Secure Programming and Common Errors

brought to you by Michele “antisnatchor” Orru’
Computer System Security course lead by Prof. Ozalp Babaoglu
5 May 2009

Who am I ?

- Bachelor Degree in Internet Sciences
- Independent Security Researcher
- Owner of http://antisnatchor.com security advisory blog
- Collaborator of Apache OFBiz (ofbiz.apache.org) and OpenTaps(www.opentaps.com)
- JEE developer

Seminar Objectives

- Discuss the most relevant SANS top 25 errors that concern Web Applications
- Practical demonstrations of some vulnerable Real World web applications (my totally independent security research)
- Understand the impact of these threats on the most valuable web-app assets

What we’ll discuss

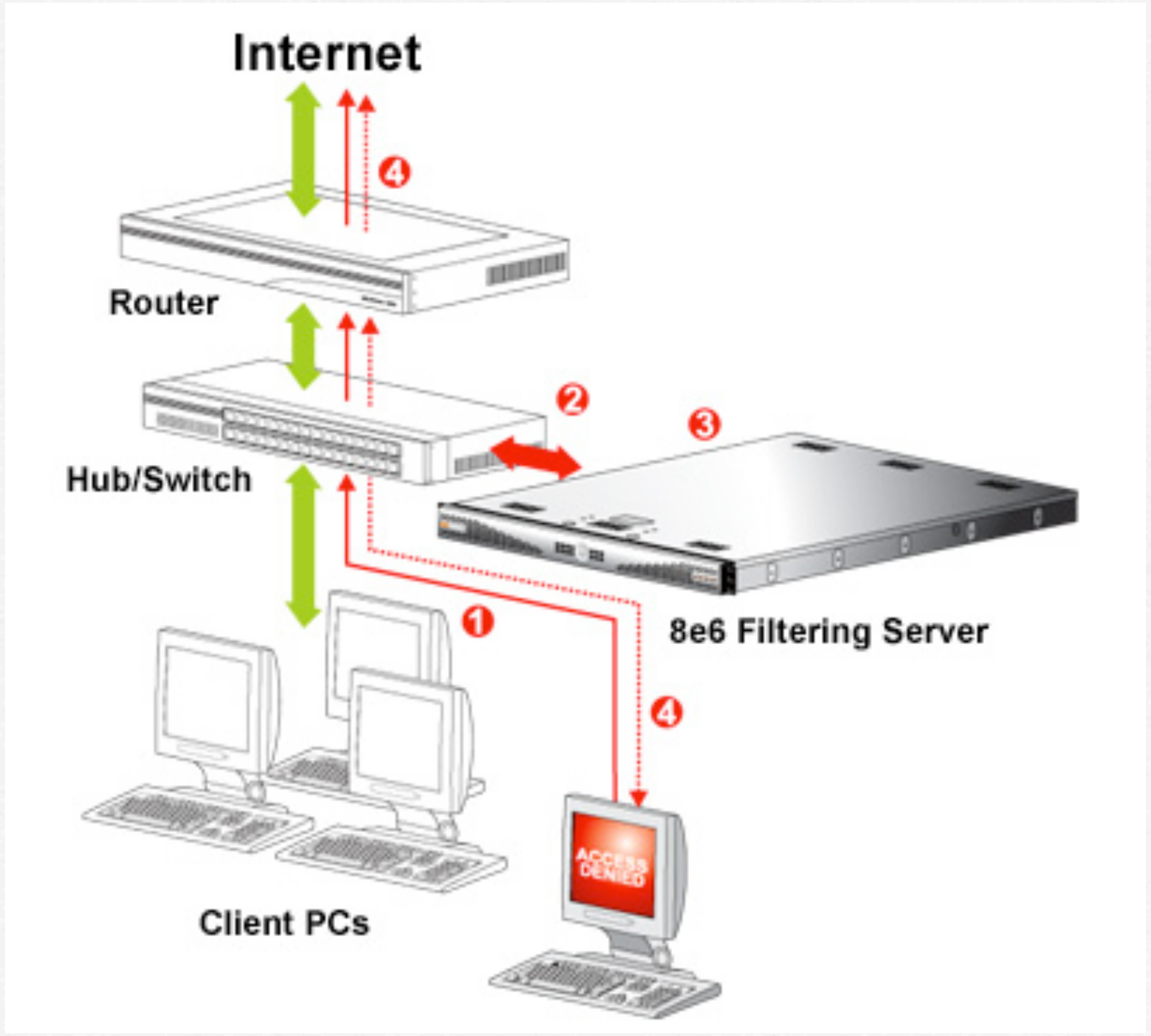
- CWE-20: Improper Input Validation
- CWE-116: Improper Encoding or Escaping of Output
- CWE-209: Error Message Information Leak
- CWE-89: Failure to Preserve SQL Query Structure (SQL injection)
- CWE-79: Failure to Preserve Web Page Structure (XSS)
- CWE-352: Cross-Site Request Forgery (XSRF)

CWE-20: Improper Input Validation

- The biggest issues on today’s Internet Applications (not just WebApps)
- Improper Input Validation can lead to security vulnerabilities when attackers can modify input in unexpected ways for the application
- The only way to protect our applications is by understanding that all input can be malicious

CWE-20: Example

- **8e6 R3000 Internet Filter** (commercial HTTP(s) Proxy filter solution)



CWE-20: Example

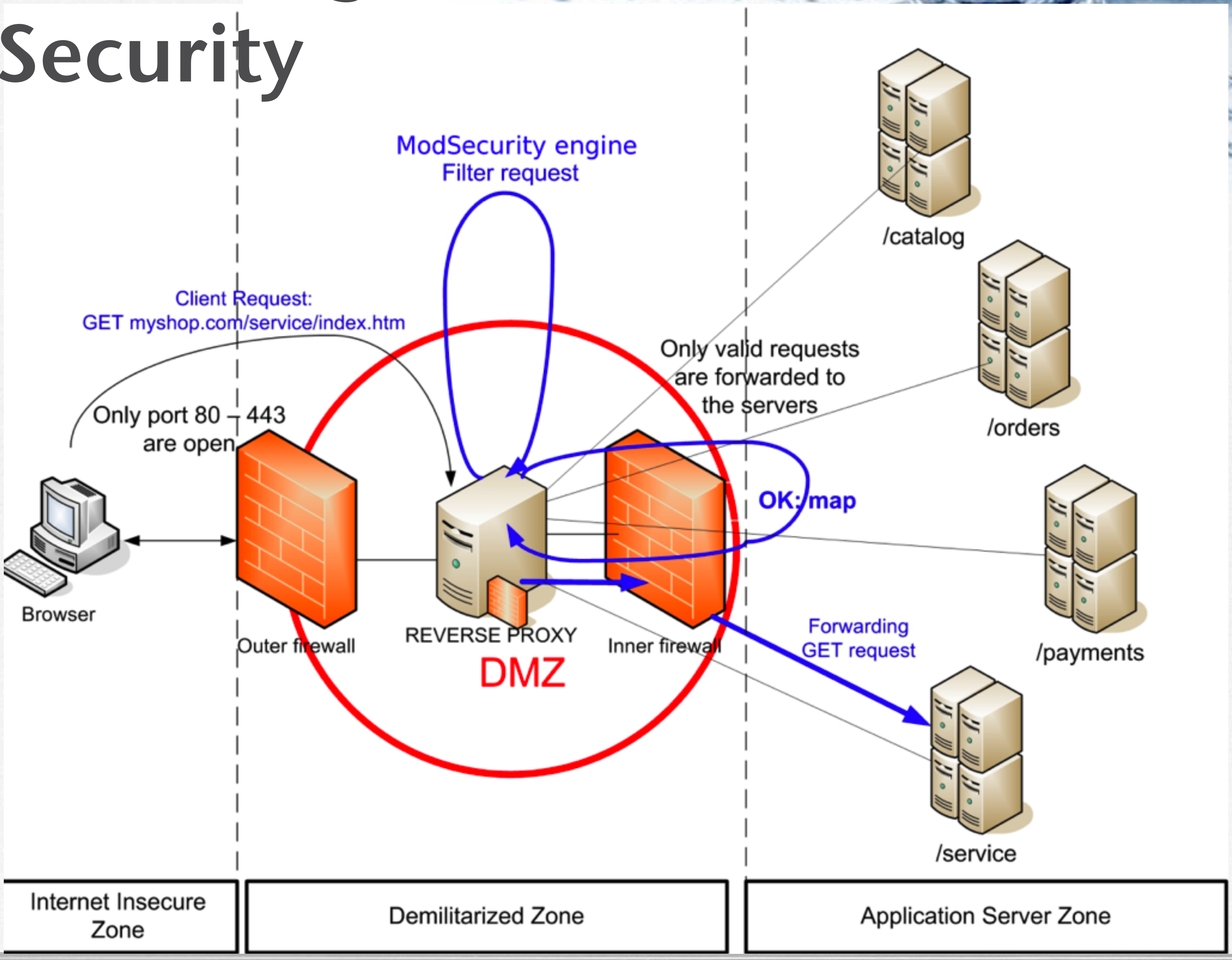
- Credits: *nnposter*
- DNS based website blacklist can be bypassed by providing a forged request with custom HTTP header

```
GET / HTTP/1.1
X-DecoyHost: www.milw0rm.org
Host: www.blocked.org
```

CWE-20: Mitigation

- Understand every potential attacks areas: parameters, arguments, cookies, headers, files, databases...
- **Whitelist approach** instead of blacklist (you’re gonna certainly miss some character encoding variants)
- WebApp case: use a WebApp Firewall (ModSecurity/F5) or an Input Validation Framework for your language.

CWE-20: Mitigation ModSecurity



CWE-20:Mitigation OWASP ESAPI

A common set of interfaces for security controls such as:

- Authentication
- Access Control
- **Input Validation**
- Output Encoding
- Cryptography (secure Java implementation of md5/sha*/BlowFish/AES)
- Error handling/logging

CWE-20:Mitigation PHPIDS

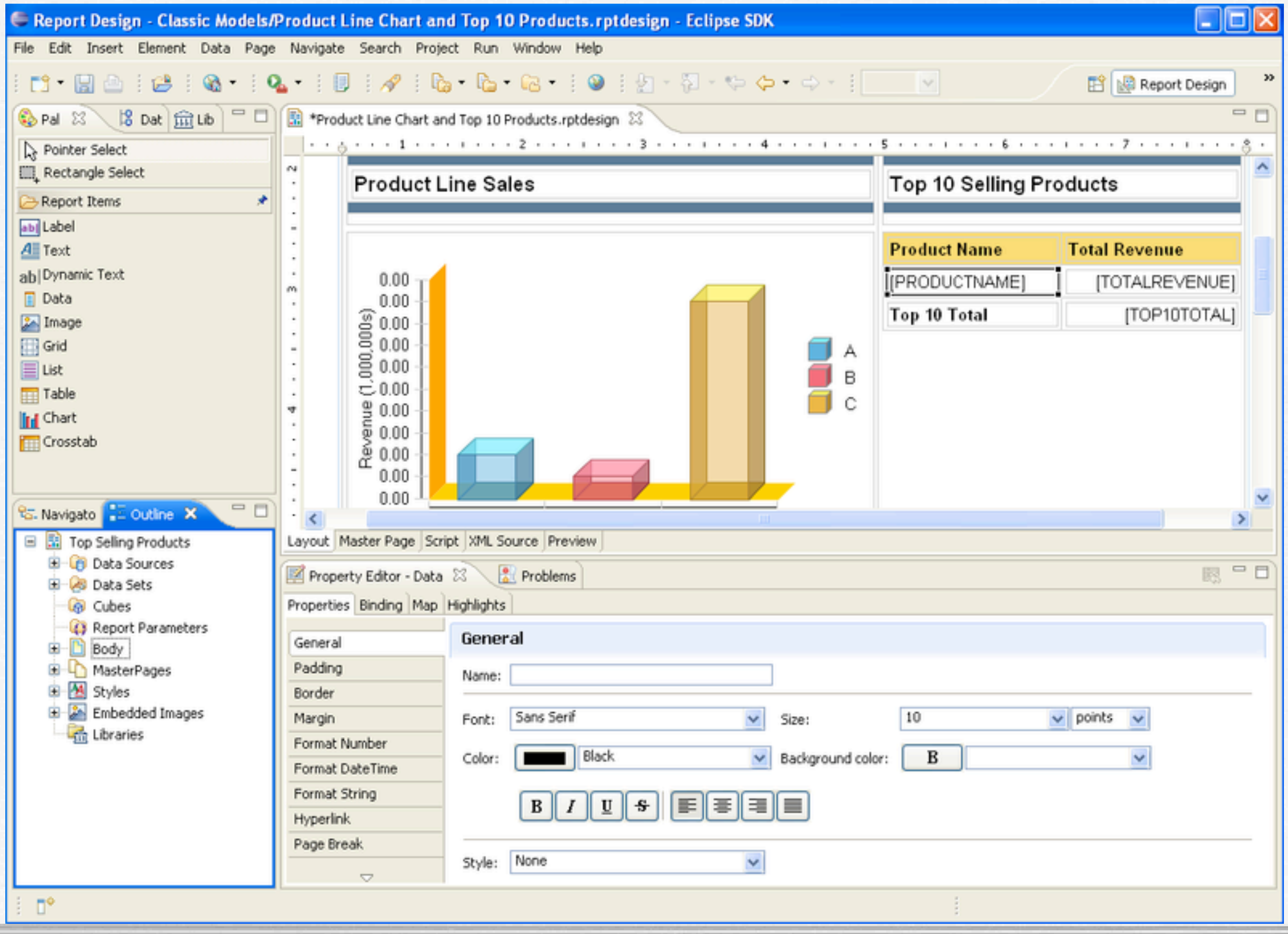
- Input validation framework for PHP based applications
- Developed by skilled hackers (Mario Heiderich - .mario on sla.ckers.org)
- Try their demo with your nasty attack vectors here: http://demo.php-ids.org/

CWE-116: Improper Encoding/Escaping of Output

- Insufficient output encoding is the often-ignored sibling to poor input validation
- Even if input has been filtered, application output could not be safe: it need to be encoded too
- Common examples: HTML/JavaScript injection on web based applications

CWE-116: Example

- **Eclipse BIRT** (reporting system that integrates with Java/JEE applications)



CWE-116: Example

- Credits: *antisnatchor*
  [http://antisnatchor.com/2008/12/18/eclipse-birt-reflected-xss]
- Java Exception stack trace was not HTML-encoded, so we can inject an iframe

```
GET

/birt-viewer/run?__report='"><iframe
%20src=javascript:alert(666)>&r=-703171660 HTTP/1.1

Host: localhost:8780
```

- Our code was executed correctly in the application output

CWE-116: Mitigation

- Always encode Java stack traces (better to don’t show them to prevent Information Leakage)
- Always encode application output, especially if it contains previously user-supplied input
- WebApp firewall and ESAPI/PHPIDS (*you lazy developers :)*)

CWE-209: Error Message Information Leak

- Chatty or debug error messages could disclose important important informations to attackers
- This information is used in the Penetration Testing phase called *“Reconnaissance”*
- Even these little secrets can greatly simplify a more concerted attack that yields much bigger rewards

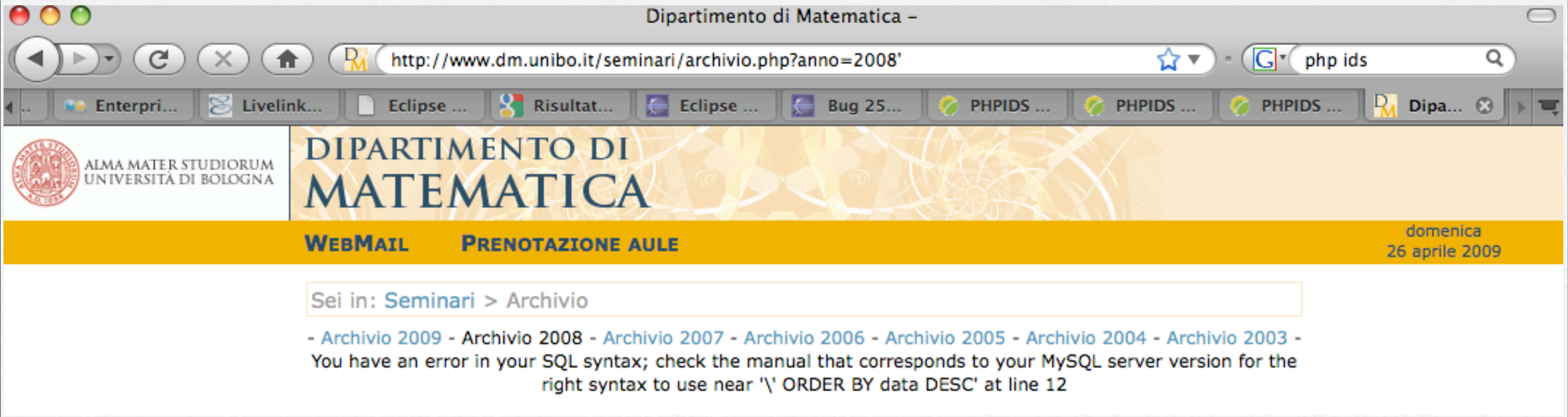
CWE-209: Examples
1. www.dm.unibo.it

- Credits: *antisnatchor*
- MySQL error when forging a malicious request altering the *anno* parameter

```
GET /seminari/archivio.php?anno=2008%27 HTTP/1.1
Host: www.dm.unibo.it
[...]
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: dm=[...]
```

CWE-209: Examples
1. www.dm.unibo.it

- Application response:


Dipartimento di Matematica –
http://www.dm.unibo.it/seminari/archivio.php?anno=2008'
php ids
Enterpri... Livelink... Eclipse ... Risultat... Eclipse ... Bug 25... PHPIDS ... PHPIDS ... PHPIDS ... Dipa...
ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA
DIPARTIMENTO DI MATEMATICA
WEBMAIL PRENOTAZIONE AULE
domenica 26 aprile 2009
Sei in: Seminari > Archivio
- Archivio 2009 - Archivio 2008 - Archivio 2007 - Archivio 2006 - Archivio 2005 - Archivio 2004 - Archivio 2003 -

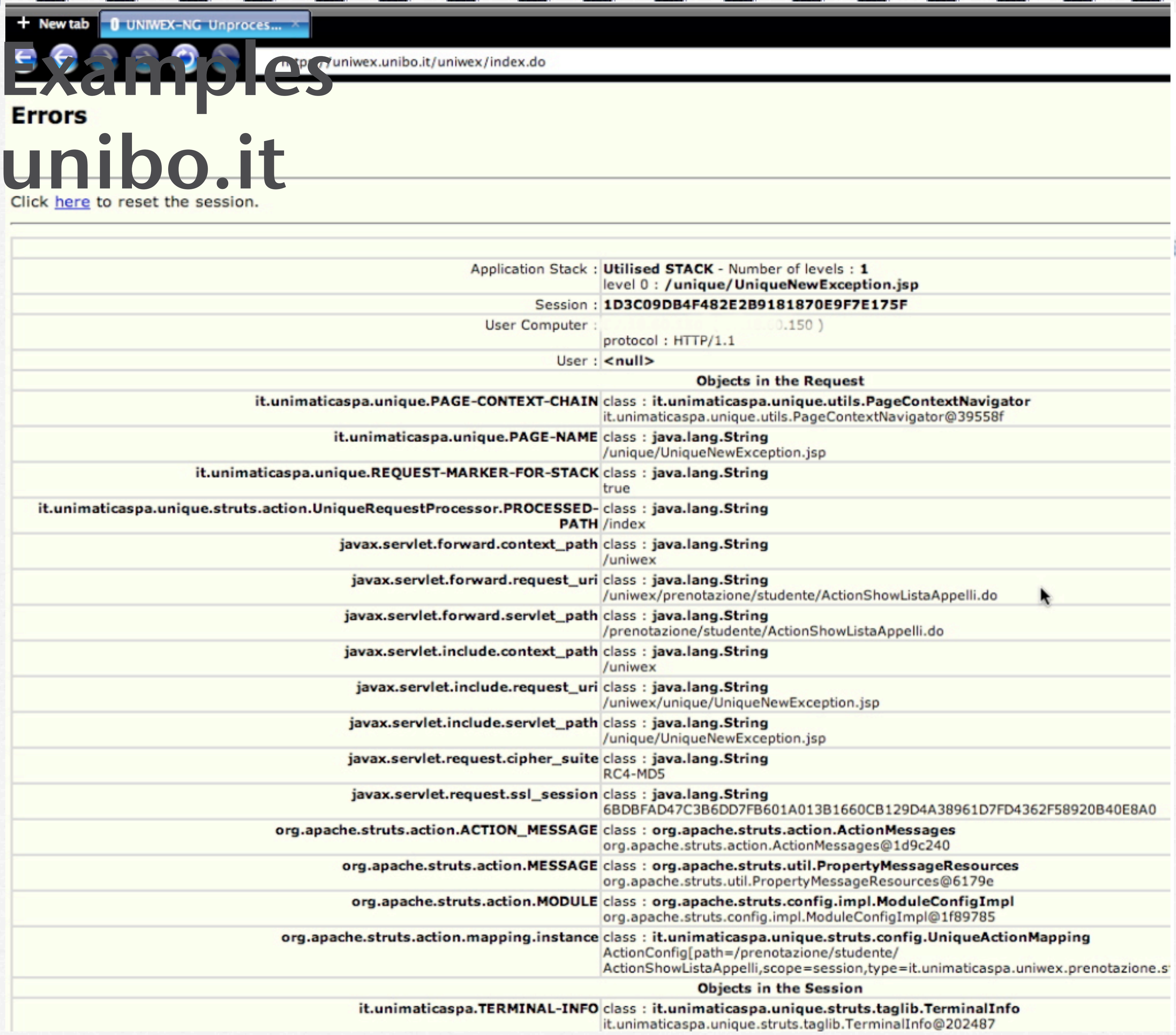
```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the
right syntax to use near '\' ORDER BY data DESC' at line 12
```

- Causing an SQL syntax error we discovered that the DB backend is **MySQL**
- We can now run **more targeted attacks**

CWE-209: Examples
2. uniwex.unibo.it

- Credits: *antisnatchor*
- Session Management was (IS actually) broken and can be manipulated
- If we are the hacker **riding** the victim’s session, and the victim then logout from Uniwex, his session (and ours, because is the same) is invalidated.
- If we invalidate a session and then we try to submit the previously “*invalid*” session token... MAGICALLY ...

CWE-209: Examples
2. uniwex.unibo.it



New tab | UNIWEX-NG Unproces...

https://uniwex.unibo.it/uniwex/index.do

Errors

Click here to reset the session.

Application Stack :	**Utilised STACK** - Number of levels : **1** level 0 : **/unique/UniqueNewException.jsp**
Session :	**1D3C09DB4F482E2B9181870E9F7E175F**
User Computer :	[redacted].150) protocol : HTTP/1.1
User :	**<null>**
	Objects in the Request
it.unimaticaspa.unique.PAGE-CONTEXT-CHAIN	class : **it.unimaticaspa.unique.utils.PageContextNavigator** it.unimaticaspa.unique.utils.PageContextNavigator@39558f
it.unimaticaspa.unique.PAGE-NAME	class : **java.lang.String** /unique/UniqueNewException.jsp
it.unimaticaspa.unique.REQUEST-MARKER-FOR-STACK	class : **java.lang.String** true
it.unimaticaspa.unique.struts.action.UniqueRequestProcessor.PROCESSED-PATH	class : **java.lang.String** /index
javax.servlet.forward.context_path	class : **java.lang.String** /uniwex
javax.servlet.forward.request_uri	class : **java.lang.String** /uniwex/prenotazione/studente/ActionShowListaAppelli.do
javax.servlet.forward.servlet_path	class : **java.lang.String** /prenotazione/studente/ActionShowListaAppelli.do
javax.servlet.include.context_path	class : **java.lang.String** /uniwex
javax.servlet.include.request_uri	class : **java.lang.String** /uniwex/unique/UniqueNewException.jsp
javax.servlet.include.servlet_path	class : **java.lang.String** /unique/UniqueNewException.jsp
javax.servlet.request.cipher_suite	class : **java.lang.String** RC4-MD5
javax.servlet.request.ssl_session	class : **java.lang.String** 6BDBFAD47C3B6DD7FB601A013B1660CB129D4A38961D7FD4362F58920B40E8A0
org.apache.struts.action.ACTION_MESSAGE	class : **org.apache.struts.action.ActionMessages** org.apache.struts.action.ActionMessages@1d9c240
org.apache.struts.action.MESSAGE	class : **org.apache.struts.util.PropertyMessageResources** org.apache.struts.util.PropertyMessageResources@6179e
org.apache.struts.action.MODULE	class : **org.apache.struts.config.impl.ModuleConfigImpl** org.apache.struts.config.impl.ModuleConfigImpl@1f89785

org.apache.struts.action.mapping.instance	class : **it.unimaticaspa.unique.struts.config.UniqueActionMapping**
	ActionConfig[path=/prenotazione/studente/
	ActionShowListaAppelli,scope=session,type=it.unimaticaspa.uniwex.prenotazione.s

	Objects in the Session
it.unimaticaspa.TERMINAL-INFO	class : **it.unimaticaspa.unique.struts.taglib.TerminalInfo** it.unimaticaspa.unique.struts.taglib.TerminalInfo@202487

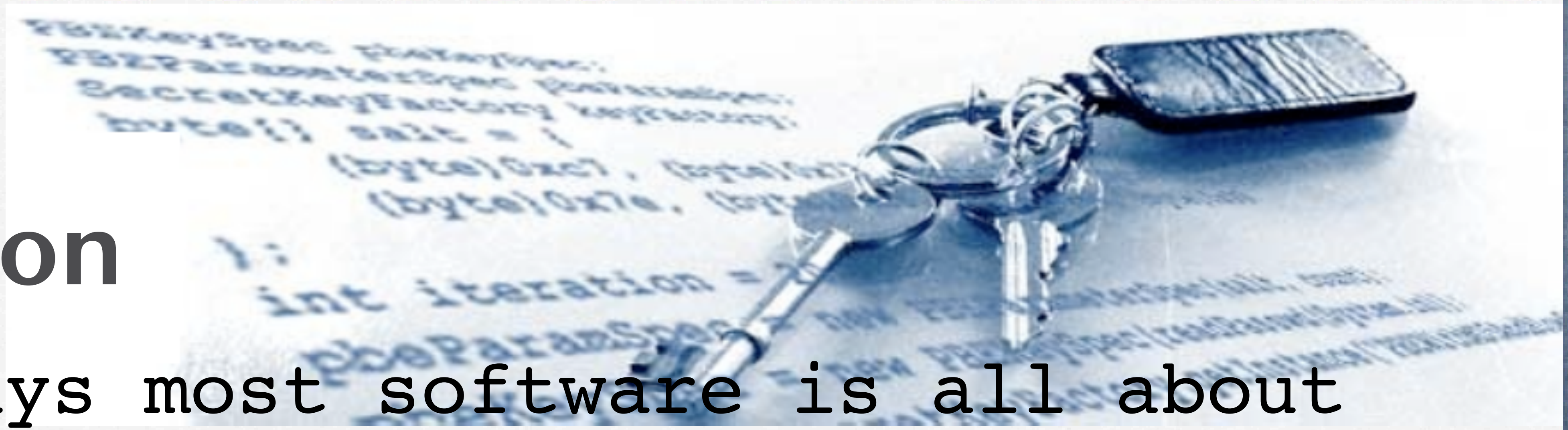
CWE-209: Examples
2. uniwex.unibo.it

- The JSP page `/unique/UniqueNewException.jsp` is clearly leaved there for debug purposes
- It shouldn't be there in production!!!
- This revealed us that Tomcat is used as Application Server, and we've also obtained the specific version of a few frameworks on which the application was built:

```
/home/unimatica/uniwex/uniwexng-4.4.0/WEB-INF/lib/struts-1.1.jar

/home/unimatica/uniwex/uniwexng-4.4.0/WEB-INF/lib/myfaces-api-1.1.4.jar
```

CWE-89: SQL Injection



- These days most software is all about the data and how it can be served to maximize user and business needs
- The most common storage solution is a Relation Database(Oracle, MySQL, Postgres, MS-SQL, Sybase)
- If attackers can influence the SQL that you use to communicate with your database, then they can do nasty things for fun and profit

CWE-89: SQL Injection

- Discovering which web application parameters/cookie/headers are querying the DB, we can test if input is properly escaped or not
- The previous example on www.dm.unibo.it demonstrates that input is not being escaped at all
- After we discovered the SQL injection we can fire-up our favorite injection tool to retrieve useful informations

CWE-89:Example 1. www.dm.unibo.it

- Credits: *antisnatchor*
- Confirmed unescaped numeric injection on GET parameter “*anno*”
- We were able to obtain details about the application stack:Apache 2.2.3, PHP 5.2.0, MySQL >= 5.0
- For demonstration we retrieved the exact name of the database name to which the web app is bounded: *dipartimento*

CWE-89:Example
1. www.virtus.it

- Credits: *antisnatchor*
- Confirmed unescaped numeric injection on GET parameter "*ID*" (SPNewsDettaglio.asp)
- We were able to obtain details about the application stack:Microsoft IIS 6, ASP and SQL Server 2000
- We retrieved the exact name of the database name to which the web app is bounded: *ServizioNews (and a few tables too)*

CWE-89: Mitigation

- Implement a validation framework (previously discussed) to protect your application
- Use stored procedures
- Hibernate on JEE, NHibernate on .NET
- DB specific: Oracle DBMS_ASSERT directive, MySQL real_escape_string() function
- Use a whitelist approach, permitting only “known good input”

CWE-89: Dangers

- As you can see SQL injection can be devastating for the integrity of your data
- Data loss is probably the most negative consequence for an Enterprise
- If the web application is storing web page content inside the DB, we can **deface** the site too

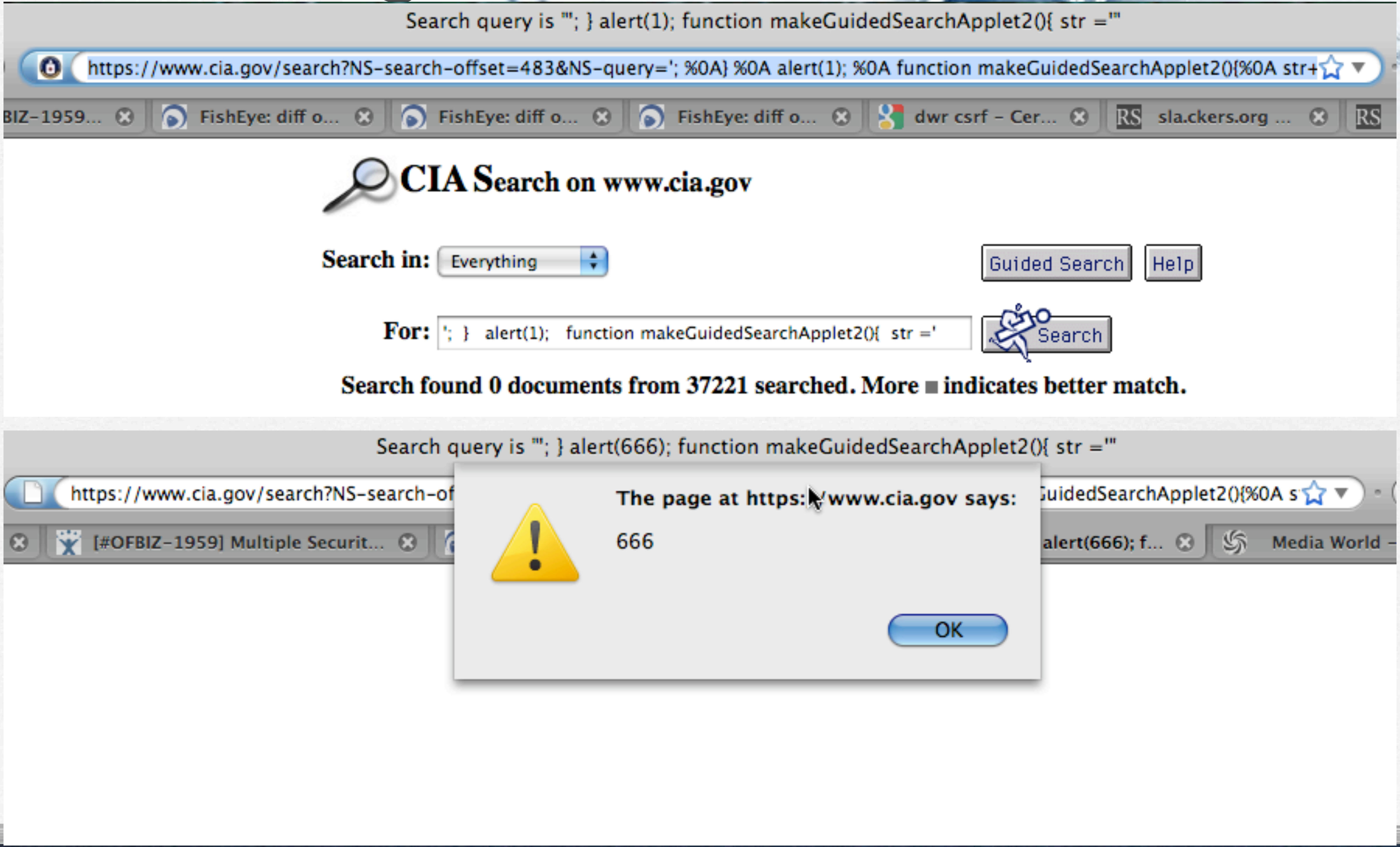
CWE-79: The Plague of Cross Site Scripting

- We can inject JavaScript,HTML,VBscript or other browser-executable content into a pages generated by the application
- The page is then accessed by other users, whose browsers execute that malicious script as if it came from the legitimate user (the victim)

CWE-79: Examples
1. www.cia.gov

- Credits: *PaPPY*
- *search* URI: *NS-query* parameter is not properly escaping malicious input, leading to reflected XSS
- https://www.cia.gov/search?NS-search-offset=483&***NS-query=%27;%20%0A}%20%0A%20alert(666);%20%0A%20function%20makeGuidedSearchApplet2(){%0A%20str+=%27***&NS-search-type=NS-boolean-query&NS-max-records=20&NS-collection=Everything&x=0&y=0&NS-search-page=results&

CWE-79: Examples
1. www.cia.gov



Search query is "'; } alert(1); function makeGuidedSearchApplet2(){ str ='"

```
https://www.cia.gov/search?NS-search-offset=483&NS-query='; %0A} %0A alert(1); %0A function makeGuidedSearchApplet2(){%0A str+
```

CIA Search on www.cia.gov

Search in: Everything

For: '; } alert(1); function makeGuidedSearchApplet2(){ str ='

Search found 0 documents from 37221 searched. More ■ indicates better match.

Search query is "'; } alert(666); function makeGuidedSearchApplet2(){ str ='"

```
https://www.cia.gov/search?NS-search-of ... uidedSearchApplet2(){%0A s
```

The page at https://www.cia.gov says:

666

CWE-79: Examples
2. compraonline.mediaworld.it

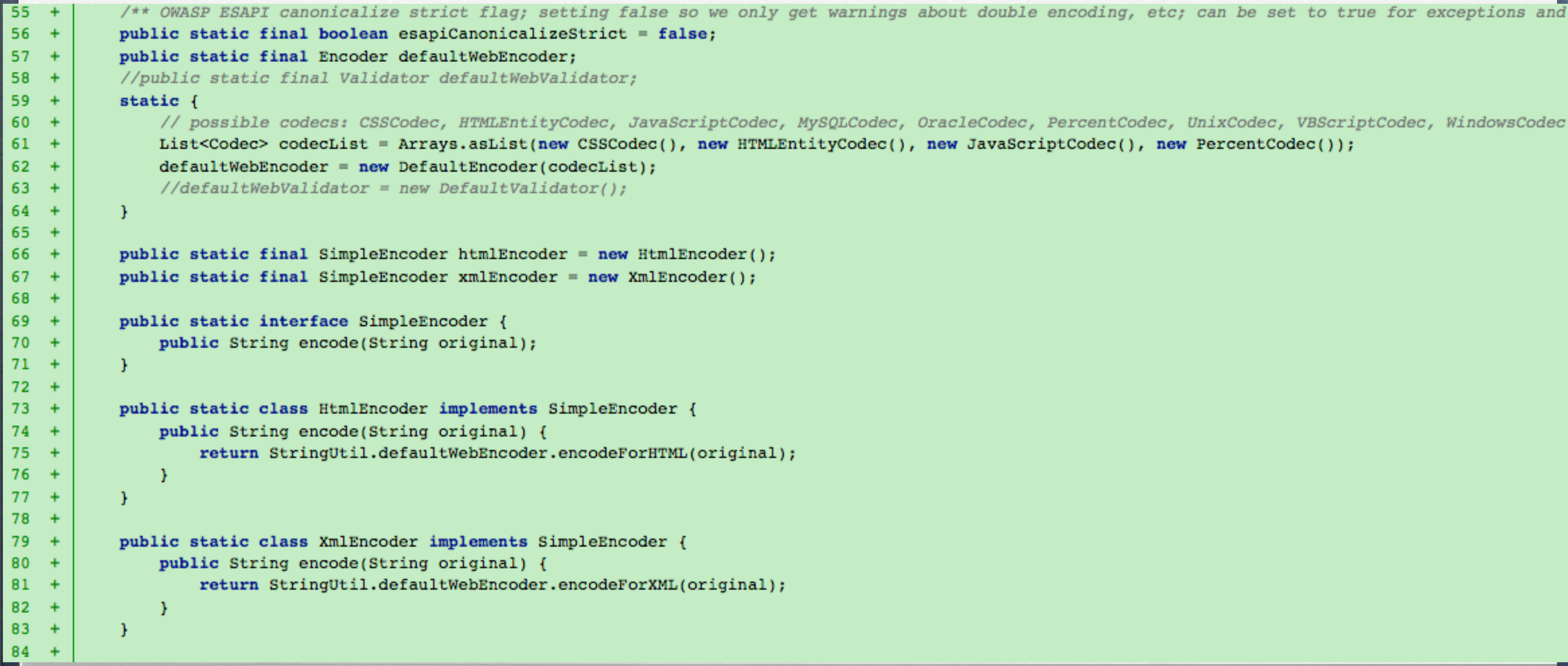
- Credits: *antisnatchor*
- *search* URI: *NS-query* parameter is not properly escaping malicious input, leading to reflected XSS
- https://www.cia.gov/search?NS-search-offset=483&***NS-query=%27;%20%0A}%20%0A%20alert(666);%20%0A%20function%20makeGuidedSearchApplet2(){%0A%20str+=%27***&NS-search-type=NS-boolean-query&NS-max-records=20&NS-collection=Everything&x=0&y=0&NS-search-page=results&

CWE-79: Mitigation

- A real world case example: **Apache OFBiz** implementation of ESAPI toolkit.
- After my JIRA issue they started to take really care of security (I’m glad to)
- See http://fisheye6.atlassian.com/changelog/ofbiz?cs=746409 and http://antisnatchor.com/2008/12/11/apache-ofbiz-multiple-security-vulnerabilities

CWE-79: Mitigation

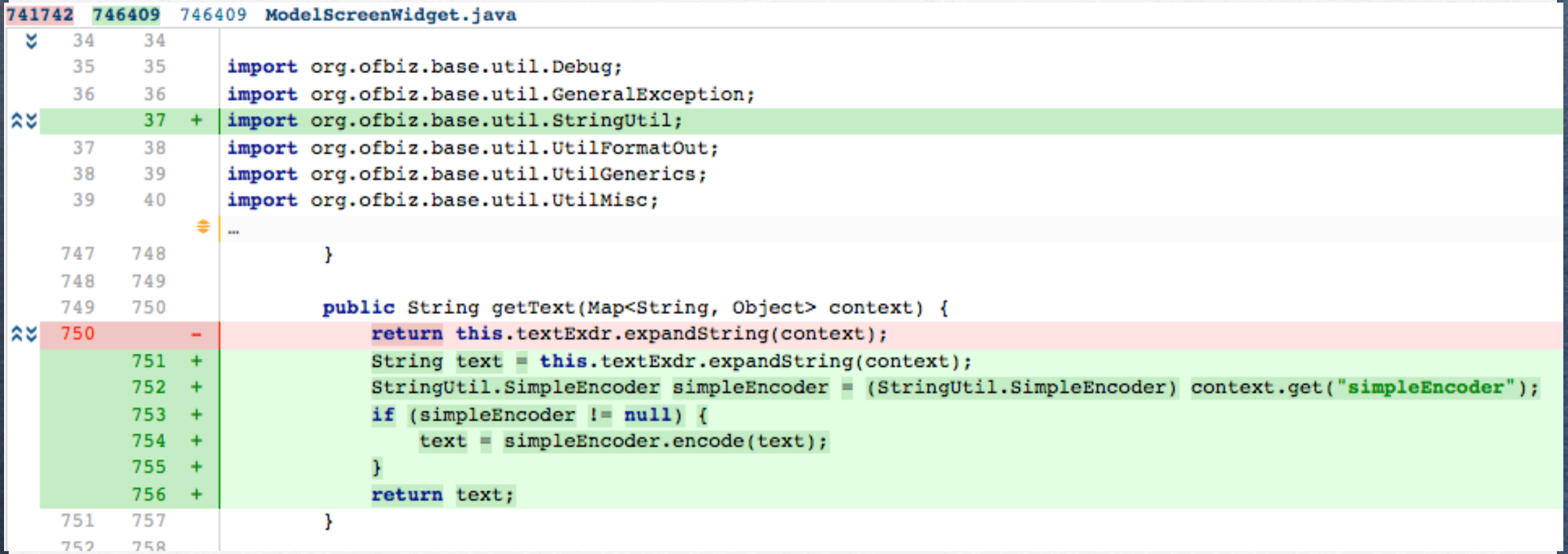
- The changes of StringUtil.java class:



```
+    /** OWASP ESAPI canonicalize strict flag; setting false so we only get warnings about double encoding, etc; can be set to true for exceptions and
+    public static final boolean esapiCanonicalizeStrict = false;
+    public static final Encoder defaultWebEncoder;
+    //public static final Validator defaultWebValidator;
+    static {
+        // possible codecs: CSSCodec, HTMLEntityCodec, JavaScriptCodec, MySQLCodec, OracleCodec, PercentCodec, UnixCodec, VBScriptCodec, WindowsCodec
+        List<Codec> codecList = Arrays.asList(new CSSCodec(), new HTMLEntityCodec(), new JavaScriptCodec(), new PercentCodec());
+        defaultWebEncoder = new DefaultEncoder(codecList);
+        //defaultWebValidator = new DefaultValidator();
+    }
+
+    public static final SimpleEncoder htmlEncoder = new HtmlEncoder();
+    public static final SimpleEncoder xmlEncoder = new XmlEncoder();
+
+    public static interface SimpleEncoder {
+        public String encode(String original);
+    }
+
+    public static class HtmlEncoder implements SimpleEncoder {
+        public String encode(String original) {
+            return StringUtil.defaultWebEncoder.encodeForHTML(original);
+        }
+    }
+
+    public static class XmlEncoder implements SimpleEncoder {
+        public String encode(String original) {
+            return StringUtil.defaultWebEncoder.encodeForXML(original);
+        }
+    }
+
```

CWE-79: Mitigation

- The changes of ModelScreenWidget.java:



741742 746409 746409 ModelScreenWidget.java

```diff
 
 import org.ofbiz.base.util.Debug;
 import org.ofbiz.base.util.GeneralException;
+import org.ofbiz.base.util.StringUtil;
 import org.ofbiz.base.util.UtilFormatOut;
 import org.ofbiz.base.util.UtilGenerics;
 import org.ofbiz.base.util.UtilMisc;
 …
         }
 
         public String getText(Map<String, Object> context) {
-            return this.textExdr.expandString(context);
+            String text = this.textExdr.expandString(context);
+            StringUtil.SimpleEncoder simpleEncoder = (StringUtil.SimpleEncoder) context.get("simpleEncoder");
+            if (simpleEncoder != null) {
+                text = simpleEncoder.encode(text);
+            }
+            return text;
         }
```

CWE-79: Mitigation

- Validate every parameter/cookie/header/input that can be manipulated by a potential attacker and then displayed on the page
- **Do not create your own filters:** you'll probably miss some attack vectors or encodings
- Use well known Encoding/Validation frameworks such as ESAPI,PHPIDS,Microsoft Anti-XSS (yes, Microsoft, don't laugh :))

CWE-352: Cross Site Request Forgery

- It exploits the trust that a website has for the currently authenticated user and executes unwanted actions on a web application on his behalf
- Once the request gets to the application, it looks as if it came from the user, not the attacker
- If the victim has admin privileges on the application: GAME OVER

CWE-352: XSRF Concrete Consequences

- Performing illegal actions such as using victim's shopping cart, executing stock trades
- Changing DNS settings of home routers (thanks pdp & GNUCITIZEN)
- Performing a Denial Of Service attack on the application
- Combining it with XSS to build WORMS

CWE-352: XSRF Concrete Consequences

1. Find a page with a lost-password form inside and find out which fields would be updated

2. **Trick the administrator** to load a hacker page with a malicious request on it that submits a new email

3. Administrator's e-mail is now changed to the email submitted by hacker

4. A hacker performs a lost-password request and **receives a new password**

CWE-352: XSRF
Who has been vulnerable?

- ING direct [We discovered CSRF vulnerabilities in ING's site that allowed an attacker to open additional accounts on behalf of a user and transfer funds from a user's account to the attacker's account.]
- Youtube
- New York Times
- Gmail [http://directwebremoting.org/blog/joe/2007/01/01/csrf_attacks_or_how_to_avoid_exposing_your_gmail_contacts.html]

CWE-352: XSRF Example

- A simple practical attack:

  `http://x.x.x.x/account/doTransfer?from=666&to=667`

  where 666 is a potential victim account and 667 the attacker one.

  Tricking the victim to load that URL will transfer money from one account to another one.

CWE-352: XSRF
1. Apache OFBiz

- Read my advisory here: https://issues.apache.org/jira/browse/OFBIZ-1959
- We can create a malicious form that will add a product (eventually with some JS inside) to the Catalog
- If the victim is already authenticated she will not even realize what she did

CWE-352: XSRF
1. Apache OFBiz

```
<form method="POST" id="xsrf" name="xsrf"

action="https://127.0.0.1:8443/catalog/control/
createProduct">

<input type=hidden name="isCreate" value="true">

<input type=hidden name="productId" value="hack02">

<input type=hidden name="productTypeId" value="DIGITAL_GOOD">

<input type=hidden name="internalName"
value="hack02<script>alert(document.cookie)</script>">

</form>

<script>document.xsrf.submit(); </script>
```

CWE-352: XSRF Mitigation

- Add a unique randomly-generated token to each request (maybe as an hidden form value): this n bit token is changed for every request and is verified by the application

```
<input id="fkey" name="fkey"
type="hidden" value="df8652852f139" />
```

CWE-352: XSRF Mitigation

- Use a secure framework such as ESAPI to add random token to your requests
- Implement AJAX functionalities with secure libraries such as DWR-2.0 (Direct Web Remoting) that automatically prevent XSRF

Thanks for your attention!

brought to you by Michele “antisnatchor” Orru’

Computer System Security course lead by Prof. Ozalp Babaoglu

5 May 2009