



# 1 Review and Hack Workshop

---

Is your web application secure? How can you tell? This workshop teaches you to review the source code of your web application for potential vulnerabilities and how to prove their existence. To practice we use the Bad-Cheesr web application which comes as source code in a Maven project. The Bad-Cheesr is application full of dubious code for you to find and exploit. When you've found all its vulnerabilities (and maybe some more) you can try your new skills on your own web application!

Duration	2 Hours
Audience	Web Application Developers of all levels.
Prerequisites	Laptop, Internet, IDE, JDK7+ and Maven 3+, Browser plugin to <a href="#">modify headers</a> . Or the prefab <a href="#">Virtual Box</a> image.
Source Code	<a href="http://www.ctrl-alt-dev.nl/workshops/rnh/resources/rnh/bad-cheesr-v1.4.zip">http://www.ctrl-alt-dev.nl/workshops/rnh/resources/rnh/bad-cheesr-v1.4.zip</a>
Instructions	<a href="http://www.ctrl-alt-dev.nl/workshops/rnh/resources/rnh/review-and-hack-workshop-v1.3.pdf">http://www.ctrl-alt-dev.nl/workshops/rnh/resources/rnh/review-and-hack-workshop-v1.3.pdf</a>

This workshop consists of 7 labs, of which the first 4 are the main content. Labs 5 to 7 go deeper and explore related subjects.

- [Review and Hack Workshop](#)
- [Setup](#)
  - [Installation](#)
    - [Option 1: Virtual Box](#)
    - [Option 2: Internet](#)
  - [Setup check and IDE integration](#)
  - [Running the Web application](#)
- [Reviewing for Vulnerabilities](#)
  - [Conceptual Model](#)
  - [Guidelines and the Conceptual Model](#)
  - [Introducing Bad-Cheesr](#)
- [Lab 1: Bad Queries, Secret Sauce](#)
  - [Theory](#)
  - [Reality](#)
- [Lab 2: Chaos Orders the Vouchers.](#)
  - [Theory](#)
    - [Escaping HTML](#)
    - [Proving a lack of Escaping](#)
    - [Including external HTML](#)
  - [Reality](#)
- [Lab 3: Input, some is more valid than others.](#)
  - [Theory](#)
    - [Normalization](#)
    - [Validation](#)
    - [Filtering](#)
    - [Authentication and Authorization](#)
  - [Reality](#)



- Lab 4: Knowledge is in the Eye of the Beholder
  - Theory
  - Reality
- Lab 5: Beyond the Guideline
  - Theory
    - ReDos
    - Xml eXternal Entity processing
    - Object Deserialization Vulnerabilities
  - Reality
- Lab 6: Find (security) Bugs?
- Lab 7: Find Libraries with Security Issues
- The End

## 2 Setup

Depending on the (non) availability internet connection you have the following options to retrieve and install the workshop material.

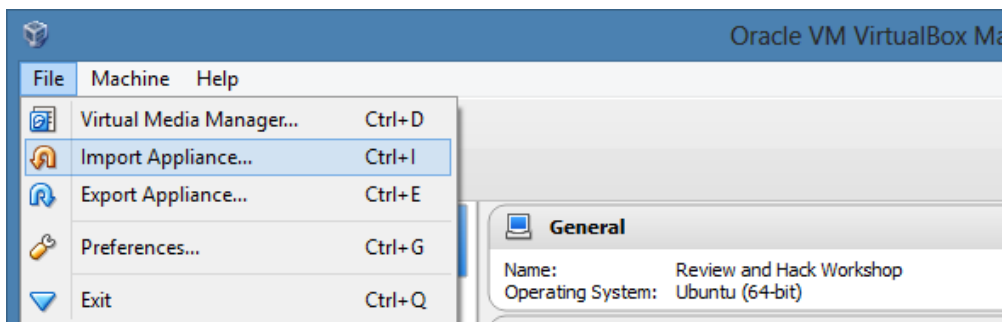
1. No internet, use Virtual Box with the provided memory stick to install the virtual box appliance which contains a fully working setup with no internet required.
2. Internet: download the source code using the link above, or use the memory stick

✔ Please return the memory sticks after use. Thanks!

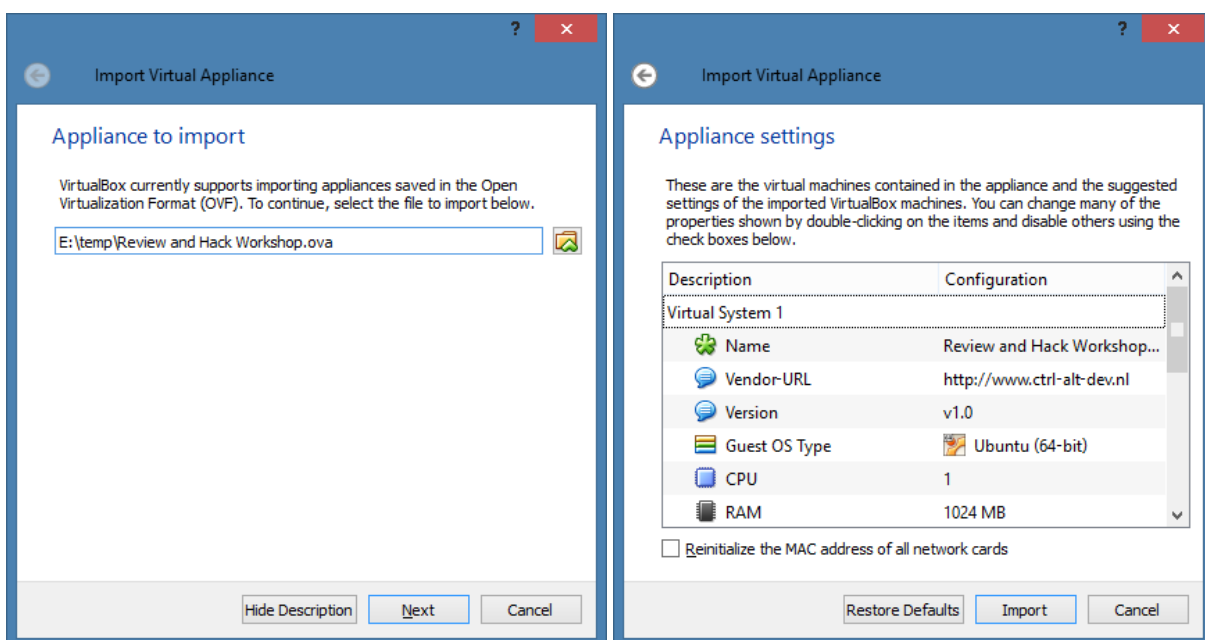
### 2.1 Installation

#### 2.1.1 Option 1: Virtual Box

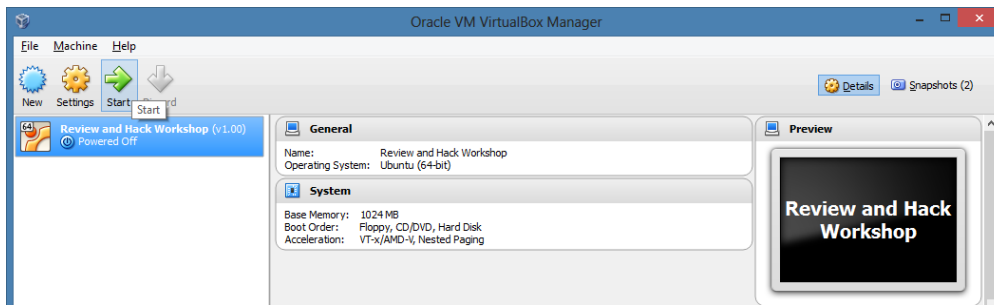
Start the Virtual Box Manager and select File.. Import Appliance from the menu.



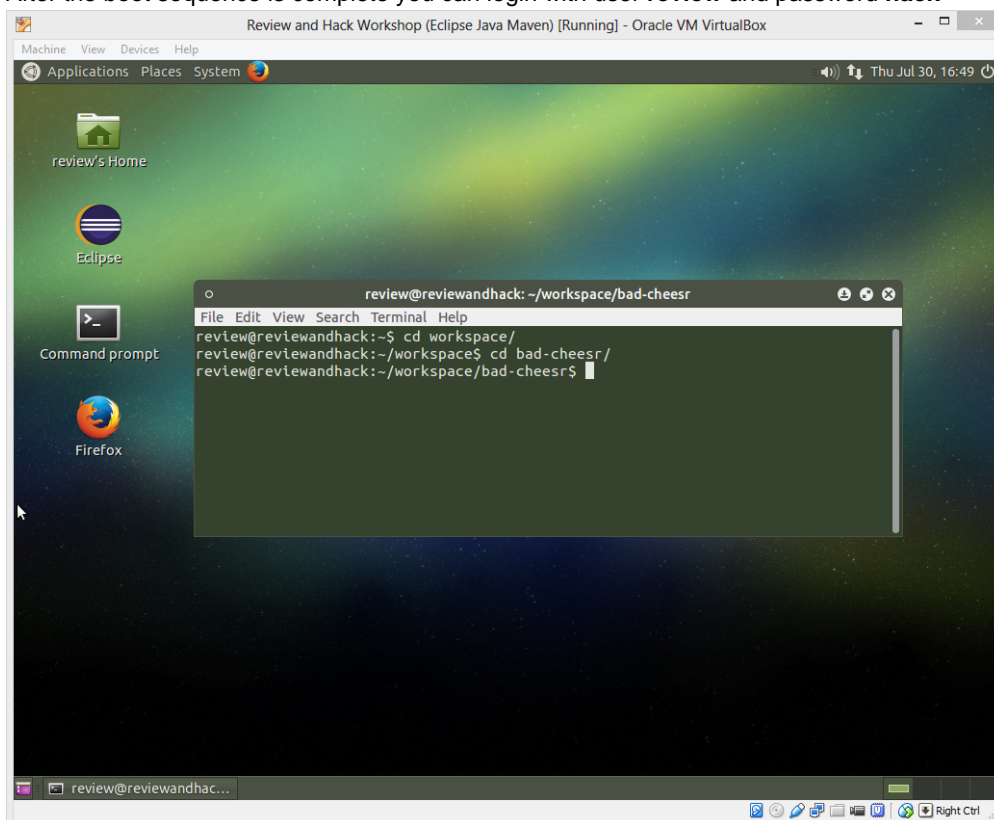
Then, select the `Review and Hack Workshop.ova` file for import. Press Next and Import and wait patiently while the appliance is imported.



When complete (this may take a few minutes depending on the speed of your hard disk), you can start the virtual machine from the user interface.



After the boot sequence is complete you can login with user **review** and password **hack**



You will find the source code of the application in `~/workspace/bad-cheesr`

Well done! You can proceed with the Setup check and IDE integration chapter.

### 2.1.2 Option 2: Internet

Using the link in the table on top of this document, or from the memory stick, unzip the `bad-cheesr.zip` file into a working folder of your choice. Also, if you haven't done so already, please install the [Modify Headers](#) plugin. The plugin is for Firefox, but similar plugins exist for other browsers.

## 2.2 Setup check and IDE integration

You should check the existence of the [Java Development Kit](#) and [Apache Maven](#) runtime on your laptop. Open a command prompt and check. Output should be similar to:



```
> java -version
java version "1.8.0_25"
Java(TM) SE Runtime Environment (build 1.8.0_25-b18)
Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode)
> mvn -version
Apache Maven 3.0.4 (r1232337; 2012-01-17 09:44:56+0100)
Maven home: D:\java\app\apache-maven-3.0.4
Java version: 1.8.0_25, vendor: Oracle Corporation
Java home: d:\java\jdk\64\jdk1.8_25\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 8", version: "6.2", arch: "amd64", family: "dos"
```

Instruct your IDE to import the project. If you're using eclipse you may need to use `mvn eclipse:eclipse` command to generate the IDE configuration.

If all is well you should now be able to comfortably browse the source code of the Bad Cheesr web application.

## 2.3 Running the Web application

While your IDE may have built in support to run maven goals, its also possible to run the web application from the command line:

```
> mvn jetty:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building bad-cheesr 1.0-SNAPSHOT
[INFO] -----
.. etc etc etc ..
2015-05-31 14:27:24.705::INFO: Started SelectChannelConnector@0.0.0.0:8080
[INFO] Started Jetty Server
```

When the command runs successfully you should be able to open the web application in your browser using <http://localhost:8080/bad-cheesr/>

Some of you may recognize this application. Its an extended, modified and corrupted version of the Cheesr sample application from the book 'Wicket in Action'. (please note: Apache Wicket is a really nice and secure framework!)

If you wish to debug the web application (which may be useful to inspect the internal state when reproducing an issue) you can use the `mvnDebug jetty:run` command and connect your IDE to the maven process.



You can run maven in offline mode using the `-o` option.

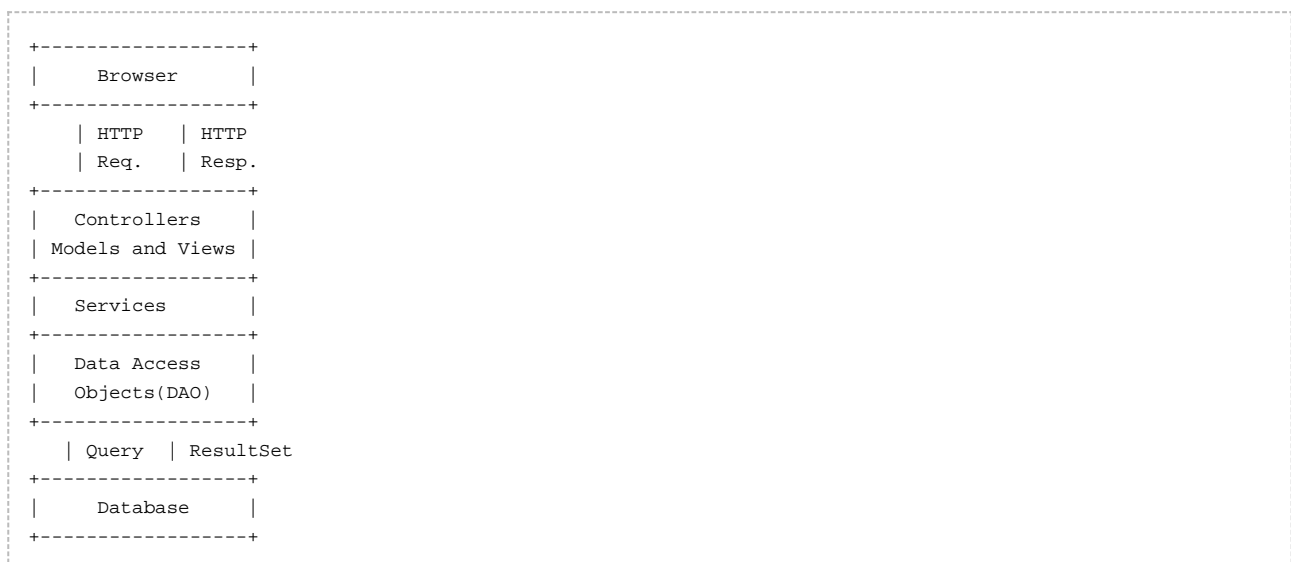
## 3 Reviewing for Vulnerabilities

This workshop is based on the '[ICT-Beveiligingsrichtlijn voor webapplicaties](#)' published by the NCSC ('National Cyber Security Centre'). It is a mandatory (but frequently overlooked) guideline for many Dutch governmental institutions. *And* more importantly: it actually recommends doing code reviews as a way of finding security issues.

The document is broad in scope and captures everything security related like procedures and paperwork, network and platform security, authentication and monitoring. For developers the sections on web applications and web protocols are relevant and the workshop will focus on those. I will mention the section numbers

### 3.1 Conceptual Model

Before we start reviewing, lets start with a conceptual model of a traditional web application (which the bad-cheesr application is) in order to be able to pinpoint where certain things can go wrong.



Typically the browser performs HTTP requests, which, after processing by the application produce HTTP responses. First entry point is the Controller which determines what the intent is of the request (display some page, take some action). Typically a call to the controller will delegate to the service layer to retrieve the data or perform the action within the rules of the application. Actual database interactions are performed by the Data Access Objects (DAO's) using Queries and Result Sets. The outcome of the service calls will be put into a Model and rendered into a View.

- ✔ Modern web applications use Javascript frontends and interface with the application using a Rest API. From the server point of view this does not really change the conceptual model except that the Model View Controller layer becomes more simple as the user interface state is moved to the browser. However the Javascript code must also be reviewed (especially when rendering output) so the amount of work increases.

- ✔ The Bad-Cheesr Web Application uses Wicket as user interface framework which encapsulates the model view controller layer as a series of Java User Interface Widgets.



## 3.2 Guidelines and the Conceptual Model

The NCSC Security Guidelines focus on three specific points in the conceptual model: Its input (the HTTP Requests), its output (the HTTP Responses) and the Queries to the database.

Guideline		Area	Summary	Where to look in the code	Lab
U/WA.07	1	Queries	Build queries using fixed strings.	Data Access Objects	1
U/WA.07	2	Queries	Use parametrized queries to prevent <a href="#">SQLi</a>	Data Access Objects	1
U/WA.04	1	Response	Escape user data in output to prevent <a href="#">XSS</a>	Views	2
U/WA.03	2	Response	White list or don't use <a href="#">dynamic includes</a>	Views	2
U/WA.03	3	Request	Normalize request input	Controller	3
U/WA.03	1	Request	Validate request input	Controller	3
U/PW.02	2	Request	User must be identified and permissions checked	Controller, Service	3
U/WA.03	4	Request	Don't assume that validated requests are correct *	Controller, Service	3

\* your application may render forms with hidden parameters or render urls with ids. Don't assume these won't be experimented with.

Then there are the other guidelines which cannot easily be reviewed against (but most of them are generally easy observable from a browser):

Guideline		Area	Summary	Where to look in the Browser	Lab
U/PW.02	4	Headers	Only send HTTP headers that are necessary	Developer tools, requests tab.	4
U/PW.02	5	Headers	Only send minimal data in HTTP headers	Developer tools, requests tab.	4
U/PW.02	6	Error Pages	Error pages must not include technical information	View HTML source, JSON Data.	4
U/WA.06	1	Resources	Comments must be stripped from resources.	View source(s)	4
U/PW.02	3	Server config	Server only supports needed HTTP methods	n/a	-
U/PW.03	2	Server config	Directory listings are disabled	Open css or images basepath.	4
C.05		Action	Perform a code review	n/a	all
C.05		Action	Perform an automated black-box security scan	n/a	-
U/PW.03	3	Server config	Session cookies must have HTTP Only and Secure flags	Developer tools, cookies tab.	4

### 3.3 Introducing Bad-Cheesr

Bad-Cheesr is a simple web-shop for cheeses. It allows anyone to browse, search for and fill up a cart with various cheeses, check out and submit a billing address, and optionally a coupon code for a special discount.

On the administrative side, open orders may be browsed, searched, opened and fulfilled. The administrative user is authenticated by a central system that puts additional headers on the request. In this case if the current user is the administrator it has the header `Role: ADMIN` set. Only the administrator may access the order pages!

The screenshot shows the Bad-Cheesr web application interface. At the top, there is a banner image of cheese with the text "Cheesr" and "Making cheese taste bad". A search bar is located below the banner. The main content area is divided into two columns. The left column lists cheese products: "Gouda" and "Edam". Each product has a description, a "Details" link, and an "Add to cart" button. The right column, titled "Your Selection", lists the items currently in the cart: "Maasdam" (€2.35), "Brie" (€3.15), and "Buxton Blue" (€0.99), each with a "remove" link. A "Total" of €6.49 is shown, along with a "Check out" button. A "admin" link is visible in the top right corner of the banner area.

**Gouda**

Gouda is a yellowish Dutch cheese named after the city of Gouda.

[Details](#)

€1.65 [Add to cart](#)

**Edam**

Edam (Dutch Edammer) is a Dutch cheese named after the town of Edam.

[Details](#)

€1.05 [Add to cart](#)

**Your Selection**

Maasdam	€2.35	<a href="#">remove</a>
Brie	€3.15	<a href="#">remove</a>
Buxton Blue	€0.99	<a href="#">remove</a>
<b>Total</b>	€6.49	

[Check out](#)

## 4 Lab 1: Bad Queries, Secret Sauce

### 4.1 Theory

If user input is appended as a string value into a database query it becomes possible to alter the meaning query, returning different results or even gaining administrative access to the database. This is called SQL Injection. Most API's such as JDBC and JPA allow the use of parametrized queries so inserting parameters by appending strings is not necessary. Consider the following example of a badly constructed query:

#### JPA: Bad use of parameters

```
em.createQuery("from Wine w where w.name = '" + name + "'", Wine.class).getResultList();
```

If the name value contains a single quote (') it will unbalance quotes in the query causing it to fail. With a little bit of creativity you can also make it return all records. The proper way to code a parameter into a query is in the following example:

#### JPA : Correct use of parameters

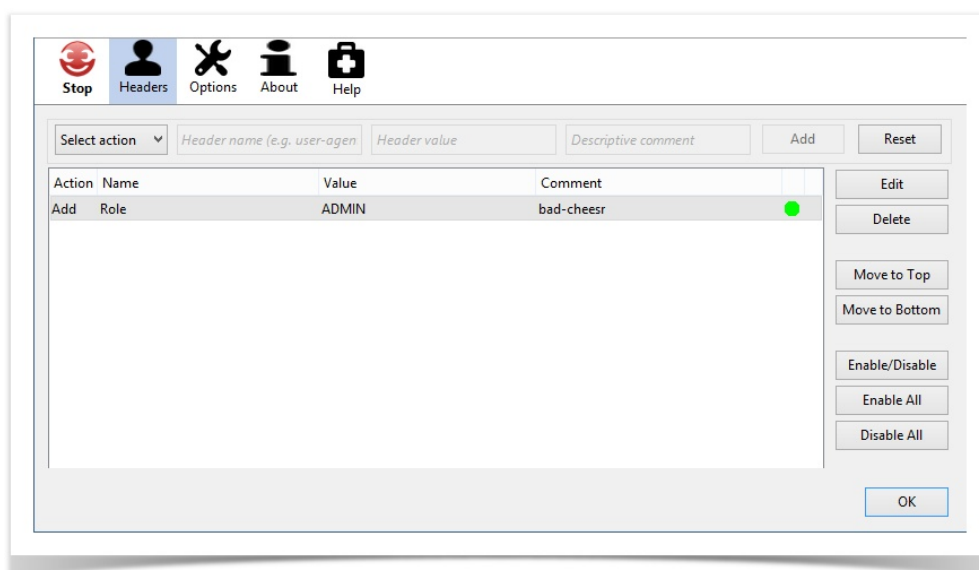
```
em.createQuery("from Wine w where w.name = :name", Wine.class).setParameter("name", name).getResultList();
```

### 4.2 Reality

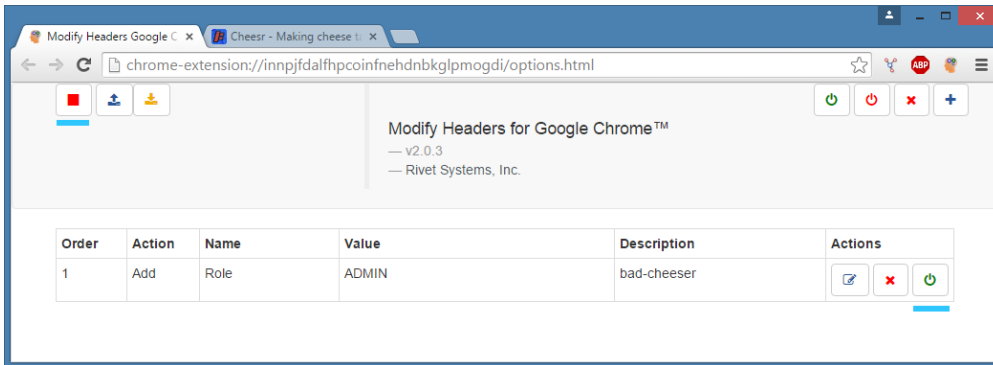
The goal of this lab is to:

- spot the incorrect use of database queries with parameters (U/WA.07) in the source code.
- find an input that reveals the Secret Cheese in the cheeses list.
- find an input that reveals fulfilled orders in the order list.

The cheeses list page is [accessible by anyone](#), provided the application is running. The [orders page](#) is only accessible if you have the header `Role: ADMIN` set. This is done easily using in Firefox with the [Modify Headers](#) plugin.



There is also a Modify Headers plugin available for Chrome, which is slightly less intuitive. After adding the rule you must first press the start button (top left) and then enable the rule (right).



Some tips to guide you on your way:

- In what layer of the application do database queries reside?
- What kind of input may break out of a string literal in the query?
- How can you creatively use the input to make the query return all the records?
- Have a look at the [OWASP testing Guide for SQL Injection](#).

Please document your findings in the table below:

Issue	Source file and Line number	Demonstration Steps .....	Points
U/WA. 07 (sqli)			5
U/WA. 07 (sqli)			5



## 5 Lab 2: Chaos Orders the Vouchers.

### 5.1 Theory

This lab will focus on the output of the application. Typically it will produce HTML pages for the user to interact with by means of a templating technology or web framework (like JSP, Velocity, JSF, Wicket and AngularJS).

#### 5.1.1 Escaping HTML

However when rendering content that may have been provided by the user or a 3rd party you must take care that no unwanted markup is contained in that content, breaking the page layout or even including malicious JavaScript (called *Cross Site Scripting* or XSS). Typically all external content must be escaped or sanitized using a library like [JSoup](#). Most template frameworks provide escaping options. Some even apply escaping as a default to external values (such as Wicket and AngularJS).

In the following JSP example the value is not escaped, allowing arbitrary HTML into the name of the wine.

##### JSP: No escaping

```
<h3>${wine.name}</h3>
```

If the value of `wine.name` would be `'</h3>Blah!'` it would break the page layout rendering `Blah!` as normal text. The correct way would be to use `c:out`

```
<h3><c:out value="${wine.name}" /></h3>
```

In this case the HTML special characters would be escaped as HTML entities (eg. `'&lt;/h3&gt;Blah!'`). Escaping HTML special characters works nicely as long as you're rendering text. However the escape rules are different depending on the context you're in. For instance when rendering a tag attribute or a HTML comment different characters must be escaped. Read this [blog post](#) for a more detail.

Correct escaping is hard work. Its best to use a templating framework that supports it out of the box (or better: applies escaping by default).

In the case of the Bad-Cheesr application, Wicket applies correct escaping by default but it can be turned off by setting `c.setEscapeModelStrings(false)`; Another common pitfall in Wicket applications is forgetting to escape user generated input when modifying the html directly, for instance when using a `SimpleAttributeModifier`.

#### 5.1.2 Proving a lack of Escaping

If you think you've found a user value that is not properly escaped, you need to look at the surrounding html and think up a character sequence that will break out of the current tag or attribute so that you can include your own markup or events. Typically XSS is proven by showing an javascript `alert(1)` messagebox. For instance in the following HTML, Banana Pancakes is an unescaped user supplied value:

```
<a href="/remove_product?product=Banana Pancakes">Remove Product</a>
```

The javascript alert box can be achieved by entering `" onclick="alert(1)" alt=""` instead of `Banana Pancakes` which will change the behaviour of the link.

```
<a href="/remove_product?product=" onclick="alert(1)" alt="">Remove Product</a>
```



Note how the double quotes are still balanced by the new input.

### 5.1.3 Including external HTML

Most templating frameworks also support including (or importing) snippets of markup from file or web based resources. This makes it easier to reuse markup. Sometimes the decision which file to include is based on user input. If the user input is part of the name of the file that is included it may be abused to include other unexpected resources that are available to the system but not to the user. This is called a [File inclusion vulnerability](#).

## 5.2 Reality

The goal of this Lab is to:

- Find the field that may be used to insert arbitrary HTML (including Javascript) into the Order screen and demonstrate it.
- Find the link that asks for confirmation in quite an unsafe way.
- Find the include that may be used to display unexpected HTML content and demonstrate it.

✔ Some tips to guide you on your way:

- Where does user interface code usually reside?
- How can you force Wicket to render unescaped HTML?
- How can you modify HTML attributes directly using Wicket?
- What Wicket component can be used to include arbitrary HTML resources?
- If you don't like .html extensions you can always make them go away as a parameter.
- Read the [Owasp XSS cheat sheet](#).

Please document your findings in the table below:

Issue	Source file and Line number	Demonstration Steps .....	Points
U/WA. 04.01  (escaping)			5
U/WA. 04.01  (escaping)			5
U/WA. 03.02  (includes)			5





### 6.1.4 Authentication and Authorization

The application should know for each request who is performing it (even if this is some abstract concept like an 'Anonymous user') and what authorizations this user has (browse products). The NCSC guideline recommends (U/TV.01) that you should use a centralized authentication and authorization system. This makes sense as [implementing one is hard](#). Frameworks such as [Spring Security](#) help a lot but still require you to write custom reset password or account lockout functionality.

Authorizations must be checked:

- before performing an action.
- before displaying any data.

For instance in the bad-cheesr webshop, an administrator is allowed to see all orders and process them, but an (anonymous) customer can only access his own order and add and remove cheeses. Also it should be impossible to access the Secret Cheese (however..)

## 6.2 Reality

A lot of things can go wrong when validating input. The four goals below demonstrate that, if you can find them!

- Find the flaw to bypass the authorization of the order pages.
- Spot the parameter that allows access to arbitrary records.
- Spot the missing validation and read some interesting files from the classpath as PDF.
- Spot the incorrect normalization and access arbitrary files on the file system.

✔ Some tips to guide you on your way:

- It's easy to make mistakes with the Servlet API.
- The authentication filter is rather specific, perhaps a little too specific.
- Owasp has a nice page on [url encoding techniques](#).

Please document your findings in the table below:

Issue	Source file and Line number	Demonstration Steps . . . . .	Points
U/PW.02.02 (authentication)			5
U/WA.03.01 (validation)			5
U/WA.03.01 (validation)			5
U/WA.03.04 <sup>(reject invalid)</sup>			5



Issue	Source file and Line number	Demonstration Steps . . . . .	Points
U/WA.03.03 (normalization)			5



## 7 Lab 4: Knowledge is in the Eye of the Beholder

This lab is about the guidelines that can not be reviewed against, but can be easily spotted from the browser.

### 7.1 Theory

Information leakage can make it a lot easier to successfully attack a web application. Application servers generally tell you its version and patch level on the response header by default, making vulnerability mapping easy. Stacktraces from the application tell you about the technology stack in use (programming language, web application framework etc). Commented out HTML may reveal juicy URLs. Sometimes it's even possible to list the files inside a folder on the webserver, like the css or javascript folders, or (heaven forbid!) the WEB-INF folder spilling the web applications most intimate details.

As a general security practice its recommended to remove any information that is not strictly required for running the application. The bad-cheesr application will demonstrate how not to do it.

### 7.2 Reality

The goals of this lab are:

- Find out what version of Jetty is being used.
- Find the Controller that produces pretty informative error pages.
- Find the TODO comment in the HTML of one of the pages..
- Display the contents of a resource folder in your browser.
- Find out if the Cookies are HTTP Only or not.



Some tips to guide you on your way:

- Most browsers include useful development tools that allow you to monitor HTTP requests.. Try `ctrl-shift-I` (firefox), `ctrl-shift-J` (chrome) or `F12` (IE) to open them.
- Wicket, when run in deployment-mode won't produce stack traces. Servlets however..

Please document your findings in the table below:

Issue	Value Found	Points
U/PW.02.04/05 (headers)		1
U/PW.02.06 (error details)		3
U/WA.06.01 (comments)		2
U/PW.03.02 (directory listing)		2



Issue	Value Found	Points
U/PW.03.03 (session cookie)		2

## 8 Lab 5: Beyond the Guideline

There are many more possible flaws than are present in the guideline (it's just a guideline and not a guarantee for safe software!). To illustrate I have included two other common flaws in bad-cheers:

- **ReDos**, a denial of service attack on badly constructed regular expressions. Yes, input validation can be used against you!
- **XXE**, XML eXternal Entity processing, a common flaw caused by using unconfigured XML parsers.
- Object deserialization vulnerabilities, caused by the default unrestricted classloading of `ObjectInputStream`.

### 8.1 Theory

#### 8.1.1 ReDos

ReDos occurs when the regular expression parser never finishes evaluating the given string against the regular expression. For backtracking parsers (the most common kind) this occurs when:

- the regular expression applies repetition ("+", "\*\*") to a complex subexpression;
- for the repeated subexpression, there exists a match which is also a suffix of another valid match.

For instance the if expression `(a|aa)+` is run against a long series of `a` characters, the parser can't determine if it should match against one `a` or two `a`'s leading to endless permutations.

A test vector for proving ReDos depends on the regular expression that's being tested but generally a *long series of identical characters* will do fine.

#### 8.1.2 Xml eXternal Entity processing

An **XML Entity** is a name for a character or series of characters in an XML document. A common example is the HTML non breaking space entity `&nbsp;`; which represents the Unicode character `0x00A0`. External entities have their contents stored in some external resource such as a file or webpage. You can declare your own entities in the header of the XML document as part of a DTD. When the XML is parsed the entities are replaced by the contents they represent. So if you would reference `file:///etc/passwd` as an entity on a Linux system the entity would be replaced by the contents of that file. Similarly you could put in an HTTP url and that resource will be loaded by the XML parser.

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



By default most XML parsers resolve external entities, so *explicit steps need to be taken to disable this feature* if the XML comes from an untrusted source such as the internet.

If part of the input xml is copied back into the output xml, XXE can be used to read files from the file system.

Alternatively you can cause a denial of service attack by using nested entities expanding to a huge document. This is called the billion laughs attack. However, modern parsers by default now limit the number of entity expansions to 65535 effectively blocking this attack vector. You can still get a nice warning though.



### 8.1.3 Object Deserialization Vulnerabilities

In Java applications its possible to Serialize and Deserialize Objects by using `ObjectInputStream` and `ObjectOutputStream`. `ObjectOutputStream` is used to convert an Object (graph) into a series of bytes and `ObjectInputStream` is used to read them back. It is a useful mechanism that allows fast implementation of a networking protocol between to Java virtual machines. `ObjectOutputStream` works on the Object level, so any Class available to its Classloader can be converted into an Object, not just the intended ones!

While you may think this can't possibly cause any trouble as no method gets invoked on the freshly deserialized object before its type is checked and therefore it becomes impossible to do anything malicious, this is not the case. There are several constructs that can trigger code execution while still deserializing the object:

- Classes that have a custom `readObject` method, which gets invoked when the object is deserialized.
- Proxy classes and Invocation handlers.
- Objects that contains code from the `java.lang.reflect` package and uses String fields to perform some custom invocation.

The actual object constructs tend be be rather complex but some [good examples exist](#). There is a [really good article by FoxGlove security](#) that describes this class of vulnerabilities in detail. Fixing the problem can be done by overriding the `resolveClass` method of `ObjectInputStream` and implementing a WhiteList of allowed classes. If you're using Remote Method Invocation (RMI) you can use [this project](#).

The vulnerability is easily spotted in the code, just search for unaltered `ObjectInputStream` usage.

## 8.2 Reality

The goal of this lab is to:

- Find and demonstrate the ReDoS. Keep your task manager ready to kill the process.
- Find and demonstrate the XXE by:
  - Stealing the pom.xml from the filesystem.
  - Attempting a [Billion laughs](#) attack.
- Find and demonstrate the Object deserialization vulnerability.

- ✔ Firing off http requests containing XML is easier if you use a command line utility such as `curl`.

```
curl -X POST http://localhost:8080/bad-cheesr/api/ --data @your_xml_file.xml
```

- ✔ The [frohoff/yoserial project](#) contains educational examples on how to use Object deserialization to execute arbitrary commands.

Please document your findings in the table below:

Issue	File and line number	Demonstration steps .....	Points
ReDoS			5
XXE, reading files			5



Issue	File and line number	Demonstration steps .....	Points
XXE, million laughs			5
Object deserialization			5



## 9 Lab 6: Find (security) Bugs?

**Findbugs** is Java source code auditing software with easy [maven integration](#). Let's find out if its any good in finding web application security bugs.

The pom.xml contains a commented out snippet at the bottom which holds the configuration for a findbugs report. Remove the outermost comments so that it looks like this:

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      <version>2.8</version>
      <reportSets>
        <reportSet>
          <reports>
            </reports>
          </reportSet>
        </reportSets>
      </plugin>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>findbugs-maven-plugin</artifactId>
      <version>3.0.1</version>
      <configuration>
        <xmlOutput>true</xmlOutput>
        <xmlOutputDirectory>target/site</xmlOutputDirectory>
        <!-- effort>Max</effort -->
        <!-- threshold>Low</threshold -->
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

The first plugin will make sure that no other reports will be rendered, the second plugin is the findbugs plugin.

Now, lets see what happens if you run this report against bad-cheers:

```
> mvn site:site
```

After a short while the report will be ready in the `./target/site/findbugs.html` file. Sadly, there is not much to report..



# bad-cheesr

Last Published: 2015-06-02 | Version: 1.0-SNAPSHOT bad-cheesr

**Project Documentation**

- Project Reports
  - FindBugs

Built by:

## FindBugs Bug Detector Report

The following document contains the results of [FindBugs](#)

FindBugs Version is *3.0.1*

Threshold is *medium*

Effort is *min*

## Summary

Classes	Bugs	Errors	Missing Classes
44	0	0	0

## Files

Class	Bugs
-------	------

Copyright © 2011-2015. All Rights Reserved.

Perhaps if we increase findbugs Effort and lower the Threshold better results will appear?

Uncomment the two parameters and run the report again. Did it help? Find out [which kind of security checks Findbugs supports](#).

Fortunately Findbugs has plugin support and more specialized plugins exist. One is the [find-sec-bugs plugin from h3xstream](#).

It is easily added to the findbugs plugin configuration by editing its configuration:

```
<configuration>
  <xmlOutput>true</xmlOutput>
  <xmlOutputDirectory>target/site</xmlOutputDirectory>
  <plugins>
    <plugin>
      <groupId>com.h3xstream.findsecbugs</groupId>
      <artifactId>findsecbugs-plugin</artifactId>
      <version>LATEST</version> <!-- Auto-update to the latest stable -->
    </plugin>
  </plugins>
</configuration>
```

Now, run the `mvn site:site` again and see if there is some improvement.



Files				
Class	Bugs			
<a href="#">com.cheesr.layers.dao.OrderDaoJpa</a>	1			
<a href="#">com.cheesr.layers.dao.ProductDaoJpa</a>	1			
<a href="#">com.cheesr.layers.web.XmlServlet</a>	1			
<a href="#">com.cheesr.layers.web.ZapFinancialsIntegrationServlet</a>	1			
<a href="#">com.cheesr.layers.web.validators.EmailValidator</a>	1			

com.cheesr.layers.dao.OrderDaoJpa				
Bug	Category	Details	Line	Priority
The query is potentially vulnerable SQL/JPQL injection	SECURITY	<a href="#">SQL_INJECTION_JPA</a>	34	Medium

com.cheesr.layers.dao.ProductDaoJpa				
Bug	Category	Details	Line	Priority
The query is potentially vulnerable SQL/JPQL injection	SECURITY	<a href="#">SQL_INJECTION_JPA</a>	39	Medium


com.cheesr.layers.web.XmlServlet				
Bug	Category	Details	Line	Priority
The usage of DocumentBuilder.parse(...) is	SECURITY	<a href="#">XXE_DOCUMENT</a>	44	Medium

Ah, much better!



## 10 Lab 7: Find Libraries with Security Issues

---

 This lab requires an internet connection.

Although the NCSC Guidelines are based on the OWASP Top 10, it doesn't include the [A9](#) 'Using components with known vulnerabilities' category.

Checking for it can be done as part of the build (it is a bit slow, so probably a nightly build) using Jeremy Long's [Dependency Check](#). It comes with a maven plugin so integration is easy.

In the `build` `plugins` section of the `pom.xml` remove the comments of the dependency check plugin. It should look like this:

```
<plugin>
  <groupId>org.owasp</groupId>
  <artifactId>dependency-check-maven</artifactId>
  <version>1.2.11</version>
  <executions>
    <execution>
      <goals>
        <goal>check</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Then run the build:



```
> mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building bad-cheesr 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ bad-cheesr ---
[INFO] Copying 6 resources
[INFO] Copying 2 resources
[INFO] Copying 8 resources
[INFO]
[INFO] --- maven-compiler-plugin:2.3.2:compile (default-compile) @ bad-cheesr ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- dependency-check-maven:1.2.11:check (default) @ bad-cheesr ---
[INFO] artifact commons-lang:commons-lang: checking for updates from central
[INFO] artifact org.easymock:easymock: checking for updates from central
.. etc ..
[INFO] artifact cglib:cglib-nodep: checking for updates from central
Jun 02, 2015 7:26:58 PM org.owasp.dependencycheck.Engine doUpdates
Jun 02, 2015 7:28:05 PM org.owasp.dependencycheck.data.update.StandardUpdate update
INFO: NVD CVE requires several updates; this could take a couple of minutes.
Jun 02, 2015 7:28:05 PM org.owasp.dependencycheck.data.update.task.DownloadTask call
INFO: Download Started for NVD CVE - 2008
Jun 02, 2015 7:28:05 PM org.owasp.dependencycheck.data.update.task.DownloadTask call
INFO: Download Started for NVD CVE - 2012
.. etc ..
INFO: Check for updates complete
Jun 02, 2015 7:29:13 PM org.owasp.dependencycheck.Engine analyzeDependencies
INFO: Analysis Starting
Jun 02, 2015 7:29:23 PM org.owasp.dependencycheck.Engine analyzeDependencies
INFO: Analysis Complete
Jun 02, 2015 7:29:23 PM org.owasp.dependencycheck.maven.BaseDependencyCheckMojo showSummary
WARNING:
One or more dependencies were identified with known vulnerabilities in bad-cheesr:
wicket-1.4.12.jar (cpe:/a:apache:wicket:1.4.12, org.apache.wicket:wicket:1.4.12) :
  CVE-2013-2055, CVE-2012-3373, CVE-2012-1089, CVE-2012-0047, CVE-2011-2712
spring-core-3.0.4.RELEASE.jar (cpe:/a:springsource:spring_framework:3.0.4,
  cpe:/a:vmware:springsource_spring_framework:3.0.4,
  org.springframework:spring-core:3.0.4.RELEASE) :
  CVE-2014-1904, CVE-2014-0054, CVE-2013-7315, CVE-2013-6429, CVE-2013-4152, CVE-2011-2894, CVE-2011-2730
spring-web-3.0.4.RELEASE.jar (cpe:/a:vmware:springsource_spring_framework:3.0.4,
  org.springframework:spring-web:3.0.4.RELEASE) :
  CVE-2011-2894
See the dependency-check report for more details.
```

Oh dear that does not look good. It seems that we have multiple vulnerable components. Quickly open the report located at `./target/dependency-check-report.html` for full details.

Since bad-cheesr uses ancient libraries it should not come as a surprise that vulnerabilities are found. You should monitor the libraries you use and upgrade when vulnerabilities are discovered. Dependency Check helps a lot. However because of the fuzzy matching it does between CPE / CVE issues and maven identifiers (GAV's) it sometimes produces **false positives** and also (worse!) false negatives.



## 11 The End

---

You have reached the end of this workshop. I hope you have gained some insight in reviewing web-applications and how to demonstrate flaws while having fun!

All workshop material compiled and written by [Erik Hooijmeijer](#). Special Thanks to Rob Gansevles for his detailed feedback. Thanks to the participants of the trial runs of this workshop at [42 B.V.](#) and IOO Gemeente Rotterdam.

(c) 2015 E.Hooijmeijer