Security Testing: Assignment #8

Security Test Cases

Fabrizio Zeni
Student Id: 153465

Contents

Introduction

This report shows the test cases developed through *JWebUnit* for testing the *true* vulnerabilities described in the *Assignment 7*. Finally, the last section is dedicated to the fix of such vulnerabilities.

Package

The package contains:

- the report
- *fixed* folder, containing the modified files to
- *notfixed* folder, containing the original versions of the files
- *dumps* folder, containing the dumps from the database
- *src* folder, source code for the JWebUnit test cases

Vulnerability 11

Brief Analysis

File: AddAssignment.php
Similar Vulnerabilities[1]:
16,18,19,63,70,71,87,88,89,90,93,126,138,141,165,180,181,186,194,200,201,230,238,269,272,316

VARIABLE	RESULT
page	true
page2	true
selectclass	true

JWebUnit test cases

prepare and cleanup

```
public void prepare(){
      tester = new WebTester();
      tester.setBaseUrl("http://localhost/sm/");
      tester.beginAt("index.php");
      Functions.login(tester,"teacher");
      Functions.click(tester,"Music",0);
      tester.assertMatch("Class Settings");
      Functions.click(tester,"Assignments",0);
      tester.assertMatch("Manage Assignments");
}
```

Listing 1: prepare function

```
public void cleanup(){
  Functions.click(tester,"Log Out",0);
  tester = null;
}
```

Listing 2: cleanup function

In these two functions there is nothing special, just navigation and call to the login/logout utilities.

Continues on the next page ...

[1] their test cases are based on the ones of this vulnerability

page

```
public void page(){
  Vulnerabilities.page(tester,"assignments","Add");
    tester.assertMatch("Add New Assignment");
    tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 3: jwebunit test code for *page*

```
public static void page(WebTester tester,String formName,String buttonName){
    IElement page = tester.getElementByXPath("//form[@name='" + formName + "']//input[
        @name='page']");
    String oldValue = page.getAttribute("value");
    page.setAttribute("value",oldValue +"'><a href='http://www.unitn.it'>malicious</a><
        br'");
    if(buttonName!=null)
      Functions.click(tester,buttonName,1);
}
```

Listing 4: function for the *page* vulnerability

This code does the test for *page*. In order to catch the correct hidden field it was necessary to filter the form first, because there were two hidden fields with the same name and the first is not the one triggered by the buttons. So the function retrieves the page2 input element and stores it into the *oldValue* variable, which at line 6 is concatenated to the malicious link and inserted into the page value.

page2

```
public void page2(){
  Vulnerabilities.page2(tester,"assignments","Add");
    tester.assertMatch("Add New Assignment");
    tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 5: jwebunit test code for *page2*

```
public static void page2(WebTester tester,String formName,String buttonName){
  IElement page2 = tester.getElementByXPath("//form[@name='" + formName + "']//input[@name
      ='page2']");
  IElement button = tester.getElementByXPath("//input [@value='" + buttonName + "']");
  String onClick = button.getAttribute("onClick");
  String[] fixedValues = Functions.page2Fix(formName, onClick);
  fixedValues[0] = fixedValues[0].replace("'","");
  page2.setAttribute("value",fixedValues[0] + "'><a href='http://www.unitn.it'>malicious</
      a><br'");
  button.setAttribute("onClick",fixedValues[1]);
  Functions.click(tester,buttonName,1);
}
```

Listing 6: function for the *page2* vulnerability

The page2 vulnerability was more subtle to automatically trigger. That was due to the fact that the form buttons have a *javascript* code in the attribute **onClick**, which write on the page2 value. So that in order to prevent the button from modify the injected value, at line 3 the button element is retrieved, then we get the value of the onClick attribute, which is processed by the *page2Fix function* - which purge the attribute from any command that modifies the page2 value and returns the value for page2 and the other instructions that need to be put back into the attribute.

Vulnerability 11 continued on next page...

selectclass

```
public void selectclass(){
  Vulnerabilities.selectclass(tester,"assignments","Add");
    tester.assertMatch("Add New Assignment");
    tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 7: jwebunit test code for *selectclass*

```
public static void selectclass(WebTester tester,String formName,String buttonName){
    IElement selectclass = tester.getElementByXPath("//form[@name='" + formName + "']//
        input[@name='selectclass']");
    String oldValue = selectclass.getAttribute("value");
    selectclass.setAttribute("value",oldValue+"'><a href='http://www.unitn.it'>malicious
        </a><br'");
    Functions.click(tester,buttonName,1);
}
```

Listing 8: function for the *selectclass* vulnerability

The selectclass vulnerability was almost straightforward and differs from the *page* function just in the attribute name in the XPath expression.

Vulnerability 13

Brief Analysis

File: AddAttendance.php
Similar Vulnerabilities[2]: 194

VARIABLE	RESULT
page	true
page2	true
student	true
semester	true

JWebUnit test cases

prepare and cleanup

```
public void prepare(){
      tester = new WebTester();
      tester.setBaseUrl("http://localhost/sm/");
      tester.beginAt("index.php");
      Functions.login(tester,"admin");
  Functions.click(tester,"Attendance",0);
  tester.assertMatch("Tardy");
}
```

Listing 9: prepare function

```
public void cleanup(){
  Functions.click(tester,"Log Out",0);
  tester = null;
}
```

Listing 10: cleanup function

page and page2

The code is adapted from the one of *Vulnerability 11* at page 6

[2]their test cases are based on the ones of this vulnerability

student

```
public void student(){
  Vulnerabilities.selectInputVulnerability(tester,"registration","Add","student");
    tester.assertMatch("Add New Attendance Record");
    tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 11: jwebunit test code for *student*

```
public static void selectInputVulnerability(WebTester tester,String formName,String
    buttonName,String vulnerability){
  IElement selectInput = tester.getElementByXPath("//form[@name='" + formName +
    "']//select[@name='" + vulnerability + "']//option[@selected]");
    String oldValue = selectInput.getAttribute("value");
    selectInput.setAttribute("value",oldValue+"'><a href='http://www.unitn.it'>malicious
        </a><br /'");
    Functions.click(tester,buttonName,1);
}
```

Listing 12: function for vulnerabilities over select input elements

In this case the input element was a **select**, so the XPATH expression was modified with *//option[@selected]* to catch the selected option. The remaining part of the code is almost equivalent to the *page* one.

semester

```
public void semester(){
  Vulnerabilities.selectInputVulnerability(tester,"registration","Add","semester");
    tester.assertMatch("Add New Attendance Record");
    tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 13: jwebunit test code for *semester*

The semester test is a copy-paste of the student one.

Vulnerability 30,31

Brief Analysis

File: ViewAssignments.php
Similar Vulnerabilities[3]: 207

VARIABLE	RESULT
coursename	true
assignment[5]	true

JWebUnit test cases

prepare and cleanup

```
public void prepare(){
      tester = new WebTester();
      tester.setBaseUrl("http://localhost/sm/");
      tester.beginAt("index.php");
      Functions.login(tester,"student");
      Functions.click(tester,"Music",0);
      tester.assertMatch("Class Settings");
}
```

Listing 14: prepare function

```
public void cleanup() {
  Functions.click(tester, "Log Out", 0);
  // BEGIN COURSENAME CLEANUP
  Functions.login(tester, "admin");
  Functions.click(tester, "Classes", 0);
  tester.assertMatch("Manage Classes");
  IElement myCheckbox = tester
      .getElementByXPath("//td[text()='Music ']/..//input[@type='checkbox ']");
  tester.setWorkingForm("classes");
  tester.checkCheckbox("delete[]", myCheckbox.getAttribute("value"));
  Functions.click(tester, "Edit", 1);
  tester.assertMatch("Edit Class");
  tester.setTextField("title","Music");
  Functions.click(tester,"Edit Class", 1);
  Functions.click(tester, "Log Out", 0);
  // END COURSENAME CLEANUP
  tester = null;
}
```

Listing 15: cleanup function

[3]their test cases are based on the ones of this vulnerability

Vulnerability 30,31 continued on next page...

coursename

```
public void coursename() {
  Functions.click(tester, "Log Out", 0);
  tester.assertMatch("TutttoBBBene");
  // INJECTING A LINK IN THE COURSENAME
  Functions.login(tester, "admin");
  Functions.click(tester, "Classes", 0);
  tester.assertMatch("Manage Classes");
  IElement myCheckbox = tester
      .getElementByXPath("//td[text()='Music ']/..//input[@type='checkbox ']");
  tester.setWorkingForm("classes");
  tester.checkCheckbox("delete[]", myCheckbox.getAttribute("value"));
  Functions.click(tester, "Edit", 1);
  tester.assertMatch("Music");
  tester.assertMatch("Edit Class");
  Vulnerabilities.textFieldVulnerability(tester, "editclass", "title",
      "Edit Class");
  tester.assertLinkPresentWithText("malicious");
  Functions.click(tester, "Log Out", 0);
  // CHECKING THE VULNERABILITY
  Functions.login(tester, "student");
  Functions.click(tester, "Music", 0);
  tester.assertMatch("Class Settings");
  Functions.click(tester, "Assignments", 0);
  tester.assertMatch("View Assignments");
  tester.assertLinkNotPresentWithText("a");
}
```

Listing 16: jwebunit test code for *coursename*

This test is a bit more verbose, because in order to test the *coursename* vulnerability a injection made through an admin account is required.

```
public static void textFieldVulnerability(WebTester tester,
    String formName, String fieldName,String buttonName) {
  String oldValue = tester.getElementByXPath("//input [@name='" + fieldName + "']").
      getAttribute("value");
  tester.setTextField(fieldName,oldValue + "<a href>malicious</a>");
  Functions.click(tester,buttonName, 1);
}
```

Listing 17: function used to inject links in textfields

For this vulnerability, I wrote a generic function in the Vulnerability class which is able to process vulnerabilities over text fields.

Vulnerability 30,31 continued on next page...

assignment[5]

```
  public void assignmentInformation(){
    MySql.executeUpdate("UPDATE assignments SET assignmentinformation = '<a href>malicious</
        a>' WHERE assignmentid = '3'");
    Functions.click(tester, "Assignments", 0);
    tester.assertMatch("View Assignments");
    tester.assertLinkNotPresentWithExactText("malicious");
  }
```

Listing 18: jwebunit test code for *asignment[5]*

Database loads test and fixes

Because *coursename* and *assignment[5]* are loaded from the database, even if we apply a fix to the update, a pre-existent injection can present on the database and then, even if the post variables are sanitez, the upcoming data from the database could be a security issue until they are updated once. So a test which checks this case should be made.

Test Cases

```
  public void coursenameSQL(){
    MySql.executeUpdate("UPDATE courses SET coursename = 'Music<a href>mal</a>' WHERE
        courseid = '1'");
    Functions.click(tester, "Assignments", 0);
    tester.assertMatch("View Assignments");
    tester.assertLinkNotPresentWithExactText("mal");
  }
```

Listing 19: jwebunit test code for *coursename* over pre-existent db data

Fix

```
// SANITIZING coursename
//$coursename = mysql_result($query,0);
$coursename = htmlspecialchars(mysql_result($query,0));
```

Listing 20: sanitization of *coursename* over a db load injection

```
// SANITIZING assignmentinformation
$assignment[5] = htmlspecialchars($assignment[5]);
```

Listing 21: sanitization of *assignment[5]* over a db load injection

Vulnerability 37

Brief Analysis

File: EditAssignment.php
Similar Vulnerabilities[4]: 41,44,85,111,115,149,161,239

VARIABLE	RESULT
page	true
page2	true
selectclass	true
delete	true

JWebUnit test cases

prepare and cleanup

```
public void prepare(){
  tester = new WebTester();
  tester.setBaseUrl("http://localhost/sm/");
  tester.beginAt("index.php");
  Functions.login(tester,"teacher");
  Functions.click(tester,"Music",0);
  tester.assertMatch("Class Settings");
  Functions.click(tester,"Assignments",0);
  tester.assertMatch("Manage Assignments");
  tester.assertMatch("verifica di prova");
  IElement myCheckbox = tester.getElementByXPath("//td[text()='prova2']/..//input[@type='
      checkbox']");
  tester.setWorkingForm("assignments");
  tester.checkCheckbox("delete[]",myCheckbox.getAttribute("value"));
}
```

Listing 22: prepare function

The prepare functions was a bit longer this time, because in order to access to the reported page one of the assignment has to be checked in the checkbox element. This is done by retrieving the line of the assignment *prova* and finally we set insert in the *delete[]* the value of the selected assignment.

```
public void cleanup(){
  Functions.click(tester,"Log Out",0);
  tester = null;
}
```

Listing 23: cleanup function

page, page2 and selectclass

The code is adapted from the one of *Vulnerability 11* at page 6

[4]their test cases are based on the ones of this vulnerability

Vulnerability 37 continued on next page...

delete

```
public void delete(){
  Vulnerabilities.delete(tester,"assignments","Edit","prova2");
  tester.assertMatch("EditAssignment.php: Unable to retrieve");
  tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 24: jwebunit test code for *delete*

```
public static void delete(WebTester tester,String formName,String buttonName,String
    checkBoxText){
  IElement myCheckBox = tester.getElementByXPath("//td[text()='" + checkBoxText
      + "']/..//input[@type='checkbox']");
  String oldValue = myCheckBox.getAttribute("value");
  myCheckBox.setAttribute("value",oldValue + "';<a href=http://www.unitn.it>malicious</a>"
      );
  tester.assertButtonPresentWithText("Edit");
  System.err.println(myCheckBox.getAttribute("value"));
  Functions.click(tester,buttonName,1);
}
```

Listing 25: function for the *delete* vulnerability

The interesting thing of this case is that even a *sql injection* is possible by putting another query after the semicolon.

Vulnerability 54

Brief Analysis

File: Login.php

VARIABLE	RESULT
text	true

JWebUnit test cases

prepare and cleanup

```
public void prepare(){
      tester = new WebTester();
      tester.setBaseUrl("http://localhost/sm/");
      tester.beginAt("index.php");
     Functions.login(tester,"admin");
      Functions.click(tester,"School",0);
      tester.assertMatch("Manage School Information");
      oldValue = tester.getElementByXPath("//textarea [@name='sitetext']").getTextContent
          ();
}
```

Listing 26: prepare function

```
public void cleanup(){
      tester.assertMatch("Today's Message");
      Functions.login(tester, "admin");
      tester.clickLinkWithText("School");
      tester.assertMatch("Manage School Information");
      tester.setTextField("sitetext", oldValue);
      Functions.click(tester," Update ",1);
      Functions.click(tester,"Log Out",0);
      tester = null;
}
```

Listing 27: cleanup function

text

```
public void siteText(){
      tester.setTextField("sitetext", "<a href=\"http://www.unitn.it\">malicious</a>");
      Functions.click(tester," Update ",1);
      Functions.click(tester,"Log Out",0);
      tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 28: jwebunit test code for *text*

Vulnerability 54 continued on next page...

Database loads test and fixes

Because *text* is loaded from the database, even if we apply a fix to the update a pre-existent injection can present on the database and then even if the post variables are sanitez, the upcoming data from the database could be a security issue until they are updated once. So a test which checks this case should be made.

Test Case

```
public void siteTextSQL(){
  MySql.executeUpdate("UPDATE schoolinfo SET sitetext = '<a href>malicious</a>'");
  Functions.click(tester,"Log Out",0);
  tester.assertLinkNotPresentWithExactText("malicious");
}
```

Listing 29: jwebunit test code for *text* over pre-existent db data

Fix

```
// SANITIZING sitetext
//$text  = mysql_result($query,0);
$text  = htmlspecialchars(mysql_result($query,0));
```

Listing 30: sanitization of *text* over a db load injection

Vulnerability 76

Brief Analysis

File: EditAnnouncement.php

VARIABLE	RESULT
page	true
page2	true
selectclass	true
assignment	true
delete	true

JWebUnit test cases

prepare and cleanup

```
public void prepare(){
  tester = new WebTester();
  tester.setBaseUrl("http://localhost/sm/");
  tester.beginAt("index.php");
  Functions.login(tester,"teacher");
  Functions.click(tester,"Music",0);
  tester.assertMatch("Class Settings");
  Functions.click(tester,"Grades",0);
  tester.assertMatch("Date Submitted");
  IElement myCheckbox = tester.getElementByXPath("//td[text()='Harry Potter']/..//input[
      @type='checkbox']");
  tester.setWorkingForm("grades");
  tester.checkCheckbox("delete[]",myCheckbox.getAttribute("value"));
}
```

Listing 31: prepare function

```
public void cleanup(){
  Functions.click(tester,"Log Out",0);
  tester = null;
}
```

Listing 32: cleanup function

page,page2,selectclass and delete

The code is adapted from the one of *Vulnerability 37* at page 14

assignment

```
public void assignment(){
  Vulnerabilities.selectInputVulnerability(tester,"grades","Edit","assignment");
3 tester.assertMatch("EditGrade.php: Unable to retrieve");
  tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 33: jwebunit test code for *assignment*

```
$query = mysql_query("SELECT submitdate, points, comment, islate, gradeid FROM grades WHERE
    studentid = '$id[0]' AND assignmentid = '$_POST[assignment]'")
```

Listing 34: EditGrade.php read of assignment

In this case, the input element is a *select*, but the posted variable is printed inside an sql query - so as already said for *Vulnerability 37* - an Sql Injection is also possible.

Vulnerability 87

Brief Analysis

File: ViewClassSettings.php
Similar Vulnerabilities[5]: 90,126,138,183,184,299,309

VARIABLE	RESULT
page	true
page2	true
selectclass	true

JWebUnit test cases

The code for *page* and *selectclass* is adapted from the one for *Vulnerability 11* at page 6

page2

```
public void page2(){
  Vulnerabilities.page2Link(tester, "student", "Settings",
    "document.student.submit();","parent");
    tester.assertMatch("Class Settings");
    tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 35: jwebunit test code for *page2*

```
public static void page2Link(WebTester tester,String formName,String linkName,String
    hrefValue,String user){
  IElement page2 = tester.getElementByXPath("//form[@name='" + formName + "']//input[@name
      ='page2']");
  if(linkName!=null){
    IElement link = tester.getElementByXPath("//a[text()='" + linkName + "']");
    link.setAttribute("href","javascript: " + hrefValue);
    Integer page2Value = Functions.getPage2(linkName,user);
    page2.setAttribute("value",page2Value + "'><a href='http://www.unitn.it'>malicious</a
        ><br'");
    Functions.click(tester,linkName,0);
  }else{
    String page2Value = page2.getAttribute("value");
    page2.setAttribute("value",page2Value + "'><a href='http://www.unitn.it'>malicious</a
        ><br'");
  }
}
```

Listing 36: function for the page2 vulnerability with links

Here a modified version of the page2 utility function is used. That is due to the fact that in this case we have to modify a link instead of a button.

[5]their test cases are based on the ones of this vulnerability

Vulnerability 92

Brief Analysis

File: ManageSchoolInfo.php

VARIABLE	RESULT
page	true
page2	true
address	true

JWebUnit test cases

prepare and cleanup

```
public void prepare(){
      tester = new WebTester();
      tester.setBaseUrl("http://localhost/sm/");
      tester.beginAt("index.php");
      Functions.login(tester,"admin");
  tester.assertMatch("Manage Classes");
  Functions.click(tester, "School", 0);
      oldValue = tester.getElementByXPath("//input [@name='schooladdress']").getAttribute(
          "value");
      Functions.click(tester, "Classes", 0);
      tester.assertMatch("Manage Classes");
}
```

Listing 37: prepare function

```
public void cleanup(){
  IElement schooladdress = tester.getElementByXPath("//input[@value='Fake']");
  schooladdress.setAttribute("value", oldValue);
      Functions.click(tester," Update ",1);
  schooladdress.setAttribute("value", oldValue);
      Functions.click(tester," Update ",1);
  Functions.click(tester,"Log Out",0);
  tester = null;
}
```

Listing 38: cleanup function

page and page2

The code is adapted from the one of *Vulnerability 11* at page 6

Vulnerability 92 continued on next page...

address

```
public void address(){
      Functions.login(tester,"admin");
      Functions.click(tester,"School",0);
      tester.assertMatch("Manage School Information");
      IElement schooladdress = tester.getElementByXPath("//input[@name='schooladdress']");
      schooladdress.setAttribute("value","Fake\\'><a href>malicious</a><br\\'");
      Functions.click(tester," Update ",1);
      tester.assertLinkNotPresentWithExactText("malicious");
}
```

Listing 39: jwebunit test code for *address*

Database loads test and fixes

Because *address* is loaded from the database, even if we apply a fix to the update a pre-existent injection can present on the database and then even if the post variables are sanitez, the upcoming data from the database could be a security issue until they are updated once. So a test which checks this case should be made.

Test Case

```
public void addressSQL(){
  MySql.executeUpdate("UPDATE schoolinfo SET" +
        "`address`= '" + oldValue + "\\'><a href>malicious</a><br\\''");
  Functions.click(tester,"School",0);
  tester.assertLinkNotPresentWithExactText("malicious");
}
```

Listing 40: jwebunit test code for *address* over pre-existent db data

Fix

```
// SANITIZING address
//$address = mysql_result($query,0);
$address = htmlspecialchars(mysql_result($query,0));
```

Listing 41: sanitization of *address* over a db load injection

Vulnerability 105

Brief Analysis

File: Login.php

VARIABLE	RESULT
page	true
message	true

JWebUnit test cases

prepare and cleanup

```
    tester = new WebTester();
    tester.setBaseUrl("http://localhost/sm/");
    tester.beginAt("index.php");
    Functions.login(tester, "admin");
    Functions.click(tester, "School", 0);
    tester.assertMatch("Manage School Information");
    IElement textArea = tester.getElementByXPath("//textarea [@name='sitemessage']");
    oldValue = textArea.getTextContent();
    tester.setTextField("sitemessage", "<a href>malicious</a>");
    Functions.click(tester," Update ", 1);
    Functions.click(tester, "Log Out", 0);
    tester.assertMatch("Today's Message");
```

Listing 42: prepare function

```
    Functions.login(tester, "admin");
    Functions.click(tester, "School", 0);
    tester.assertMatch("Manage School Information");
    tester.setTextField("sitemessage", oldValue);
    Functions.click(tester," Update ", 1);
    Functions.click(tester, "Log Out", 0);
    tester.assertLinkNotPresentWithText("malicious");
    tester = null;
```

Listing 43: cleanup function

page

```
  public void page() {
    Vulnerabilities.page(tester, "login","Login");
    tester.assertMatch("Today's Message");
    tester.assertLinkNotPresentWithText("malicious");
  }
```

Listing 44: jwebunit test code for *page*

message

```
    tester.assertLinkNotPresentWithText("malicious");
```

Listing 45: jwebunit test code for *message*

Vulnerability 105 continued on next page...

Database loads test and fixes

Because *message* is loaded from the database, even if we apply a fix to the update a pre-existent injection can present on the database and then even if the post variables are sanitez, the upcoming data from the database could be a security issue until they are updated once. So a test which checks this case should be made.

Test Case

```
public void messageSQL(){
  Functions.login(tester,"student");
  MySql.executeUpdate("UPDATE schoolinfo SET" +
        "`sitemessage`= '<a href>malicious</a>'");
  Functions.click(tester, "Log Out", 0);
  tester.assertLinkNotPresentWithExactText("malicious");
}
```

Listing 46: jwebunit test code for *message* over pre-existent db data

Fix

```
// SANITIZING message
//$message = mysql_result($query,0);
$message = htmlspecialchars(mysql_result($query,0));
```

Listing 47: sanitization of *message* over a db load injection

Vulnerability 142

Brief Analysis

File: ParentViewCourses.php

VARIABLE	RESULT
page	true
page2	true
student	true

JWebUnit test cases

page and page2

The code is adapted from the one of *Vulnerability 11* at page 6.

student

```
public void student(){
        IElement student = tester.getElementByXPath("//form[@name='student']//input[@name='
            student']");
        String oldValue = student.getAttribute("value");
        student.setAttribute("value",oldValue +"';<a href=http://www.unitn.it>malicious</a>"
            );
    Functions.click(tester,"Classes",0);
        tester.assertMatch("ParentViewCourses.php: Unable to get the studentid 2");
        tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 48: jwebunit test code for *student*

Vulnerability 146

Brief Analysis

File: ViewAnnouncements.php
Similar Vulnerabilities[6]: 147,148,183,184,257,260,268,273,283,288,293,309,320

VARIABLE	RESULT
page	true
page2	true
onpage	true

JWebUnit test cases

page and page2

The code is adapted from the one of *Vulnerability 11* at page 6.

onpage

```
<a href='JavaScript: document.announcements.deleteannouncement.value=0;document.
    announcements.page2.value=4;document.announcements.onpage.value=1;document.announcements
    .submit();' class='selectedpagenum' onMouseover="window.status='Go to page 1';return
    true;" onMouseout="window.status='';return true;">1</a>
```

Listing 49: portion of code of the generated ViewAnnouncements page

In this page there were a coding error, infact the *document.announcements.deleteannouncement.value=0;* command was responsible of the malfunctioning of the above link. That was due to the fact that *deleteannouncement* was not an item of the *announcements* form and so it turns out in an error. I removed from the page that first command and so now the page works properly.

```
  @Test
  public void onpage(){
3   Functions.click(tester,"Announcements",0);
    Vulnerabilities.onpage(tester,"1","announcements");
    Functions.click(tester,"1",0);
6   tester.assertMatch("View Announcements");
    tester.assertLinkNotPresentWithText("malicious");
  }
```

Listing 50: jwebunit test code for *onpage*

[6] their test cases are based on the ones of this vulnerability

Vulnerability 191

Brief Analysis

File: DeficiencyReport.php
Similar Vulnerabilities[7]: 212,241

VARIABLE	RESULT
page	true
page2	true

JWebUnit test cases

The JWebUnit test cases of this vulnerability, were a bit different from the others, the access to the page is done through a *select* with an *onChange trigger*.

page

```
  public void page(){
    Vulnerabilities.page(tester,"students",null);
3   tester.selectOption("report","Deficiency Report");
    tester.assertMatch("Deficiency Report");
    tester.assertLinkNotPresentWithText("malicious");
6 }
```

Listing 51: jwebunit test code for *page*

page2

```
  public void page2(){
    IElement mySelect = tester.getElementByXPath("//option[text()='Deficiency Report']");
3   String optionValue = mySelect.getAttribute("value");
    mySelect.setAttribute("value",optionValue + "'><a href='http://www.unitn.it'>malicious</
        a><br'");
    tester.selectOption("report","Deficiency Report");
6   tester.assertMatch("Deficiency Report");
    tester.assertLinkNotPresentWithText("malicious");
  }
```

Listing 52: jwebunit test code for *page*

The page2 test case took advantage of this part of the onChange attribute of the select item:

```
<select name='report' onChange='document.students.page2.value=document.students.report.
    value;document.students.deletestudent.value=0;document.students.submit();'>
```

Listing 53: portion of the source code of the displayed page (ViewStudents)

In particular, *document.students.page2.value=document.students.report.value;*, give the possibility to inject the attack in the value of the select option, as can be seen in the Listing 52 from line 2 to 4.

[7] their test cases are based on the ones of this vulnerability

Vulnerability 234

Brief Analysis

File: ManageSemesters.php

VARIABLE	RESULT
term	true

JWebUnit test cases

prepare and cleanup

```
public void prepare(){
      tester = new WebTester();
      tester.setBaseUrl("http://localhost/sm/");
      tester.beginAt("index.php");
      Functions.login(tester,"admin");
      Functions.click(tester,"Terms",0);
      tester.assertMatch("Manage Terms");
  IElement myCheckbox = tester.getElementByXPath("//td[text()='09/10/2012']/..//input[
      @type='checkbox']");
  tester.setWorkingForm("terms");
  tester.checkCheckbox("delete[]",myCheckbox.getAttribute("value"));
  Functions.click(tester,"Edit",1);
  tester.setTextField("title","<a href>a</a>");
  Functions.click(tester,"Edit Term",1);
}
```

Listing 54: prepare function

```
public void cleanup(){
  Functions.click(tester,"Terms",0);
  IElement myCheckbox = tester.getElementByXPath("//td[text()='09/10/2012']/..//input[
      @type='checkbox']");
  tester.setWorkingForm("terms");
  tester.checkCheckbox("delete[]",myCheckbox.getAttribute("value"));
  Functions.click(tester,"Edit",1);
  tester.setTextField("title","2012-2013");
  Functions.click(tester,"Edit Term",1);
  Functions.click(tester,"Log Out",0);
  tester = null;
}
```

Listing 55: cleanup function

term

```
public void term(){
  Functions.click(tester,"Semesters",0);
      tester.assertMatch("Manage Semesters");
      tester.assertLinkNotPresentWithExactText("a");
}
```

Listing 56: jwebunit test code for *page*

Vulnerability 234 continued on next page...

Database loads test and fixes

Because *term* is loaded from the database, even if we apply a fix to the update a pre-existent injection can present on the database and then even if the post variables are sanitez, the upcoming data from the database could be a security issue until they are updated once. So a test which checks this case should be made.

Test Case

```
public void termSQL(){
  MySql.executeUpdate("UPDATE terms SET" +
        "`title`= '<a href>a</a>'");
  Functions.click(tester,"Semesters",0);
      tester.assertMatch("Manage Semesters");
  tester.assertLinkNotPresentWithExactText("a");
}
```

Listing 57: jwebunit test code for *term* over pre-existent db data

Fix

```
// SANITIZING term
//$term = mysql_result($query2,0);
$term = htmlspecialchars(mysql_result($query2,0));
```

Listing 58: sanitization of *term* over a db load injection

Vulnerability 269

Brief Analysis

File: AddClass.php

VARIABLE	RESULT
page	true
page2	true
fullyear	true

JWebUnit test cases

The implementation of *prepare*,*cleanup*, *page* and *page2* are adapted from the code for *Vulnerability 11* at page 6.

term

```
public void fullyear(){
  Functions.click(tester,"Add",1);
  tester.assertMatch("Add Class");
  IElement fullyearButton = tester.getElementByXPath("//form[@name='addclass']//input[@value='Full Year']");
  fullyearButton.setAttribute("onClick","document.addclass.submit();");
      IElement fullyear = tester.getElementByXPath("//form[@name='addclass']//input[@name='fullyear']");
      fullyear.setAttribute("value","1'><a href=http://www.unitn.it>malicious</a>");
  Functions.click(tester,"Full Year",1);
      tester.assertMatch("Add Class");
      tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 59: jwebunit test code for *fullyear*

Fixing the Vulnerabilities

First Attempt

The first idea for the sanitization was to sanitized the POST variables in the page where they are used. For instance the sanitization of page and page2 in this case was as follows.

```
// SANITIZE page
//$page = $_POST["page"];
$page = htmlspecialchars($_POST["page"]);
```

Listing 60: sanitization of *page*

The *page* is accessed directly from the *index.php* page, which is the base for each displayed page. So by sanitizing it at line ∼36, the vulnerability is removed.

```
//$page2 = $_POST["page2"];
// SANITIZING PAGE2
$page2 = htmlspecialchars($_POST["page2"]);
```

Listing 61: sanitization of *page2*

The sanitization of page2 has to be done in the main page of each *user type*. In the example of Listing 61 can be seen the sanitization of *TeacherMain.php*, where the sanitization is done at line ∼8

```
// SANITIZING selectclass
$_POST['selectclass'] = htmlspecialchars($_POST['selectclass']);
```

Listing 62: sanitization of *selectclass*

As for *page2*, this procedure should be repeated for each user type, but this time because the page uses directly (and several times), the *$_POST* variable, then I added the line above at line ∼69, so that the post variable is now safe.

However, after fixing some of them, I though about a more efficient (and less time consuming) solution.

Sanitizing the POST variables

I thought that a better solution was to sanitize all the POST parameters by putting a *foreach* cycle at the beginning of *index.php*.

```
foreach ($_POST as $key => $value){
  echo $key;
if($key != "delete")
  $_POST[$key] =  htmlspecialchars($value);
else{
  foreach ($value as $newkey => $newvalue){
    $value[$newkey] = htmlspecialchars($newvalue);
  }
  $_POST[$key] = $value;
}
}
```

Listing 63: POST variables sanitization cycle

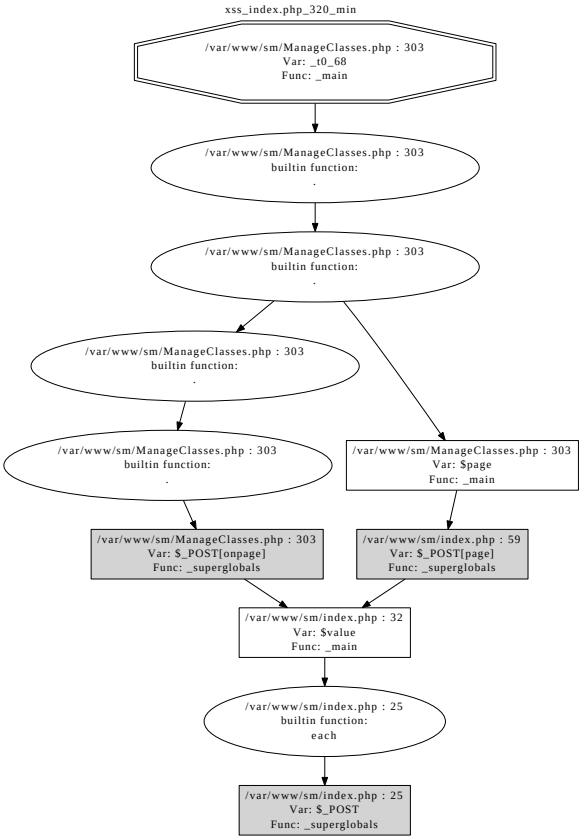
The if-else guard was needed because the *delete* variable is an array and so its cells should be sanitized. By doing this sanitization, all the test cases were running successfully.

Pre-fix database entries

Even if the fix should prevent any successful injection, the database might contain something that was injected before the fix took place. So whenever the vulnerabilities comes from a database enty, it would be better to sanitize it. Because there are some vulnerabilities that may suffer of this, the code and explanation was made in the *Database loads test and fixes* of each vulnerability subjected to this problem.
The modified files for this fixes are: *index.php*, *ViewAssignments.php*, *ManageAssignments.php*, *Login.php*, *ManageSchoolInfo.php* and *ManageSemesters.php*.

Taint Analysis rerun

Pixy was not able to find any new (true) xss vulnerability, the old false positives were reported along with some new false positives which were of the form of the graph below:



All the POST variables now are inserted into $value when the *sanitization block* of Listing 63 is performed. But the $value variable is just used to sanitize the post and is not used elsewhere, so all of these new cases are false positives.