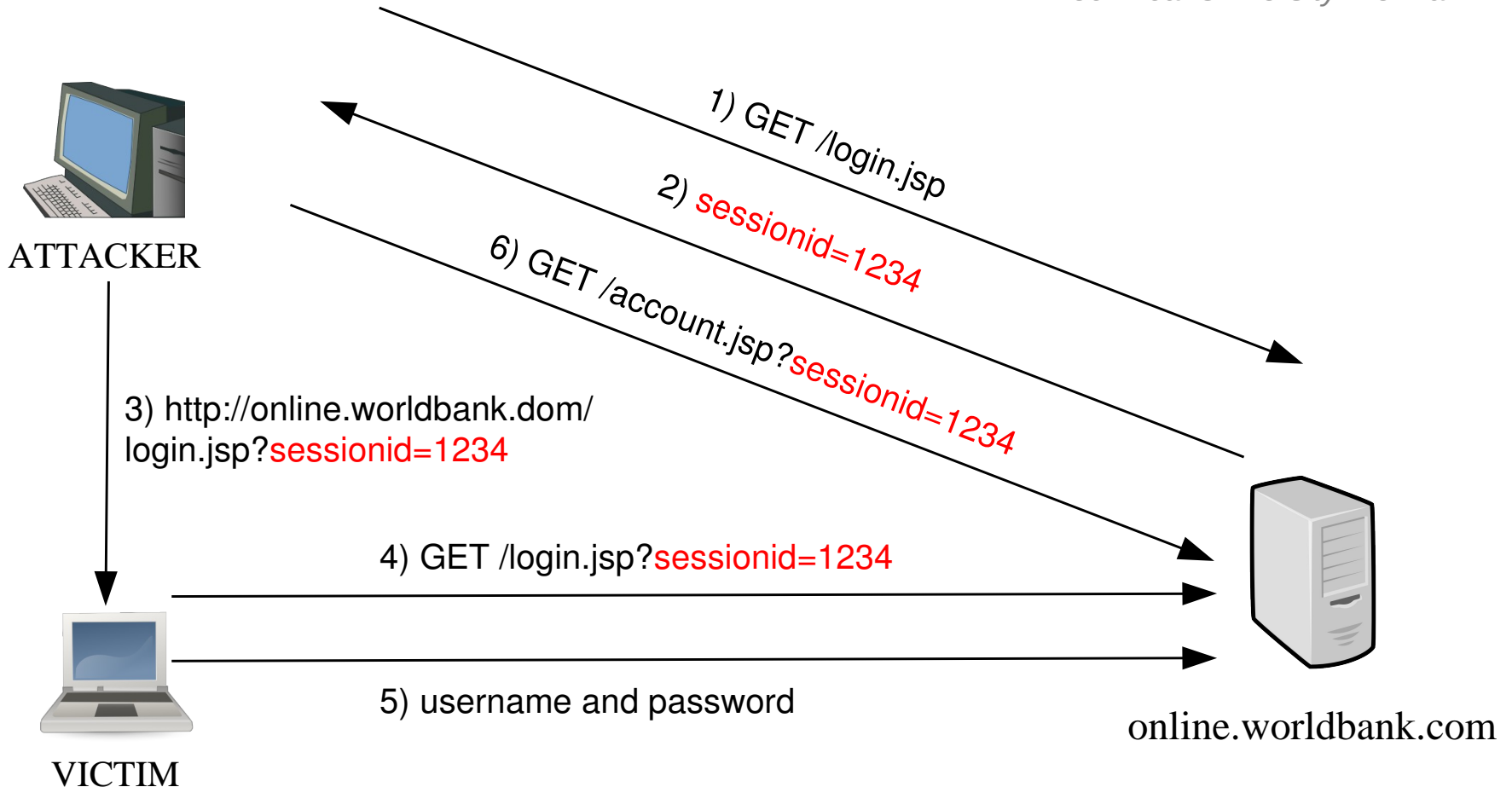
Another Scenario



Another Scenario



Another Scenario



Session Fixation vs Hijacking

*Int. Secure Systems Lab
Technical University Vienna*

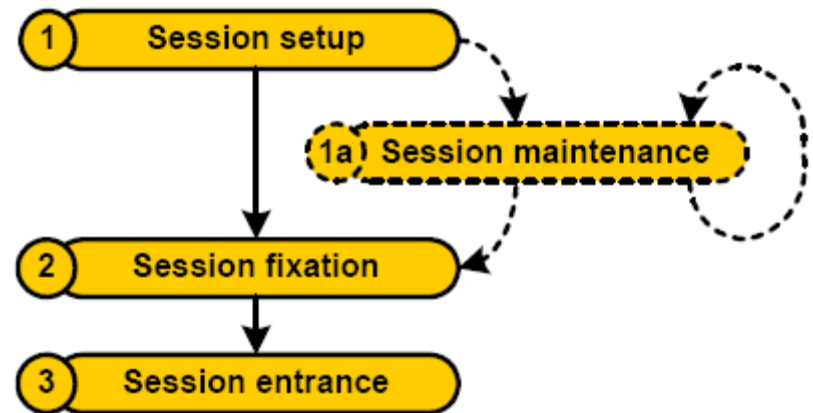
- Timing!
- Session Fixation:
 - Victim's browser is attacked **before** she logs into the target server
- Session Hijacking:
 - Victim's browser is attacked **after** she logs into the target server

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

*Int. Secure Systems Lab
Technical University Vienna*

- 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 form 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.com
- 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

Cross Site Request Forgery

*Int. Secure Systems Lab
Technical University Vienna*

- CSRF: A style of attack that lets attacker send arbitrary HTTP requests on behalf of a victim user
- 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

*Int. Secure Systems Lab
Technical University Vienna*

- 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) administrator 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:

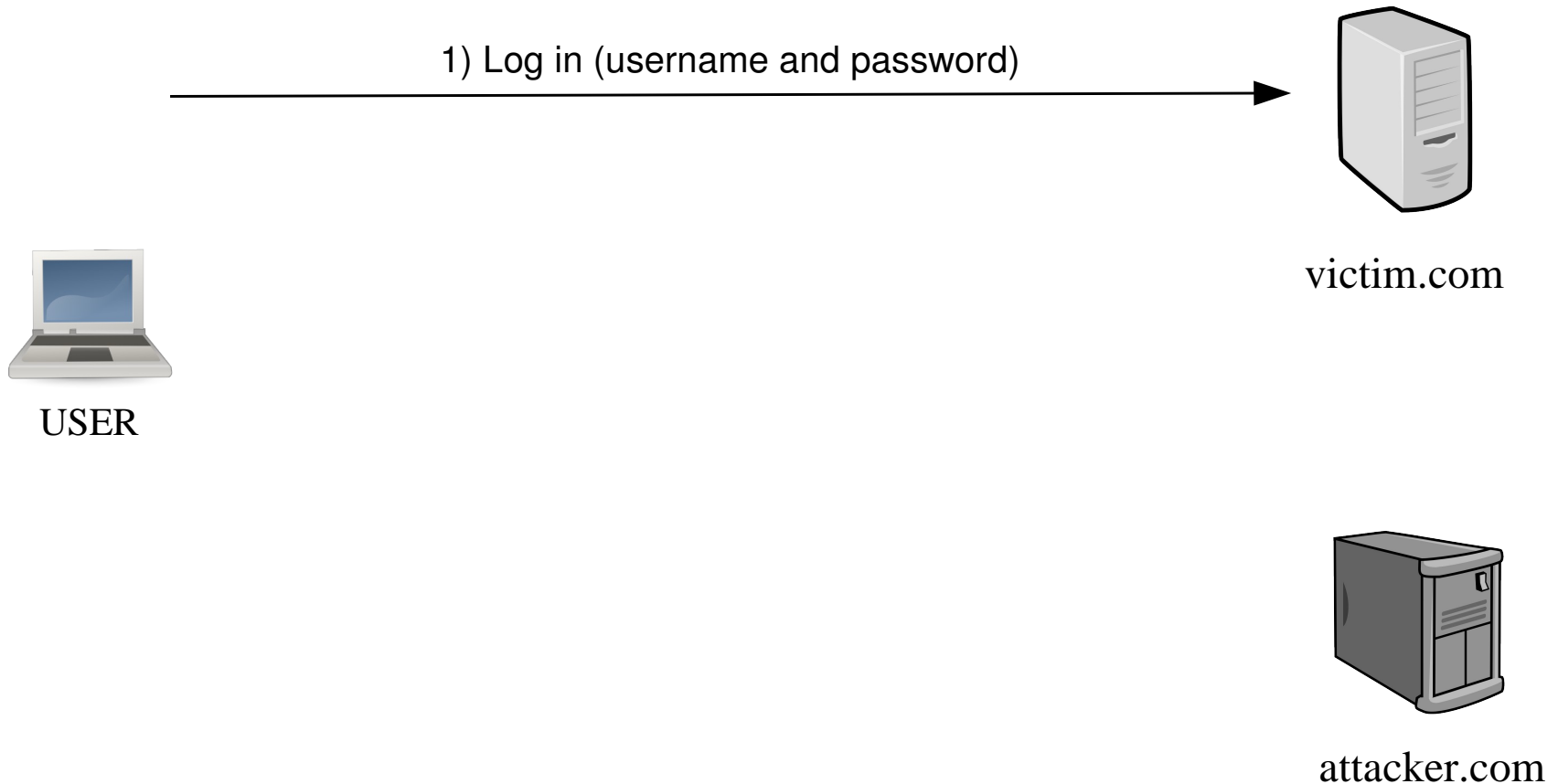
```

```

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

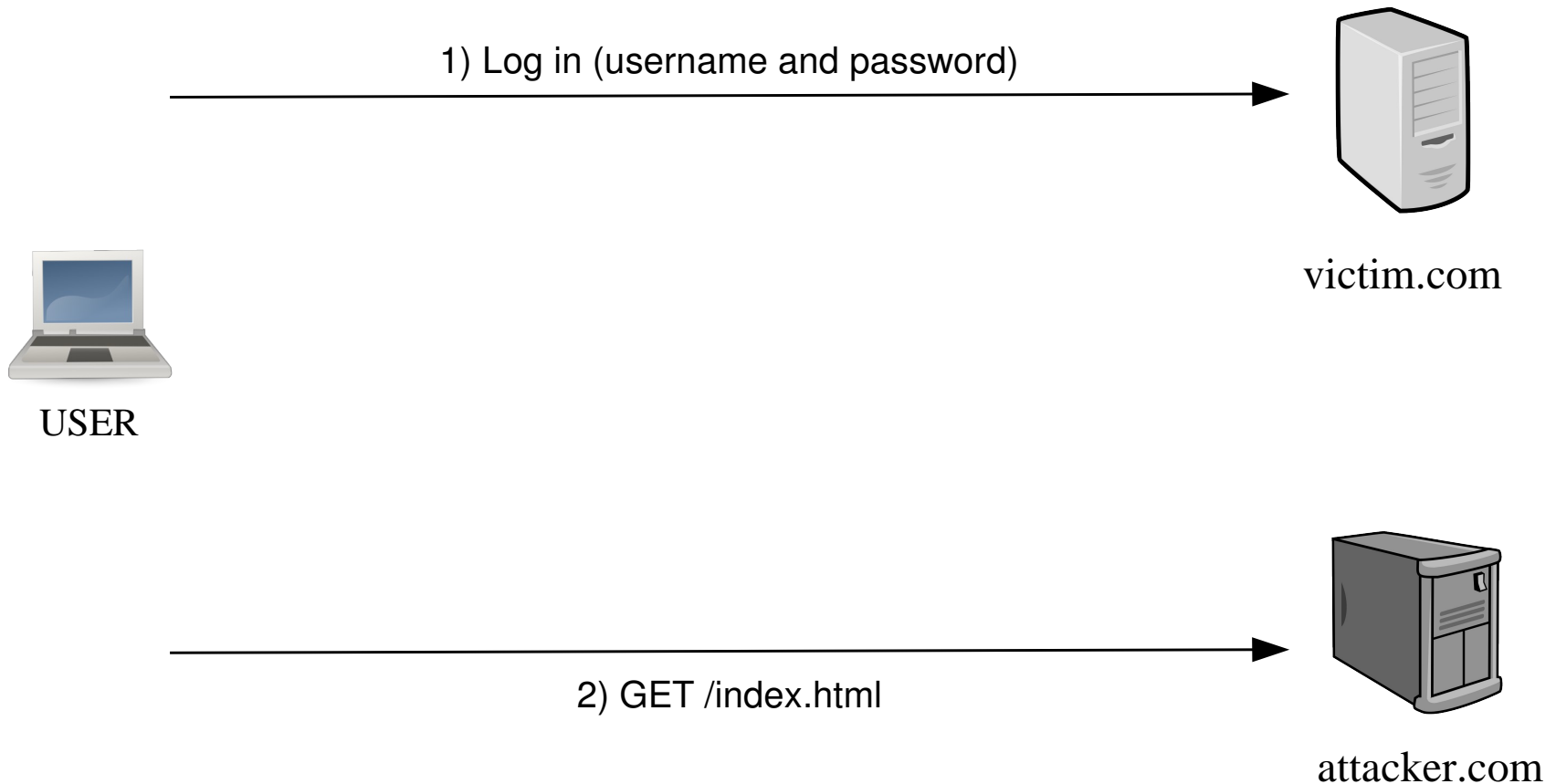
CSRF Example

*Int. Secure Systems Lab
Technical University Vienna*



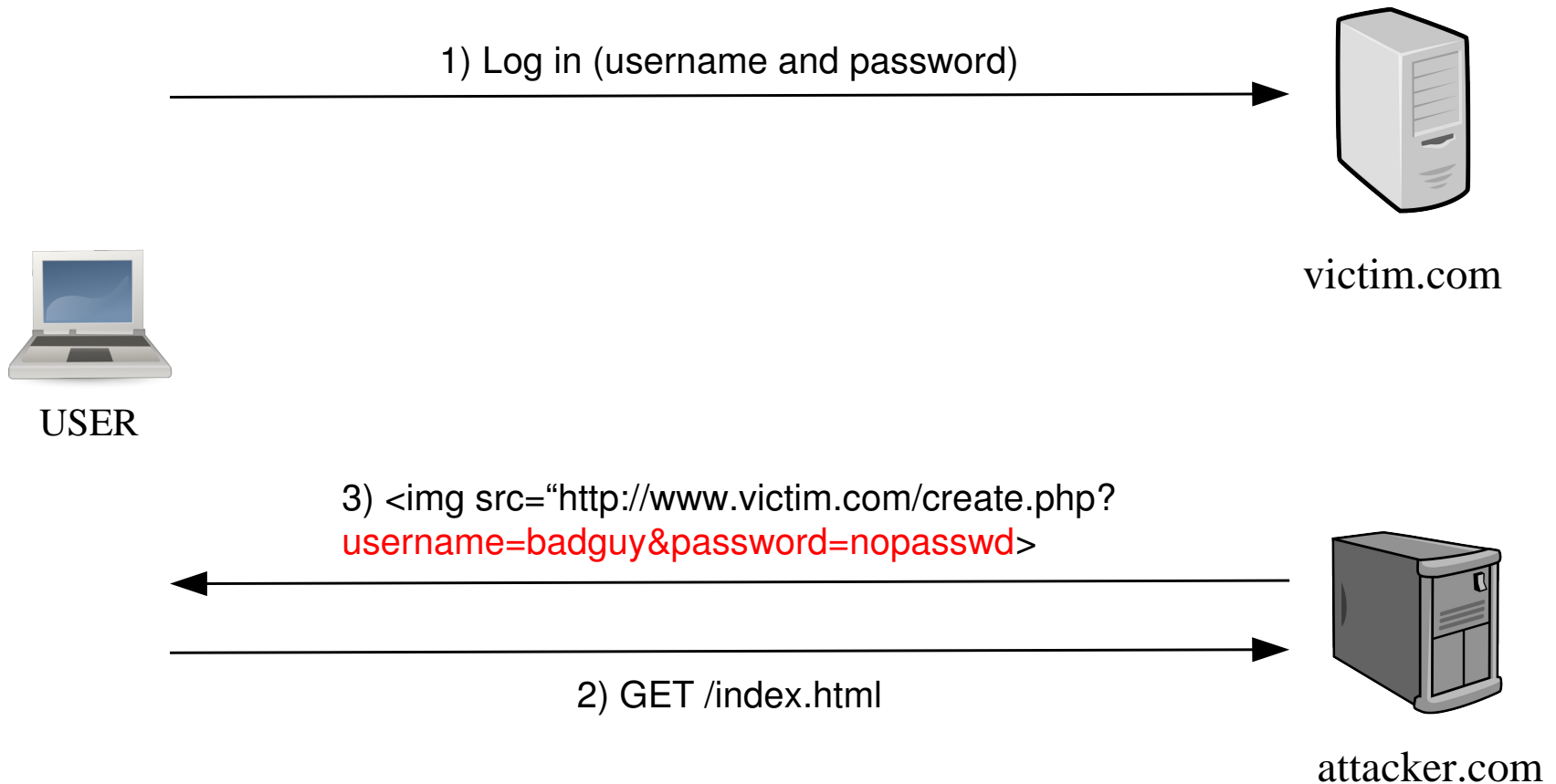
CSRF Example

*Int. Secure Systems Lab
Technical University Vienna*



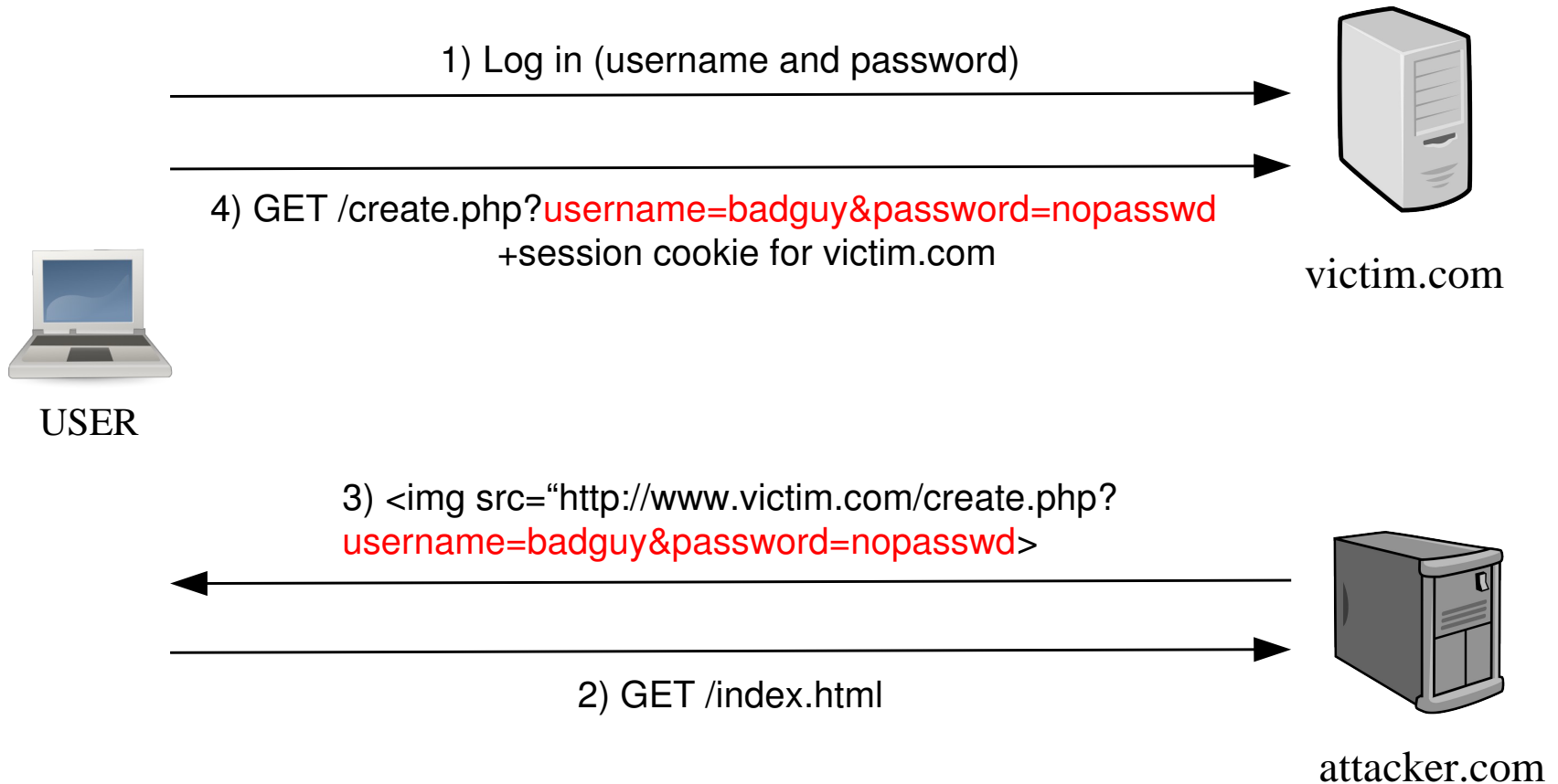
CSRF Example

Int. Secure Systems Lab
Technical University Vienna



CSRF Example

Int. Secure Systems Lab
Technical University Vienna



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

*Int. Secure Systems Lab
Technical University Vienna*

- 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

*Int. Secure Systems Lab
Technical University Vienna*

- 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

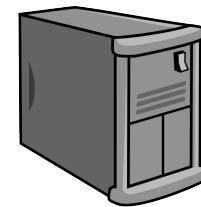
*Int. Secure Systems Lab
Technical University Vienna*



Home User
192.168.0.101



DSL router
192.168.0.1



attacker.cm

CSRF against home routers

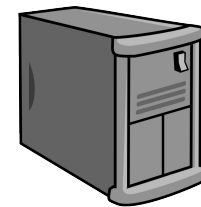
Int. Secure Systems Lab
Technical University Vienna



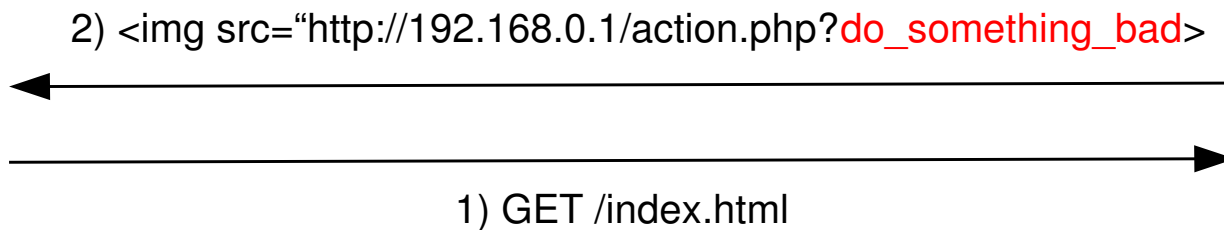
DSL router
192.168.0.1



Home User
192.168.0.101

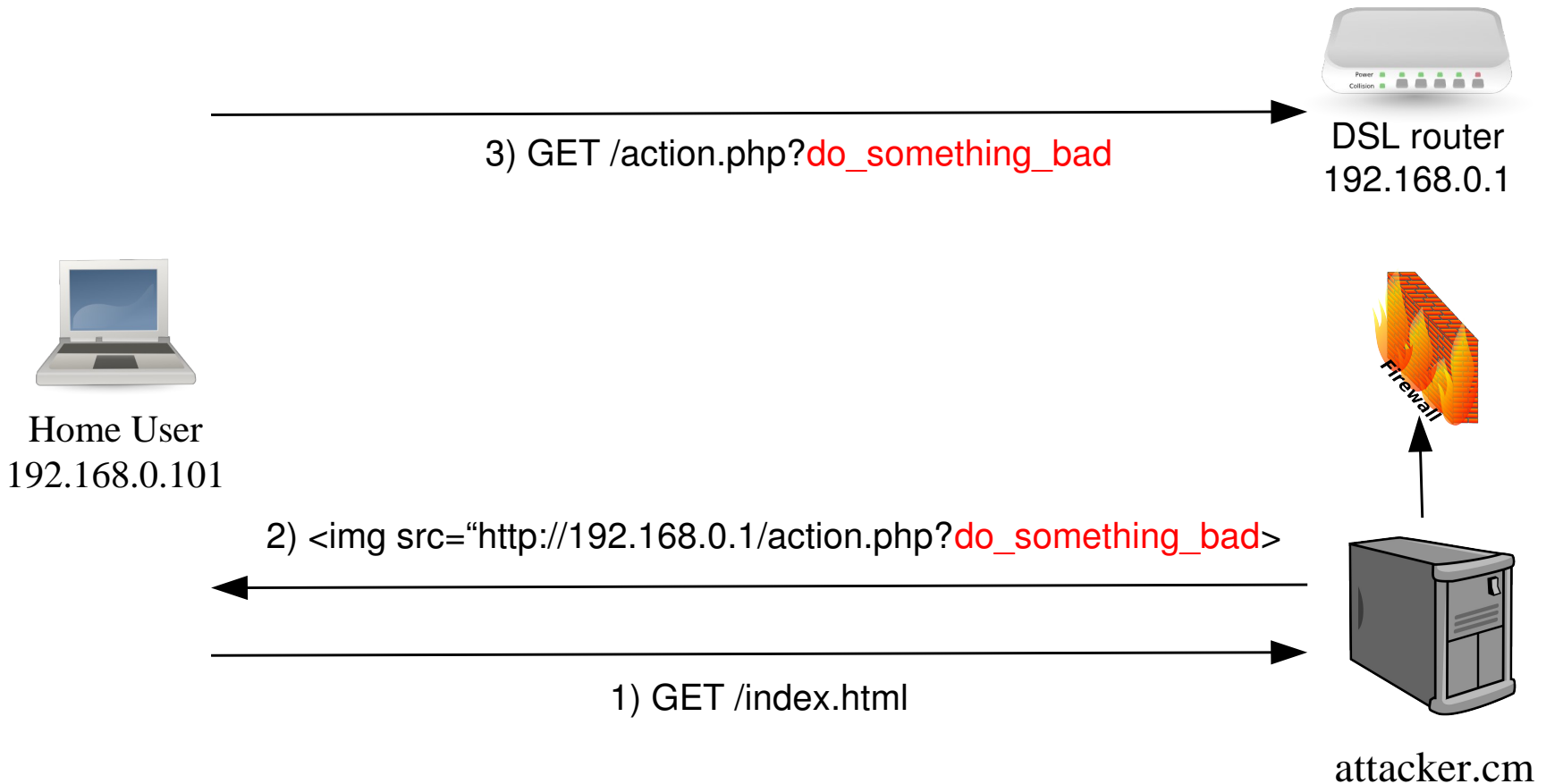


attacker.cm



CSRF against home routers

Int. Secure Systems Lab
Technical University Vienna



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&NEFTPAGE=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&NEFTPAGE=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

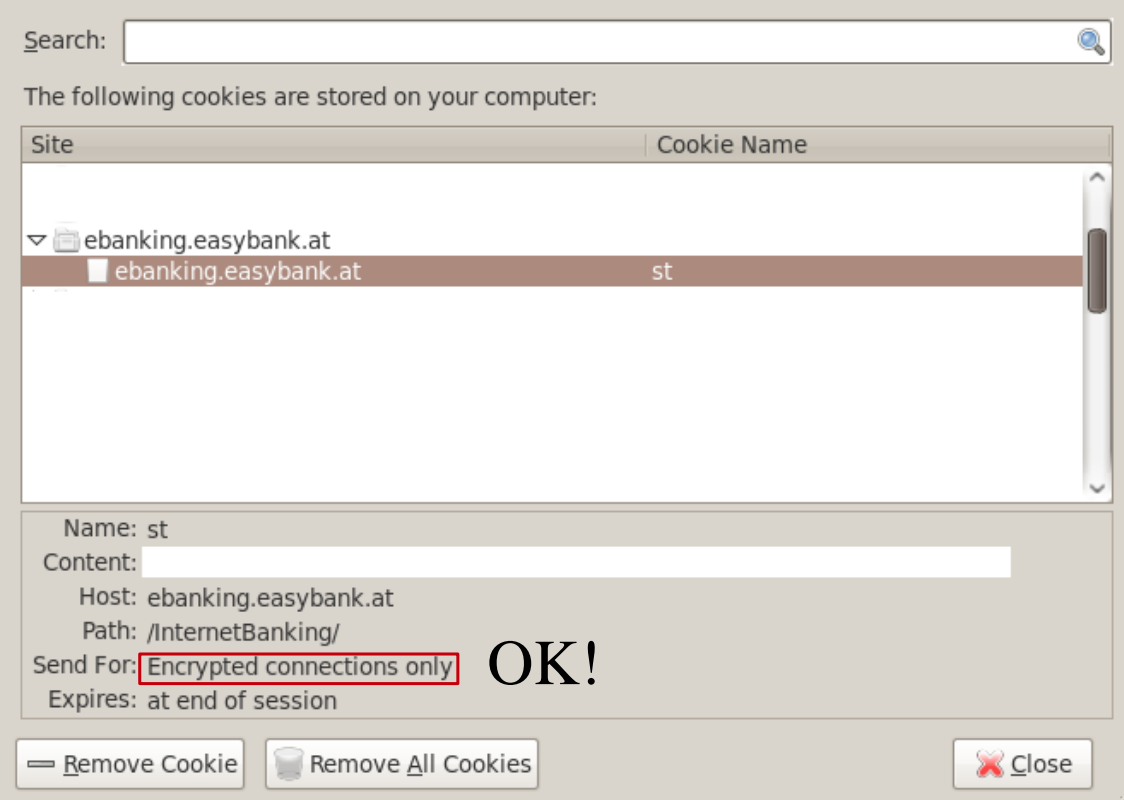
*Int. Secure Systems Lab
Technical University Vienna*

- 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

Int. Secure Systems Lab
Technical University Vienna

- 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



The screenshot shows a browser's cookie manager interface. At the top, there is a search bar. Below it, a message states: "The following cookies are stored on your computer:". A table lists the cookies with columns for "Site" and "Cookie Name". One cookie is listed for the site "ebanking.easybank.at" with the name "st". Below the table, the details for this cookie are shown: Name: st, Content: [redacted], Host: ebanking.easybank.at, Path: /InternetBanking/, Send For: Encrypted connections only (highlighted with a red box), and Expires: at end of session. To the right of the "Send For" field, the text "OK!" is displayed. At the bottom, there are three buttons: "Remove Cookie", "Remove All Cookies", and "Close".

SSL Stripping

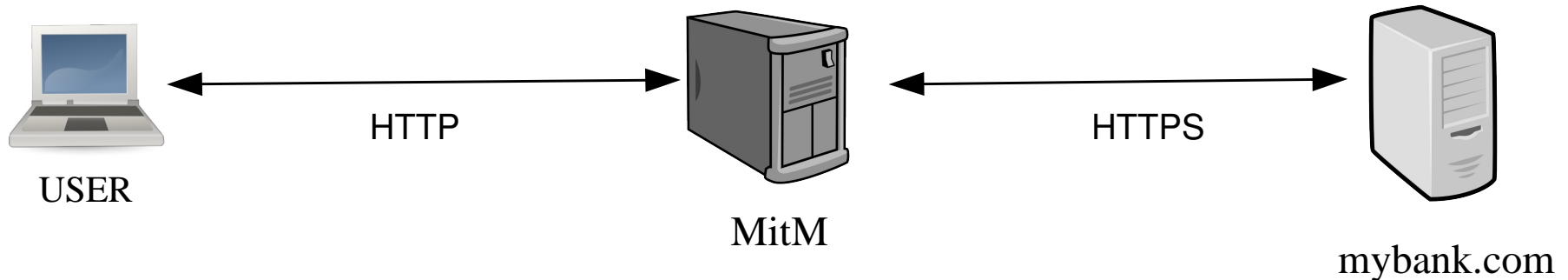
*Int. Secure Systems Lab
Technical University Vienna*

- A more powerful MitM attack against HTTPS
- Old problem, recently highlighted by:
 - <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

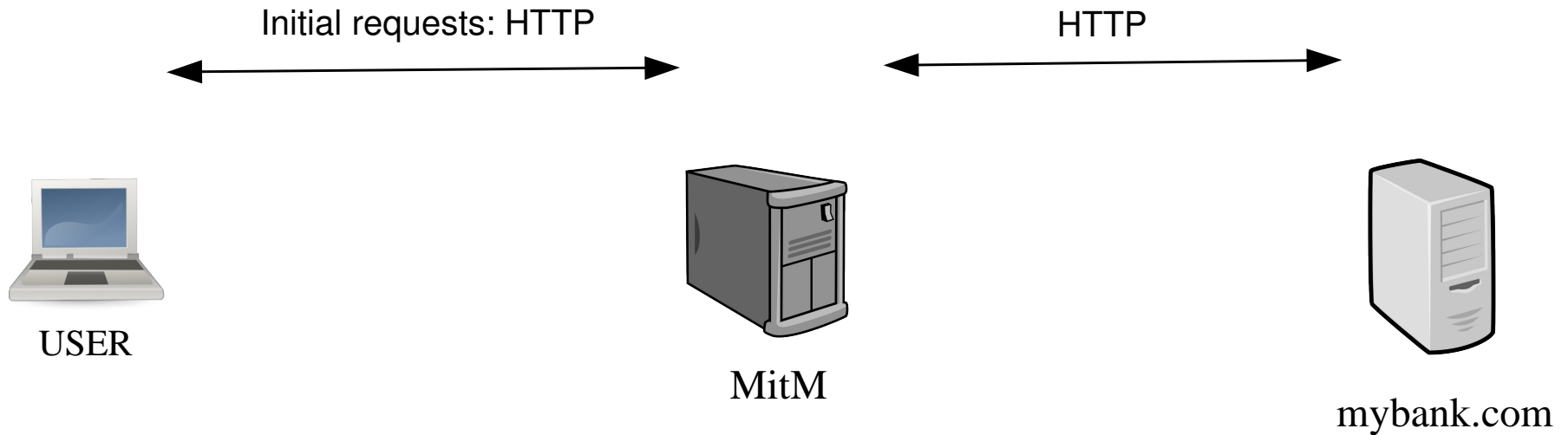
*Int. Secure Systems Lab
Technical University Vienna*

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



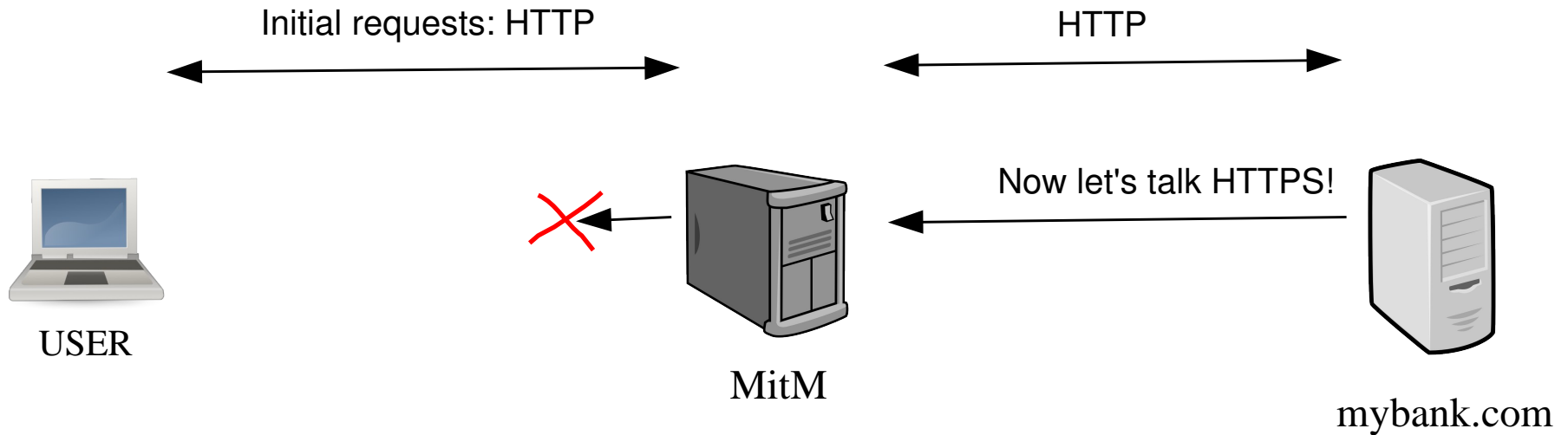
SSL Stripping

*Int. Secure Systems Lab
Technical University Vienna*



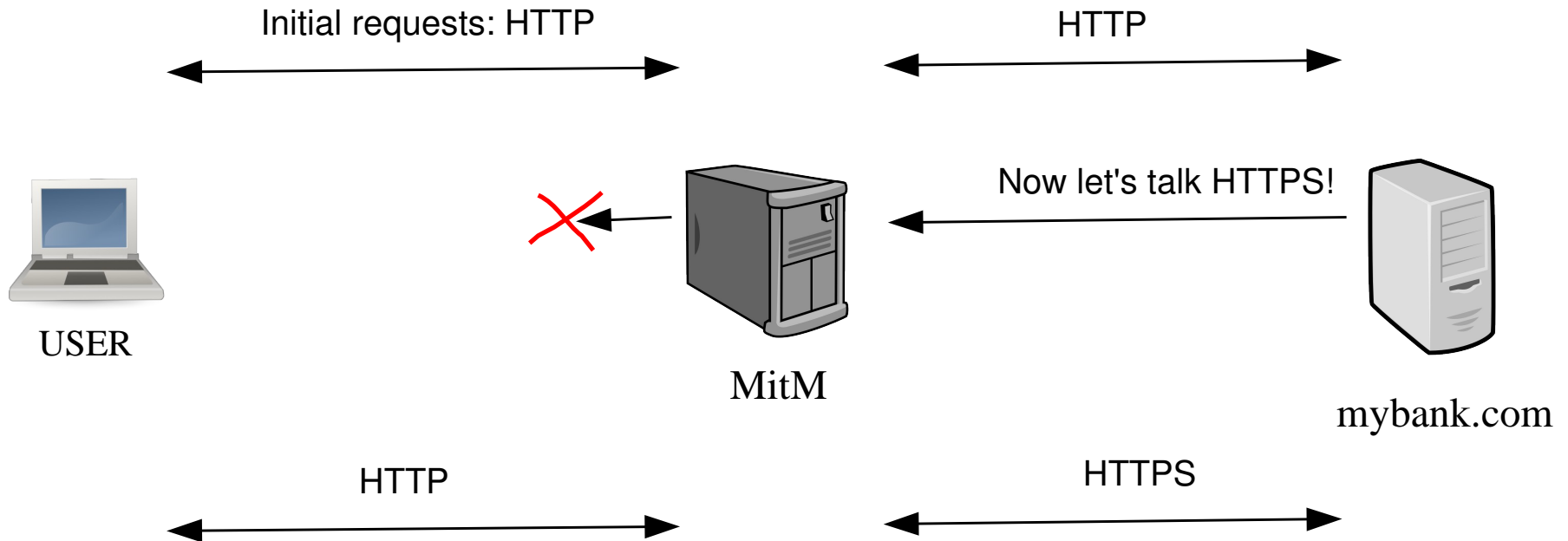
SSL Stripping

*Int. Secure Systems Lab
Technical University Vienna*



SSL Stripping

*Int. Secure Systems Lab
Technical University Vienna*



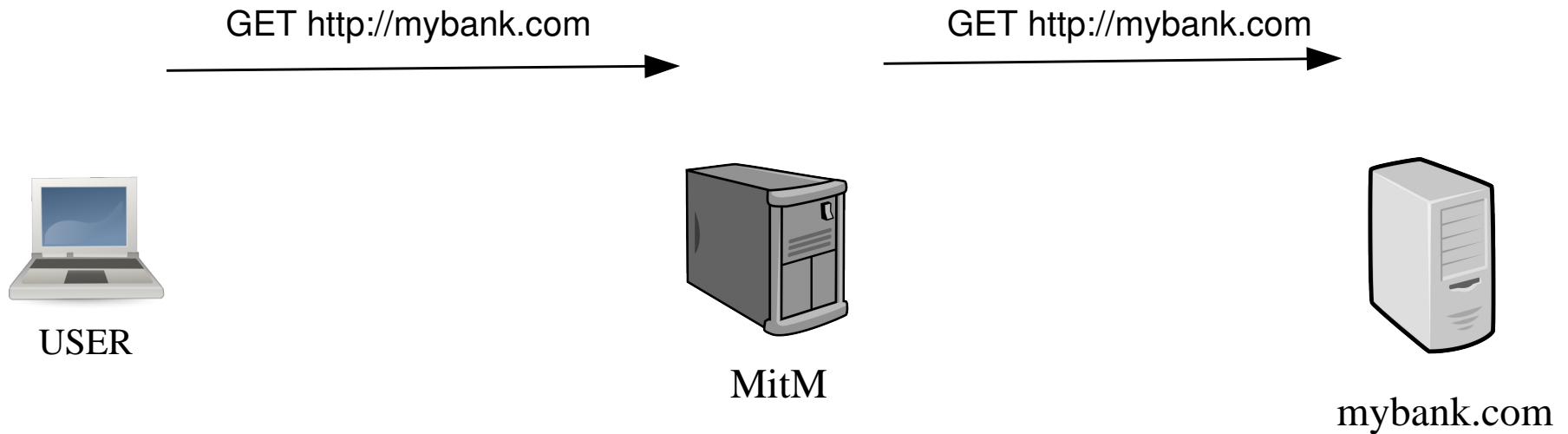
SSL Stripping

*Int. Secure Systems Lab
Technical University Vienna*

- 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!

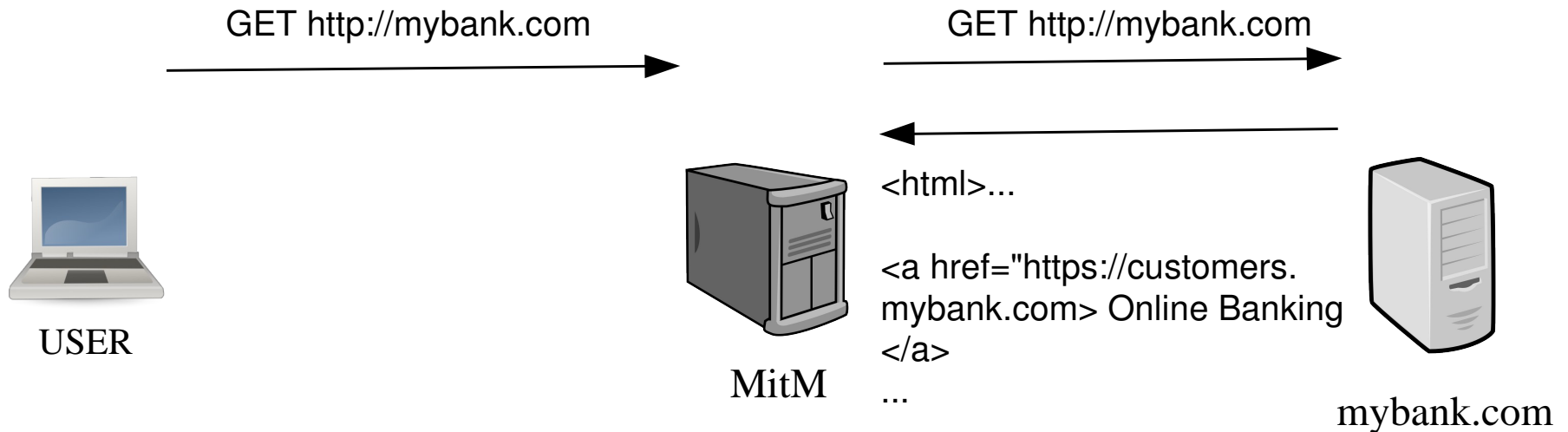
SSL Stripping

*Int. Secure Systems Lab
Technical University Vienna*



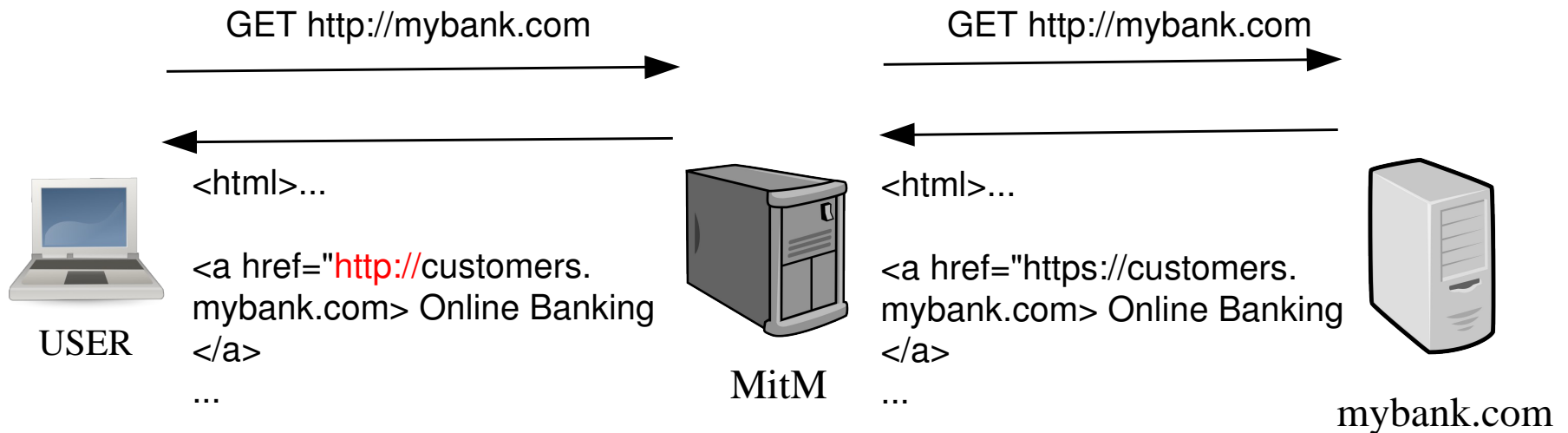
SSL Stripping

*Int. Secure Systems Lab
Technical University Vienna*

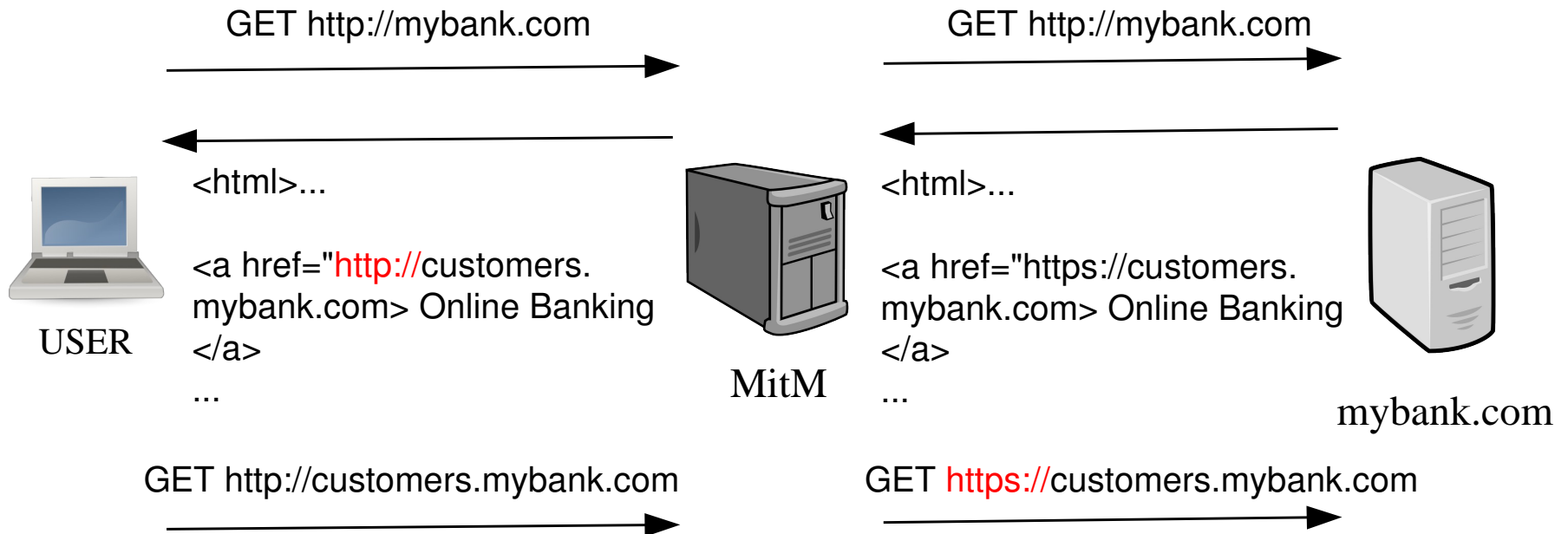


SSL Stripping


*Int. Secure Systems Lab
Technical University Vienna*



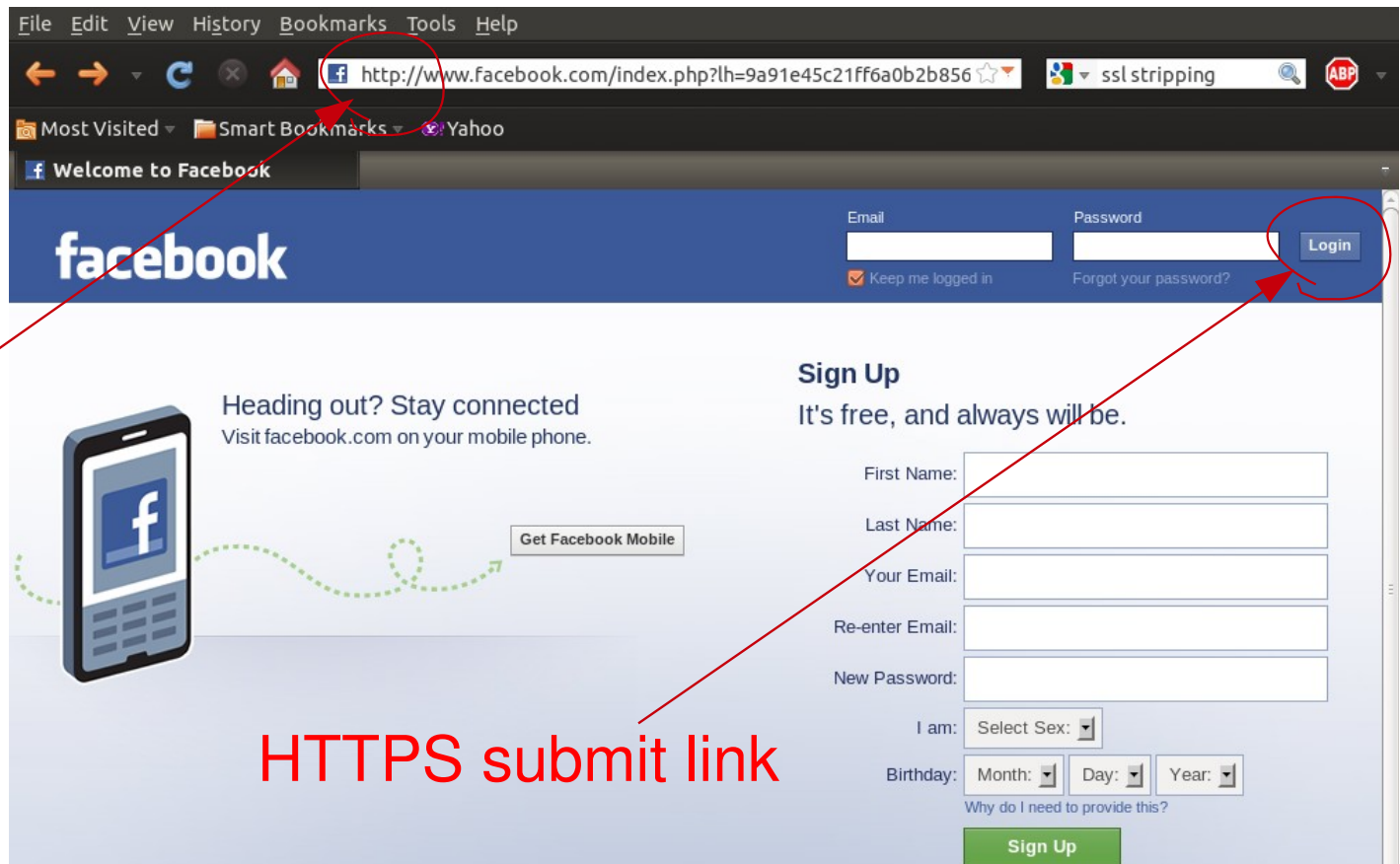
SSL Stripping



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

HTTPS submit link

Implementing SSL Stripping

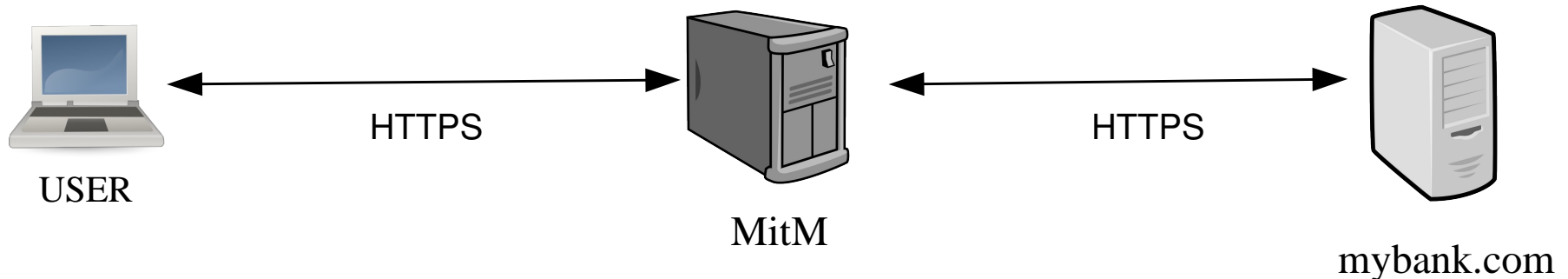
*Int. Secure Systems Lab
Technical University Vienna*

- 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

*Int. Secure Systems Lab
Technical University Vienna*

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



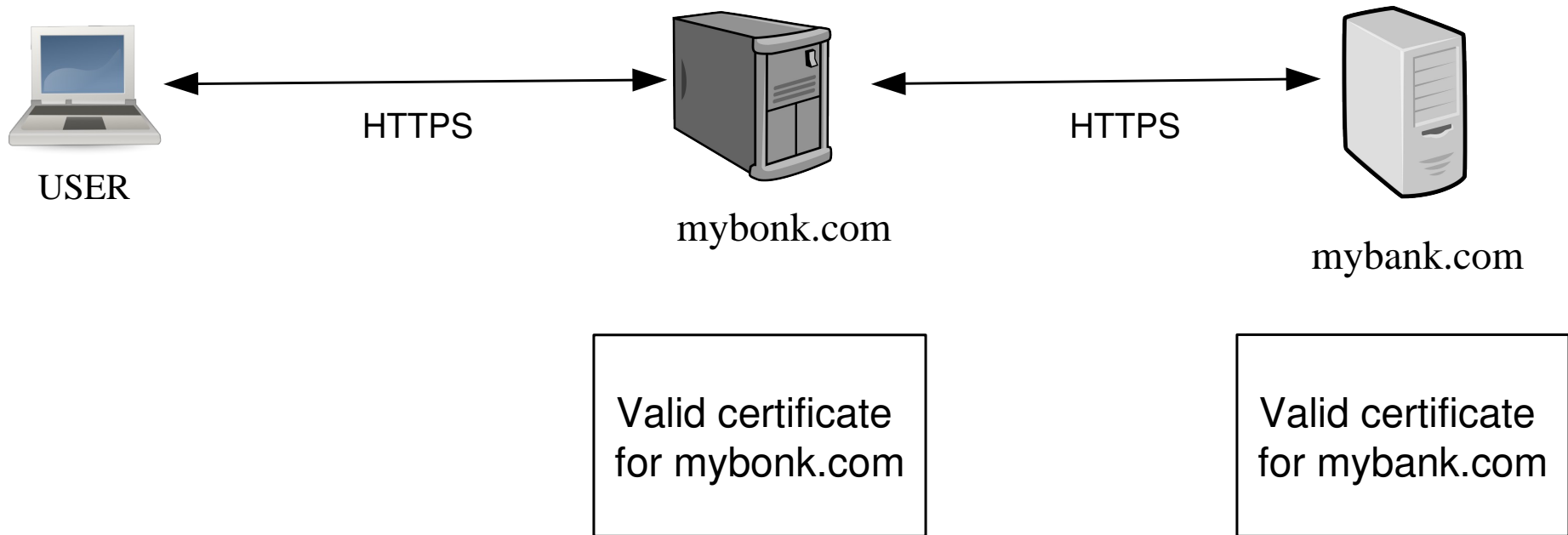
SSL Stripping Variant

*Int. Secure Systems Lab
Technical University Vienna*

- 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

*Int. Secure Systems Lab
Technical University Vienna*



Recent SSL Stripping Attack

*Int. Secure Systems Lab
Technical University Vienna*

- DigiNotar hack
- July 10, 2011, wildcard certificates for google, etc.
- Short Video
 - <https://www.youtube.com/watch?v=wZsWoSxxwVY>

Browser History Stealing (Historical)

Browser History Stealing

- Browsers display visited links differently from non-visited links
 - very helpful to users (which google search results have I already visited?)
- This can be used to find out if the user has visited a site
 - has user visited example.com?
 - attacker prepares webpage at attacker.com with link:
``
 - attacker checks how the link is displayed

Browser History Stealing

*Int. Secure Systems Lab
Technical University Vienna*

- How can the attacker check the link color?
- the link is displayed in the web page attacker.com
 - SOP allows attacker to access the content
 - use javascript to check color or font size
- Attack can also be performed without javascript
 - using CSS (Cascading Style Sheets)

History Stealing without Javascript

*Int. Secure Systems Lab
Technical University Vienna*

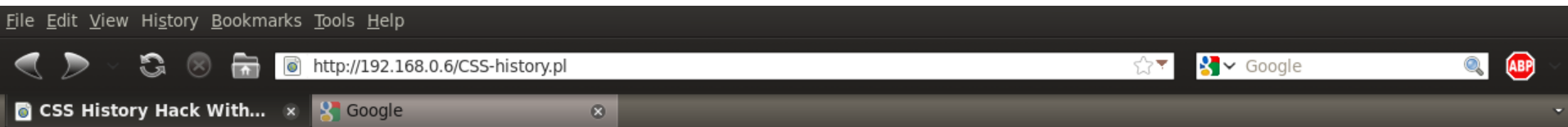
- CSS is used to improve HTML by separating presentation from document structure
- HTML defines the content
- CSS specifies how to display that content
 - uses simple rules to match content
 - specify how to display content matching the rules
- a:visited
 - define the style for visited links
 - a style can include a background image
 - background: url(log_visited.php?example.com)
 - use div+span to associate different "background" urls to each link

History Stealing Proofs of Concept

*Int. Secure Systems Lab
Technical University Vienna*

- CSS based
 - <http://ha.ckers.org/weird/CSS-history.cgi>
 - used for this demo

- Javascript based
 - <http://jeremiahgrossman.blogspot.com/2006/08/i-know-where-youve-been.html>



CSS History Hack Without JavaScript

http://www.yahoo.com/	VISITED
http://www.google.com/	VISITED
http://www.myspace.com/	
http://www.msn.com/	
http://www.ebay.com/	VISITED
http://ha.ckers.org/	

Now [click here](#) to see that your information has been logged.

This is an example of how [the original CSS history hack found by Jeremiah Grossman](#) can be modified to work without a single line of JavaScript. It uses the fact that properties within **display:** when combined with **a:visited** creates conditional logic. That condition will not fire certain things within the block. In this case I am including a nonexistant background image **background: url(...);** set in the CSS itself that is seamless to the user. The image actually points to a CGI script with the information about the URL that has been visited and is then logged along with the IP address of the user for later retrieval.

This should be mitigated by the [SafeHistory plugin in Firefox](#) but turning off JavaScript has no effect on the technique in the browsers tested (Internet Explorer 7.0 and Firefox 2.0.0.2). This did not work in Opera 9.02 when tested. Note: I chose this list because it is the Alexa top 5 in the United States. I also added one that's on the same site as this demo so it should not be blocked by cross domain restrictions imposed by SafeHistory.

- By [RSnake](#)

http://192.168.0.6/CSS-history.pl?::ffff:192.168.0.1.90005466372

noscript is on

Browser History Stealing

*Int. Secure Systems Lab
Technical University Vienna*

- Browser history can leak user's private information
 - not just porn!..
- Recent work from seclab:
 - de-anonymizing users of social networks using browser history stealing
 - checks for URLs specific to facebook/xing/linkedin groups in browser history
 - the groups you are part of, can uniquely identify you

Attacks exploiting the caching infrastructure

Cache busting

- HTTP responses are subject to caching (e.g., by a proxy)
 - Suppose you have developed an application and are dropping a new cookie on request to URL:
 - http://sometesturl.com/welcome/register_for_info.cgi
 - A QueryString name/value pair is not used.
 - The request to the same URL might be considered a cache hit and the cached page (including the cookie!) might be returned to the user
- solution: cache-busting (i.e., prevent that certain data is cached)

Cache busting

- A popular technique when cookies are disabled:
 - If no session ID is presented, an HTTP Redirect (302) is issued and the client is redirected to session assignment
 - URL-encode a session ID through dynamic modification of the links in the page. E.g.:
 - <http://www.helloworld.org/index.html> →
<http://www.helloworld.org/index.html?session=1234>
 - This approach is vulnerable... if the page with hard-coded sessions is cached, many users may see each others sessions (!).
 - Furthermore, people may bookmark the links and this may cause more confusion

Cache Busting

Int. Secure Systems Lab
Technical University Vienna

- Cookies can be used for flawless client tracking
 - If they are used in conjunction with SSL to prevent caching of *Set-Cookie* header
 - Avoid issuing Set-Cookie headers in response to HTTP requests that use the GET request method.
 - use *Cache-Control* headers to forbid caching
 - All sessions must expire after a short amount of inactivity (e.g., 20 minutes)

Cache Busting

*Int. Secure Systems Lab
Technical University Vienna*

- The only guaranteed secure way to prevent cache-effects:
 - Use SSL as part of the process of meeting and greeting a new client
- The logic for initial SSL-secured HTTP request is simple:
 - 1.) If no valid session ID, redirect browser to `https:// session generator: Cookie is set`
 - 2.) Redirect browser to original `http:// URL`

Because SSL is used, cookie cannot be cached.

MS .NET Passport uses this approach

HTTP Response Splitting

*Int. Secure Systems Lab
Technical University Vienna*

- 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

*Int. Secure Systems Lab
Technical University Vienna*

- 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

*Int. Secure Systems Lab
Technical University Vienna*

- 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

*Int. Secure Systems Lab
Technical University Vienna*

- 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/1.1 302 Moved Temporarily
...
Location: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: 21\r\n
\r\n
<html> H4x0r! </html>
\r\n
<html><head><title>
302 Moved Temporarily
</title></head></html>
```

First HTTP reply

HTTP Response Splitting

- This is what the whole reply now looks like

HTTP/1.1 302 Moved Temporarily ... Location: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: 21\r\n \r\n <html> H4x0r! </html>
\r\n <html><head><title> 302 Moved Temporarily </title></head></html>

First HTTP reply

Second reply (page defacement)

HTTP Response Splitting

- This is what the whole reply now looks like

<pre>HTTP/1.1 302 Moved Temporarily ... Location:foo\r\n Content-Type: text/html\r\n Content-Length: 0\r\n \r\n</pre>	First HTTP reply
<pre>HTTP 1.1 200 OK\r\n Content-Type:text/html\r\n Content-Length: 21\r\n \r\n <html> H4x0r! </html></pre>	Second reply (page defacement)
<pre>\r\n <html><head><title> 302 Moved Temporarily </title></head></html></pre>	Garbage (ignored)

HTTP Response Splitting

*Int. Secure Systems Lab
Technical University Vienna*

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>...
```

Conclusions

*Int. Secure Systems Lab
Technical University Vienna*

- We covered a range of advanced topics in web security
 - session fixation
 - https cookie theft
 - cross-site request forgery
 - browser history stealing
 - attacks exploiting the caching infrastructure

- Next week, no lecture