Security Testing: Assignment #8

Due on Friday, April 19, 2013

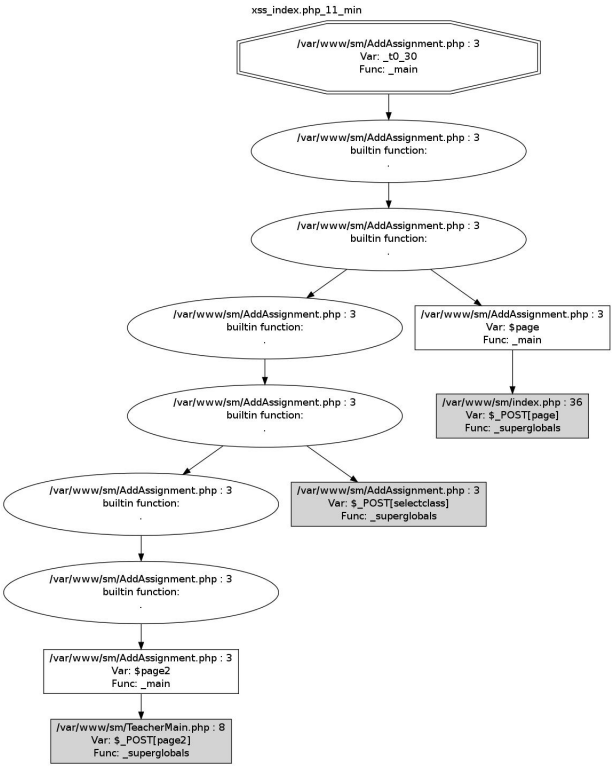
Jones 13:30am

Fabrizio Zeni

Contents

Vulnerability 11

Brief Analysis



File: AddAssignment.php

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);
3   tester = null;
  }
```

Listing 2: cleanup function

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

page

```
  public void page(){
    Vulnerabilities.page(tester,"assignments","Add");
3       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']");
3       String oldValue = page.getAttribute("value");
        page.setAttribute("value",oldValue +"'><a href='http://www.unitn.it'>malicious</a><
            br'");
        if(buttonName!=null)
6         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");
3       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']");
3   IElement button = tester.getElementByXPath("//input [@value='" + buttonName + "']");
    String onClick = button.getAttribute("onClick");
    String[] fixedValues = Functions.page2Fix(formName, onClick);
6   page2.setAttribute("value",fixedValues[0] + "'><a href='http://www.unitn.it'>malicious</
        a><br'");
    button.setAttribute("onClick",fixedValues[1]);
    Functions.click(tester,buttonName,1);
9 }
```

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.

selectclass

```
  public void selectclass(){
    Vulnerabilities.selectclass(tester,"assignments","Add");
3       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']");
3       String oldValue = selectclass.getAttribute("value");
        selectclass.setAttribute("value",oldValue+"'><a href='http://www.unitn.it'>malicious
            </a><br'");
        Functions.click(tester,buttonName,1);
6   }
```

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

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

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

Listing 11: jwebunit test code for *page*

page2

```
public void page2(){
  Vulnerabilities.page2(tester,"registration","Add");
      tester.assertMatch("Add New Attendance Record");
      tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 12: jwebunit test code for *page2*

Vulnerability 13 continued on next page...

student

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

Listing 13: 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 14: 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

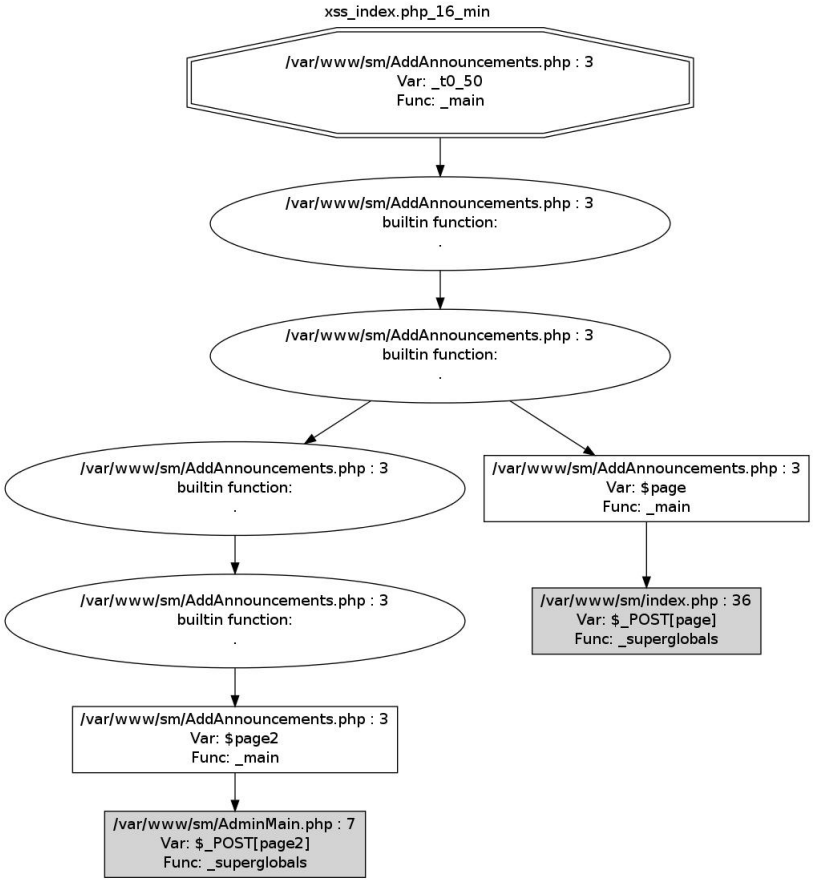
```
public void semester(){
  Vulnerabilities.selectInputVulnerability(tester,"registration","Add","semester");
      tester.assertMatch("Add New Attendance Record");
      tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 15: jwebunit test code for *semester*

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

Vulnerability 16

Brief Analysis



File: AddAnnouncements.php

VARIABLE	RESULT
page	true
page2	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,"Announcements",0);
  tester.assertMatch("Manage Announcements");
}
```

Listing 16: prepare function

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

Listing 17: cleanup function

page

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

Listing 18: jwebunit test code for *page*

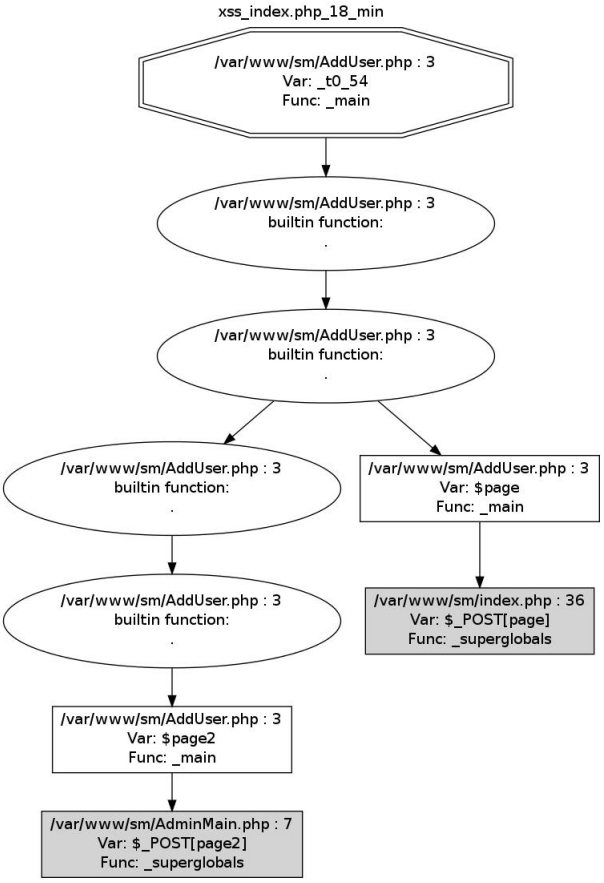
page2

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

Listing 19: jwebunit test code for *page2*

Vulnerability 18

Brief Analysis



File: AddUser.php

VARIABLE	RESULT
page	true
page2	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,"Users",0);
  tester.assertMatch("Manage Users");
}
```

Listing 20: prepare function

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

Listing 21: cleanup function

page

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

Listing 22: jwebunit test code for *page*

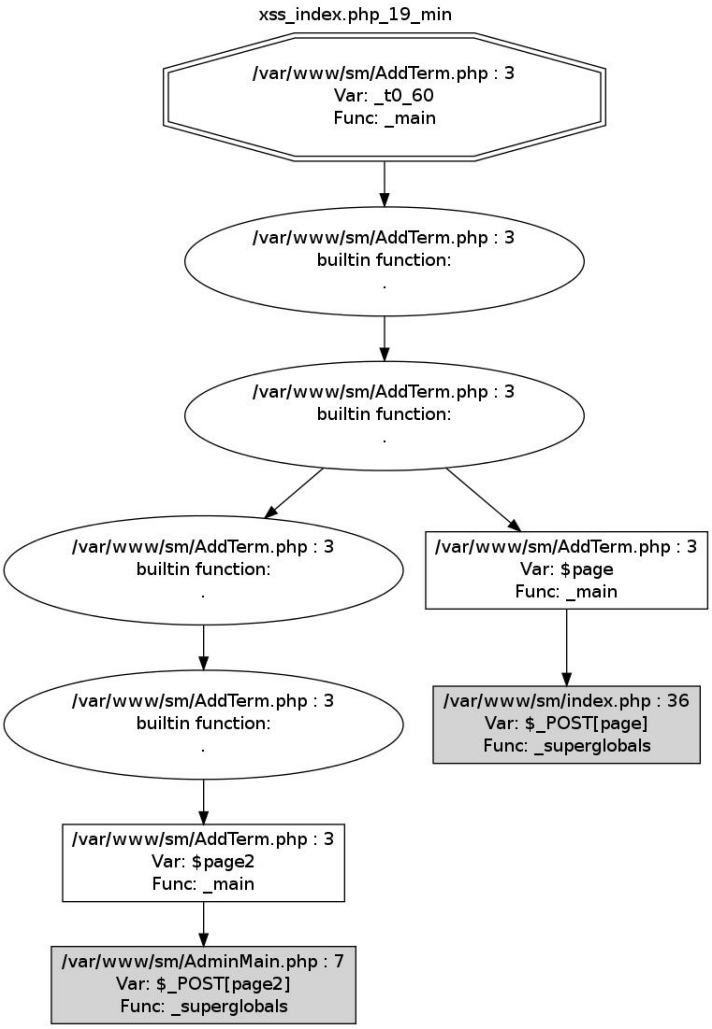
page2

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

Listing 23: jwebunit test code for *page2*

Vulnerability 19

Brief Analysis



File: AddTerm.php

VARIABLE	RESULT
page	true
page2	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");
}
```

Listing 24: prepare function

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

Listing 25: cleanup function

page

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

Listing 26: jwebunit test code for *page*

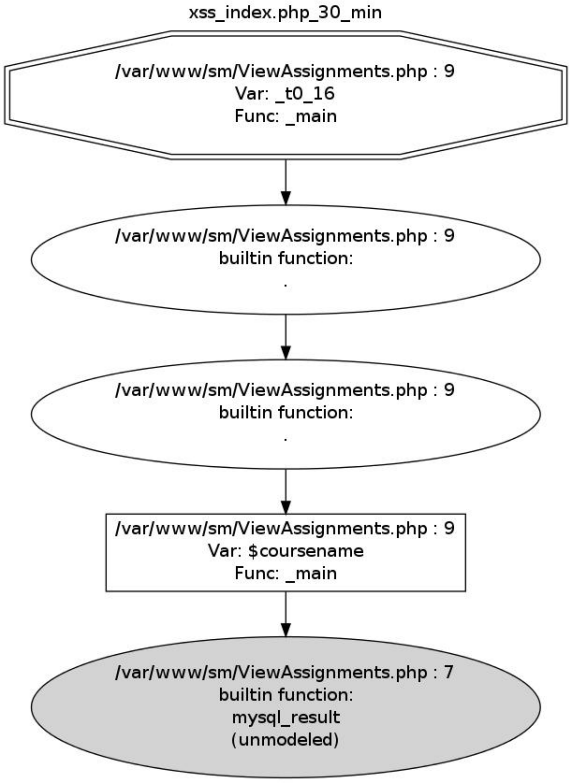
page2

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

Listing 27: jwebunit test code for *page2*

Vulnerability 30,31

Brief Analysis



File: ViewAssignments.php

VARIABLE	RESULT
page	true
page2	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 28: prepare function

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

Listing 29: cleanup function

page

```
public void page(){
  Vulnerabilities.page(tester,"student",null);
    Functions.click(tester,"Assignments",0);
    tester.assertMatch("View Assignments");
    tester.assertMatch("verifica di prova");
      tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 30: jwebunit test code for *page*

page2

```
public void page2(){
  Vulnerabilities.page2Link(tester,"student","Assignments","document.student.submit();");
    tester.assertMatch("View Assignments");
    tester.assertMatch("verifica di prova");
      tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 31: jwebunit test code for *page2*

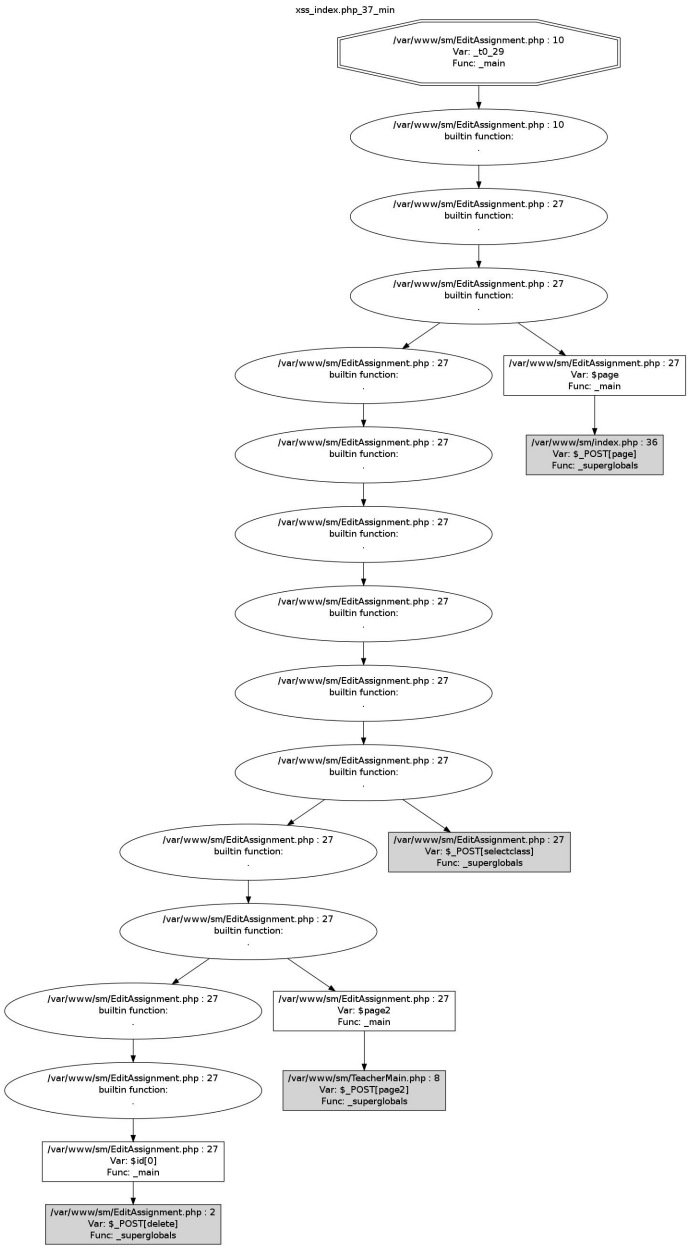
```
public static void page2Link(WebTester tester,String formName,String linkName,String
    hrefValue){
  IElement page2 = tester.getElementByXPath("//form[@name='" + formName + "']//input[@name
      ='page2']");
  IElement link = tester.getElementByXPath("//a[text()='" + linkName + "']");
  link.setAttribute("href","javascript: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

Listing 32: 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.

Vulnerability 37

Brief Analysis



File: EditAssignment.php

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 33: 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 34: cleanup function

page

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

Listing 35: jwebunit test code for *page*

page2

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

Listing 36: jwebunit test code for *page2*

selectclass

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

Listing 37: jwebunit test code for *selectclass*

delete

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

Listing 38: jwebunit test code for *delete*

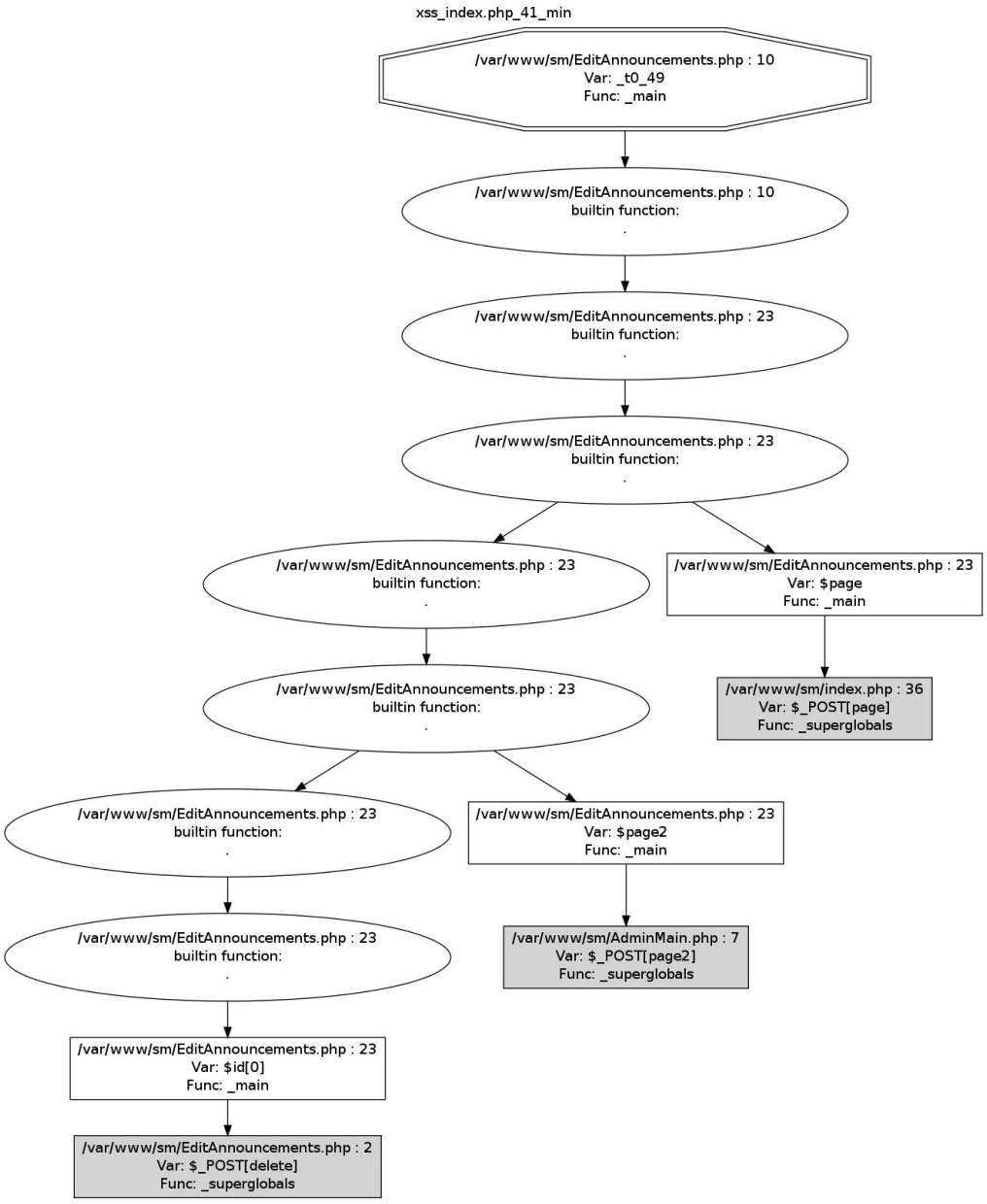
```
public static void delete(WebTester tester,String formName,String buttonName,String
    checkBoxText){
  IElement myCheckBox = tester.getElementByXPath("//td[text()='" + checkBoxText
      + "']/..//input[@type='checkbox']");
  String oldValue = myCheckBox.getAttribute("value");
  myCheckBox.setAttribute("value",oldValue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

Listing 39: 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 41

Brief Analysis



File: EditAnnouncement.php

VARIABLE	RESULT
page	true
page2	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,"admin");
  Functions.click(tester,"Announcements",0);
  tester.assertMatch("Manage Announcements");
  IElement myCheckbox = tester.getElementByXPath("//td[text()='announcement2']/..//input[
      @type='checkbox']");
  tester.setWorkingForm("announcements");
  tester.checkCheckbox("delete[]",myCheckbox.getAttribute("value"));
}
```

Listing 40: prepare function

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

Listing 41: cleanup function

page

```
public void page(){
  Vulnerabilities.page(tester,"announcements","Edit");
  tester.assertMatch("Edit Announcement");
  tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 42: jwebunit test code for *page*

page2

```
public void page2(){
  Vulnerabilities.page2(tester,"announcements","Edit");
  tester.assertMatch("Edit Announcement");
  tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 43: jwebunit test code for *page2*

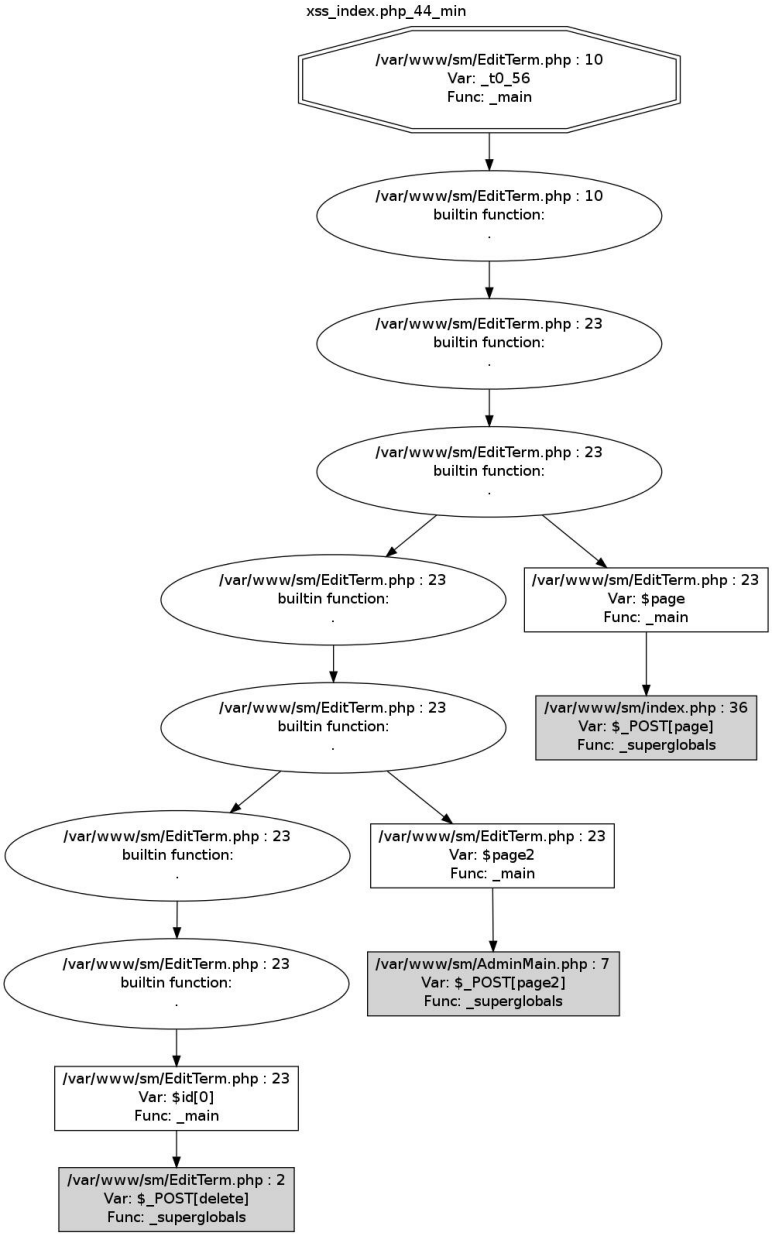
delete

```
public void delete(){
  Vulnerabilities.delete(tester,"announcements","Edit","announcement2");
  tester.assertMatch("EditAnnouncement.php: Unable to retrieve");
  tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 44: jwebunit test code for *delete*

Vulnerability 44

Brief Analysis



File: EditTerm.php

VARIABLE	RESULT
page	true
page2	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,"admin");
  Functions.click(tester,"Terms",0);
  tester.assertMatch("Manage Terms");
  IElement myCheckbox = tester.getElementByXPath("//td[text()='2012-2013']/..//input[@type
      ='checkbox']");
  tester.setWorkingForm("terms");
  tester.checkCheckbox("delete[]",myCheckbox.getAttribute("value"));
}
```

Listing 45: prepare function

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

Listing 46: cleanup function

page

```
public void page(){
  Vulnerabilities.page(tester,"terms","Edit");
  tester.assertMatch("Edit Term");
  tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 47: jwebunit test code for *page*

page2

```
public void page2(){
  Vulnerabilities.page2(tester,"terms","Edit");
  tester.assertMatch("Edit Term");
  tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 48: jwebunit test code for *page2*

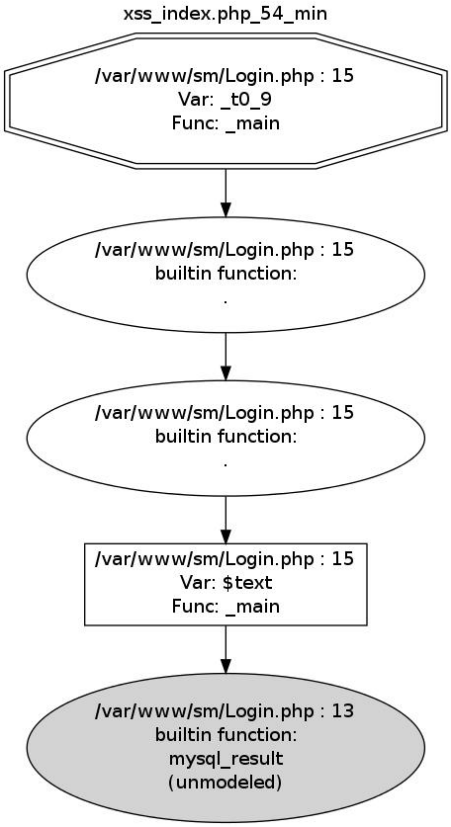
delete

```
public void delete(){
  Vulnerabilities.delete(tester,"terms","Edit","2012-2013");
  tester.assertMatch("EditTerm.php: Unable to retrieve");
  tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 49: jwebunit test code for *delete*

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");
}
```

Listing 50: 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 51: cleanup function

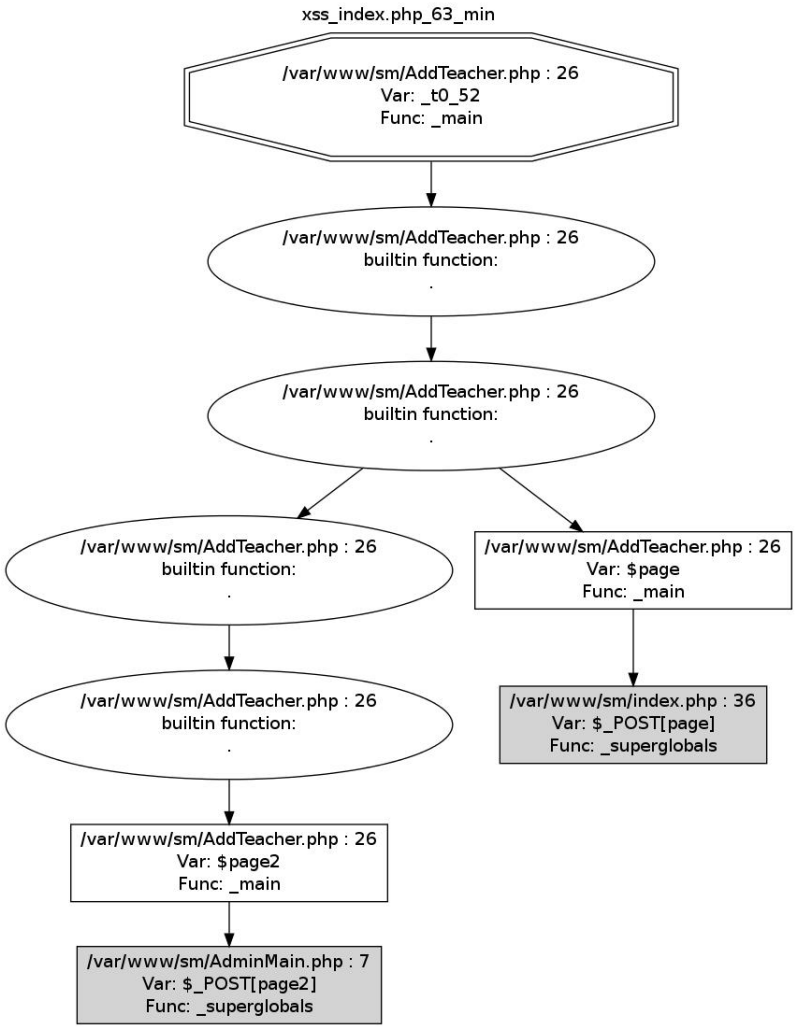
page

```
public void siteText(){
      oldValue = tester.getElementByXPath("//textarea [@name='sitetext']").getTextContent
          ();
      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 52: jwebunit test code for *text*

Vulnerability 63

Brief Analysis



File: ViewAssignments.php

VARIABLE	RESULT
page	true
page2	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,"Teachers",0);
  tester.assertMatch("Manage Teachers");
}
```

Listing 53: prepare function

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

Listing 54: cleanup function

page

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

Listing 55: jwebunit test code for *page*

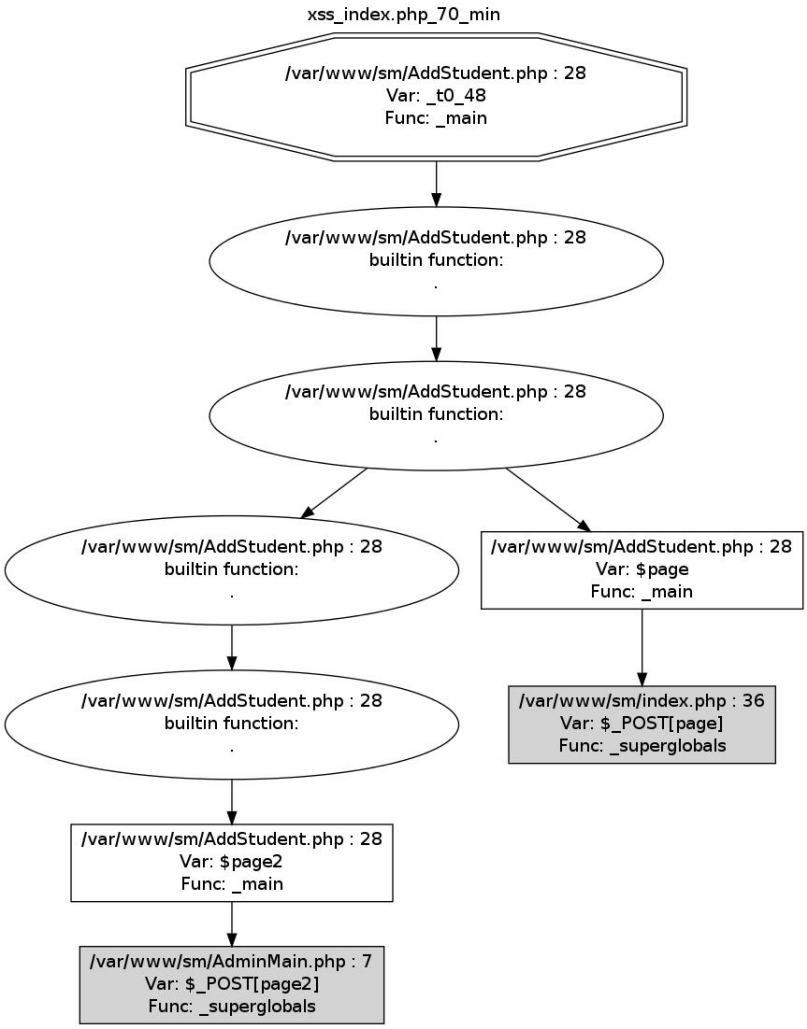
page2

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

Listing 56: jwebunit test code for *page2*

Vulnerability 70

Brief Analysis



File: ViewAssignments.php

VARIABLE	RESULT
page	true
page2	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,"Students",0);
  tester.assertMatch("Manage Students");
}
```

Listing 57: prepare function

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

Listing 58: cleanup function

page

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

Listing 59: jwebunit test code for *page*

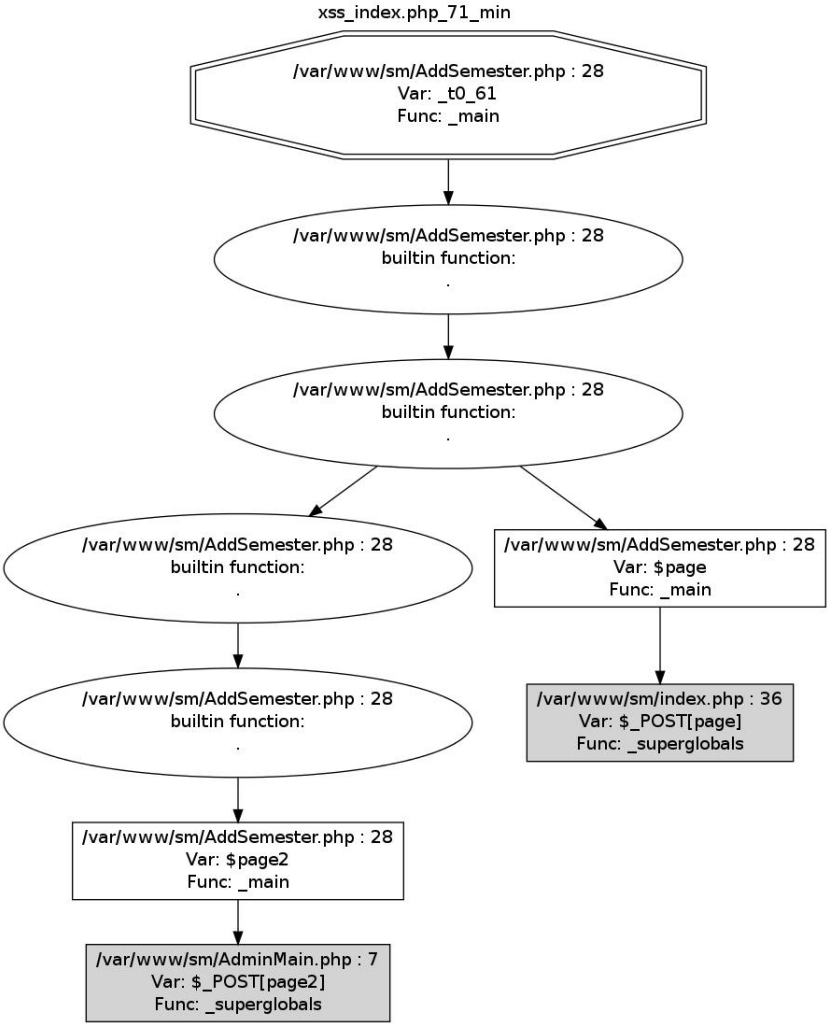
page2

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

Listing 60: jwebunit test code for *page2*

Vulnerability 71

Brief Analysis



File: ViewAssignments.php

VARIABLE	RESULT
page	true
page2	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,"Semesters",0);
  tester.assertMatch("Manage Semesters");
}
```

Listing 61: prepare function

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

Listing 62: cleanup function

page

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

Listing 63: jwebunit test code for *page*

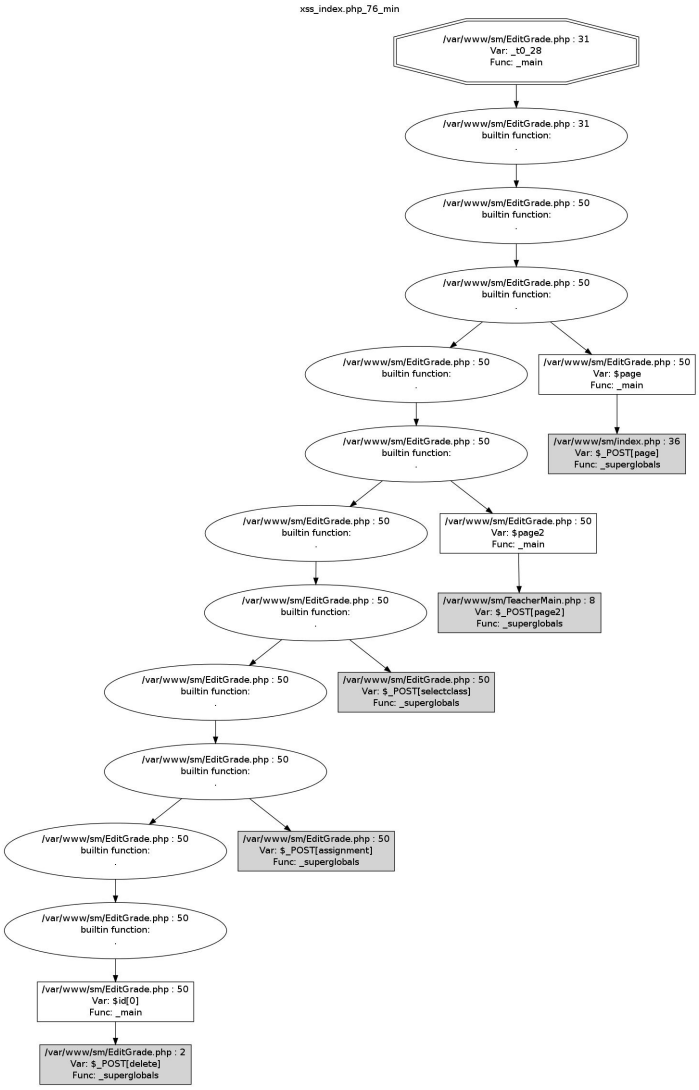
page2

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

Listing 64: jwebunit test code for *page2*

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 65: prepare function

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

Listing 66: cleanup function

page

```
public void page(){
  Vulnerabilities.page(tester,"grades","Edit");
  tester.assertMatch("Edit Grade");
  tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 67: jwebunit test code for *page*

page2

```
public void page2(){
  Vulnerabilities.page2(tester,"grades","Edit");
  tester.assertMatch("Edit Grade");
  tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 68: jwebunit test code for *page2*

selectclass

```
public void selectclass(){
  Vulnerabilities.selectclass(tester,"grades","Edit");
      tester.assertMatch("Edit Grade");
      tester.assertLinkNotPresentWithText("malicious");
}
```

Listing 69: jwebunit test code for *selectclass*

assignment

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

Listing 70: 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 71: 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.

delete

```
  public void delete(){
    tester.assertMatch("Dewey Duck");
3   Vulnerabilities.delete(tester,"grades","Edit","Harry Potter");
    tester.assertMatch("EditGrade.php: Unable to retrieve");
    tester.assertLinkNotPresentWithText("malicious");
6 }
```

Listing 72: jwebunit test code for *delete*

Vulnerabilities(*)

Brief Analysis

VARIABLE	AFFECTED PAGES(*)	RESULT
page	all	true
page2	all	positive
selectclass	11,37,76,87,89,165,180,181,183,194,200,201,309,316	positive
student	13,142,194	positive
semester	13	positive
delete	37,41,44,76,85,111,115,149,161	positive
assignment	76	positive
onpage	146,183,257,260,268,273,283,288,293,309,320	positive

(*) 11: AddAssignment.php — 13: AddAttendance.php —16: AddAnnouncements.php — 18: AddUser.php — 19: AddTerm.php — 37: EditAssignment.php — 41: EditAnnouncements.php — 44: EditTerms.php — 63: AddTeacher.php — 70: AddStudent.php — 71: AddSemester.php — 76: EditGrade.php — 85: EditSemester.php — 87/88: ViewClassSettings.php — 90: ViewStudents.php — 93: AddParent.php — 111: EditTeacher.php — 115: EditStudent.php — 126: ViewCourses.php — 138: StudentViewCourses.php — 141: AddClass.php — 142: ParentViewCourses.php — 146/147/148: ViewAnnoucements.php — 149: EditUser.php — 161: EditParent.php — 165: StudentMain.php — 180: TeacherMain.php — 181: ViewStudents.php — 183/184: ViewAssignments.php — 186/241: AdminMain.php — 191: DeficiencyReport.php — 194: ParentMain.php — 200/201: ViewGrades.php — 212: PointsReport.php — 130: VisualizeClasses.php — 238: VisualizeRegistration.php — 239: EditClasses.php — ManageAnnouncements.php — 260: ManageTerms.php — 268. ManageTerms.php — 272: ManageAttendance.php — 273: ManageTeachers.php — ManageUsers.php — 288: ManageParents.php — 293: ManageStudents.php — 299: Registration.php — 309: ManageAssignments.php — 316: ManageGrades.php — 320: ManageClasses.php

Explanation

These parameters are used to process the web-application flow through. The problem is that the page which receive these values through a POST, do not validate them and they are put inside a the *value* of a *input* element.

page

```
<input type='hidden' name='page' value='$page'>
```

Listing 73: AddAssignment.php load of *page*

page2

```
<input type='hidden' name='page2' value='$page2'>
```

Listing 74: AddAssignment.php load of *page2*

selectclass

```
<input type='hidden' name='selectclass' value='$_POST[selectclass]' />
```

Listing 75: AddAssignment.php load of *selectclass*

student

```
<input type='hidden' name='student' value='$_POST[student]' />
```

Listing 76: AddAttendance.php load of *student*

semester

```
<input type='hidden' name='semester' value='$_POST[semester]' />
```

Listing 77: AddAttendance.php load of *student*

delete

```
$id = $_POST["delete"];
```

Listing 78: EditAssignment.php load of *delete*

```
<input type='hidden' name='assignmentid' value='$id[0]'>
```

Listing 79: EditAssignment.php read of *delete*

assignment

```
<input type='hidden' name='assignment' value='$_POST[assignment]' />
```

Listing 80: EditGrade.php load of *assignment*

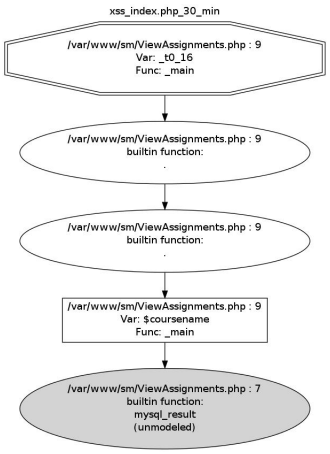
onpage

```
<input type='hidden' name='onpage' value='$_POST[onpage]'>
```

Listing 81: ViewAnnouncements.php load of *onpage*

Vulnerabilities 30,31,207

Brief Analysis



Files: ViewAssignments.php,ManageAssignments.php

VARIABLE	RESULT
coursename	false positive
assignment[5]	positive

Explanation

coursename

In ManageClasses we have the 3 queries which do *insertion* and one which do an *update* inside the database table:

```
$query = mysql_query("INSERT INTO courses VALUES('', '$_POST[semester]', '$termid', '
    $_POST[title]', '$_POST[teacher]', '$_POST[sectionnum]', '$_POST[roomnum]', '$_POST[
    periodnum]','','','','','','','','','','$dotw','$_POST[substitute]','')")
or die("ManageClasses.php: Unable to insert new class - " . mysql_error());
```

Listing 82: ManageClasses.php insert1 of *coursename*

```
$query = mysql_query("INSERT INTO courses VALUES('', '$_POST[semester]', '$termid', '
    $_POST[title]', '$_POST[teacher]', '$_POST[sectionnum]', '$_POST[roomnum]', '$_POST[
    periodnum]','','','','','','','','','','$dotw','$_POST[substitute]', '')")
or die("ManageClasses.php: Unable to insert new class - " . mysql_error());
```

Listing 83: ManageClasses.php insert2 of *coursename*

```
$query = mysql_query("INSERT INTO courses VALUES('', '$_POST[semester2]', '$termid', '
    $_POST[title]', '$_POST[teacher]', '$_POST[sectionnum]', '$_POST[roomnum]', '$_POST[
    periodnum]','','','','','','','','','','$dotw','$_POST[substitute]', '')")
or die("ManageClasses.php: Unable to insert new class - " . mysql_error());
```

Listing 84: ManageClasses.php insert3 of *coursename*

```
$query = mysql_query("UPDATE `courses` SET `coursename`='$_POST[title]', `teacherid`='
    $_POST[teacher]', `semesterid`='$_POST[semester]', `sectionnum`='$_POST[sectionnum]',
    `roomnum`='$_POST[roomnum]', `periodnum`='$_POST[periodnum]', `dotw`='$dotw', `
    substituteid`='$_POST[substitute]' WHERE `courseid`='$_POST[courseid]' LIMIT 1")
or die("ManageClasses.php: Unable to update the class information - ".mysql_error());
```

Listing 85: ManageClasses.php update of *coursename*

No sanitization is made over the $_POST[title], so an xss can be injected.
In *ViewAssignments.php* and *ManageAssignments.php* we have the read of the tainted value:

```
$query = mysql_query("SELECT coursename FROM courses WHERE courseid = '$_POST[selectclass]'
    ") or die("ManageAssignments.php: Unable to get the course name - ".mysql_error());
$coursename = mysql_result($query,0);
```

Listing 86: ViewAssignment.php load of *coursename*

```
$query = mysql_query("SELECT coursename FROM courses WHERE courseid = '$_POST[selectclass]'
    ") or die("ManageAssignments.php: Unable to get the course name - ".mysql_error());
$coursename = mysql_result($query,0);
```

Listing 87: ManageAssignments.php load of *coursename*

So far it seems legit to say that an XSS attack can be done over this vulnerability, having a look to the forms which are the source of the insertions and updates, we can see that a limit of 20 chars is set for the field:

```
<td><input type='text' name='title' maxlength='20' /></td>
```

Listing 88: AddClass.php form field

```
<td><input type='text' name='title' maxlength='20' value='$class[0]' /></td>
```

Listing 89: EditClass.php form field

Anyway we know that such restriction can be by-passed by intercepting the requests and modify them on-the-fly. However a closer look to the database structure denies our expectation and explains why this result is a false positive:

```
coursename varchar(20) NOT NULL default '',
```

Listing 90: Sql structure of the field

In fact, the filed coursename is restricted to 20 chars even in the database structure and so any larger string is going to be truncated to that size. So no XSS can be injected because the shorter one that we know (*¡script¿alert(")¡/script¿*) is **26** chars long.

assignment[5]

The source of this vulnerability is the value of the column *assignmentinformation* of the table *assignments*. The page *ManageAssignments.php* can do an insertion inside that table and the value passed is not validated:

```
$query = mysql_query("INSERT INTO assignments VALUES('', '$_POST[selectclass]', '$ids
    [0]', '$ids[1]', '$_POST[title]', '$_POST[total]', '$_POST[assigneddate]', '$_POST[
    duedate]', '$_POST[task]')")
or die("ManageAssignments.php: Unable to insert new assignment - " . mysql_error());
```

Listing 91: ManageAssignments.php store of *assignment[5]*

Vulnerabilities 30,31,207 continued on next page...

Later on, the injected value can be read from the *ViewAssignments.php* page and no validation is done:

```
$query = mysql_query("SELECT assignmentid, title, totalpoints, assigneddate, duedate,
    assignmentinformation FROM assignments WHERE courseid = $_POST[selectclass] ORDER BY
    assigneddate DESC")
   or die("ManageAssignments.php: Unable to get a list of assignments - ".mysql_error());
$row = 0;
$actualrow = 0;
while($assignment = mysql_fetch_row($query))
```

Listing 92: ViewAssignments.php load query for *assignment[5]*

```
print("<tr class='".( $row%2==0 ? "even" : "odd" )."'>
 <td align='left' style='padding-left: 20px;'>$assignment[1]</td>
 <td style='text-align: left;'>$assignment[5]</td>
 <td>$assignment[2]</td>
 <td>$assignment[3]</td>
 <td>$assignment[4]</td>
</tr>");
}
```

Listing 93: ViewAssignments.php variable read of *assignment[5]*

Vulnerability 92

Brief Analysis

File: ManageSchoolInfo.php

VARIABLE	RESULT
page	false positive
page2	false positive
numperiods	false positive
numsemesters	false positive
phone	false positive
address	positive
schoolname	false positive

Explanation

The analysis of the section *Vulnerabilities*$^{(*)}$ can also fit for *page* and *page2*. Moreover, *Vulnerabilities 2,3,4,6,10,53* explains the result over *schoolname*.

numperiods,numsemesters

```
$query = mysql_query("SELECT numsemesters FROM schoolinfo")
    or die("ManageSchoolInfo.php: Unable to retrieve NumSemesters " . mysql_error());

$numsemesters = mysql_result($query,0);

$query = mysql_query("SELECT numperiods FROM schoolinfo")
    or die("ManageSchoolInfo.php: Unable to retrieve NumPeriods " . mysql_error());

$numperiods = mysql_result($query,0);
```

Listing 94: ManageSchoolInfo.php load of *numperiods* and *numsemesters*

The load of the two values is not validated and that's why the software highlight the case, moreover we have a not validated update over the table:

```
$query = mysql_query("UPDATE schoolinfo SET schoolname = \"".htmlspecialchars($_POST["
    schoolname"])."\", address = '$_POST[schooladdress]', phonenumber = '$_POST[
    schoolphone]', sitetext = '$_POST[sitetext]', sitemessage = '$_POST[sitemessage]',
    numsemesters = '$_POST[numsemesters]', numperiods = '$_POST[numperiods]', apoint = '
    $_POST[apoint]', bpoint = '$_POST[bpoint]', cpoint = '$_POST[cpoint]', dpoint = '
    $_POST[dpoint]', fpoint = '$_POST[fpoint]' where schoolname = '$schoolname' LIMIT 1 ")
    ;
```

Listing 95: header.php store of *numperiods-numsemesters-phone-address*

As happened for *coursename* on *Vulnerabilities 30,31,207*, the database schema tell us that no injection is possile, because the interested columns are set as int(3):

```
CREATE TABLE schoolinfo (
  schoolname varchar(50) NOT NULL default '',
  address varchar(50) default NULL,
  phonenumber varchar(14) default NULL,
  sitetext text,
  sitemessage text,
  currenttermid int(11) default NULL,
  numsemesters int(3) NOT NULL default '0',
  numperiods int(3) NOT NULL default '0',
  apoint double(6,3) NOT NULL default '0.000',
  bpoint double(6,3) NOT NULL default '0.000',
  cpoint double(6,3) NOT NULL default '0.000',
  dpoint double(6,3) NOT NULL default '0.000',
  fpoint double(6,3) NOT NULL default '0.000',
  PRIMARY KEY  (schoolname)
) ENGINE=MyISAM;
```

Listing 96: Sql schema for *schoolinfo*

phone

```
$query = mysql_query("SELECT phonenumber FROM schoolinfo")
     or die("ManageSchoolInfo.php: Unable to retrieve PhoneNumber " . mysql_error());

$phone = mysql_result($query,0);
```

Listing 97: ManageSchoolInfo.php load of *phone*

The load works as for the two field above, and as in that case, the result can be addressed as a *false positive* thanks to the database schema (*Listing 96*). In this case the column has type *varchar(14)*, which is more sensitive than int, but the size prevent any possible injection, because, as said in *Vulnerabilities 30,31,207*, the smaller xss that we can apply - even if useless - is about 26 chars long.

address

```
$query = mysql_query("SELECT address FROM schoolinfo")
     or die("ManageSchoolInfo.php: Unable to retrieve School Address " . mysql_error());

$address = mysql_result($query,0);
```

Listing 98: ManageSchoolInfo.php load of *address*

This time the database schema cannot help us, because the field type is *varchar(50)*, so an injection is possible. However it seems that the only page which reads from that field is the page itself - ManageSchoolInfo.php -, which puts the content as value in a form field. Still I will consider it as a positive result, because if in a future deployment the value will be displayed in an another page, the vulnerability will became exploitable.

Vulnerability 92 continued on next page...

Testing Code

address

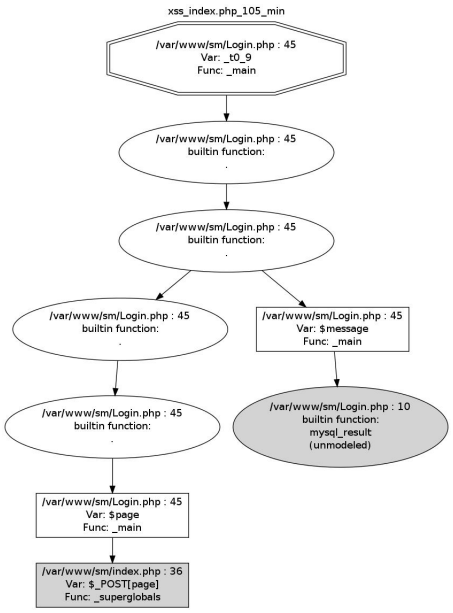
```
        tester = new WebTester();
        tester.setBaseUrl("http://localhost/sm/");
    }

    @AfterClass
    public static void tearDownClass() {
        tester.beginAt("index.php");
        Functions.login(tester, "admin");
        tester.submit();
        tester.clickLinkWithText("School");
        tester.assertMatch("Manage School Information");
```

Listing 99: jwebunit test code for *address*

Vulnerability 105

Brief Analysis



File: Login.php

VARIABLE	RESULT
message	positive
page	positive

Explanation

The analysis of the section *Vulnerabilities*$^{(*)}$ can fit for *page*.

message

```
$query = mysql_query("select sitemessage from schoolinfo");

$message = mysql_result($query,0);
```

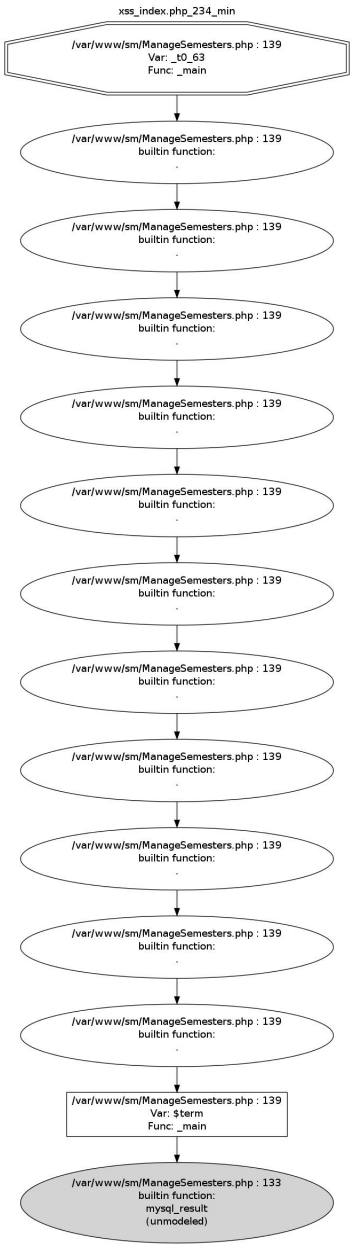
Listing 100: Login.php load of *sitemessage*

```
$query = mysql_query("UPDATE schoolinfo SET schoolname = \"".htmlspecialchars($_POST["
    schoolname"])."\", address = '$_POST[schooladdress]', phonenumber = '$_POST[
    schoolphone]', sitetext = '$_POST[sitetext]', sitemessage = '$_POST[sitemessage]',
    numsemesters = '$_POST[numsemesters]', numperiods = '$_POST[numperiods]', apoint = '
    $_POST[apoint]', bpoint = '$_POST[bpoint]', cpoint = '$_POST[cpoint]', dpoint = '
    $_POST[dpoint]', fpoint = '$_POST[fpoint]' where schoolname = '$schoolname' LIMIT 1 ")
    ;
```

Listing 101: header.php store of *sitemessage*

Vulnerability 234

Brief Analysis



File: ManageSemesters.php

VARIABLE	RESULT
term	positive

Explanation

```
$query2 = mysql_query("SELECT title FROM terms WHERE termid='$smstr[1]'");
$term = mysql_result($query2,0);
```

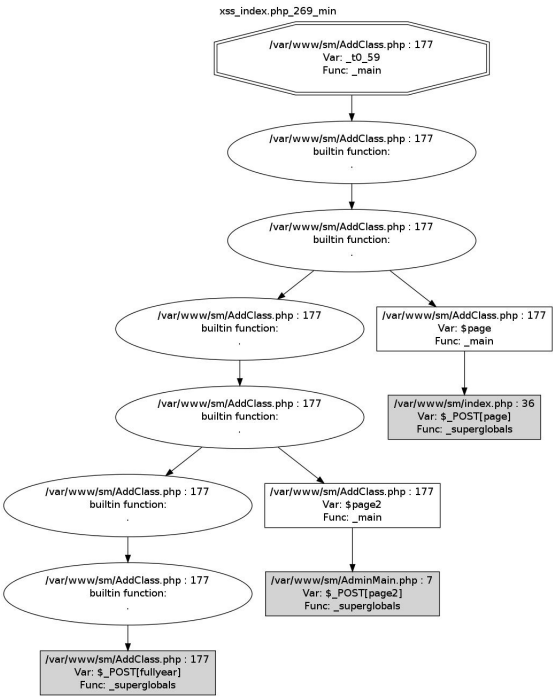
/var/www/sm/ManageSemesters.php

```
$query = mysql_query("UPDATE `terms` SET `title`='$_POST[title]', `startdate`='$_POST[
    startdate]', `enddate`='$_POST[enddate]' WHERE `termid`='$_POST[termid]' LIMIT 1")
or die("ManageTerms.php: Unable to update the term information - ".mysql_error());
```

/var/www/sm/ManageTerms.php

Vulnerability 269

Brief Analysis



File: AddClass.php

VARIABLE	RESULT
page	false positive
page2	false positive
fullyear	false positive

Explanation

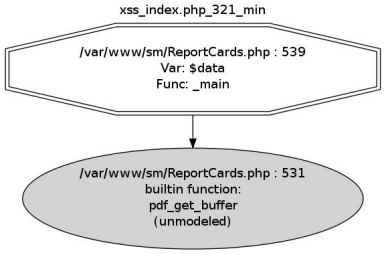
The analysis of the section *Vulnerabilities*$^{(*)}$ can also fit for *page* and *page2*.

fullyear

The parameter is just used to display a different form of insertion of the class, so no xss is possible here.

Vulnerability 321

Brief Analysis



File: ReportCards.php

VARIABLE	RESULT
data	positive

Explanation

```
$sql = mysql_query("SELECT coursename, q1points, q2points, totalpoints, aperc, bperc, cperc
    , dperc, fperc, secondcourseid, semesterid FROM courses WHERE courseid = $cid[0]
    $clause");
while($class = @mysql_fetch_row($sql))
{
```

/var/www/sm/ReportCards.php

```
pdf_show_xy($pdf, "$class[0]", 55, $start);
```

/var/www/sm/ReportCards.php

As long as seen at *Vulnerabilities 30,31,207*, *coursename* can be a injected with malicious strings which can lead to an xss vulnerability. In this case the pdf generated can contain such malicious string.