University of Trento
Security Testing, May 28, 2015

XSS vulnerability assessment and fix on Schoolmate platform

Martintoni Davide 171076

May 15, 2015

1 Purpose

For the course of Security Testing[5] we are supposed to analyze a web application in order to identify the Cross Site Scripting (XSS) vulnerability and fix it. So in order to complete this assignment I performed these tasks:

- Detect vulnerabilities
- Create a Proof of Concept for every vulnerability
- Fix vulnerabilities
- Verify fixes

In this report I will explain with details all my work on the application that is the subject of this analysis.

2 Use Case

We analyze a open source application for the management of elementary, middle and high schools. This web application is written using the PHP framework and rely on MySQL as database to store data. This application is available for free on the sourceforge website[4] and can be easily mounted on a server. For this project I have deployed on an Apache2[2] server into a linux environment that can be configured without much problems to run this server and a MySql database. So once I finished my setup and the application runs properly on my local server let's take a closer look to the application in order to understand the services that provides, the properties available and how those features are designed and coded.

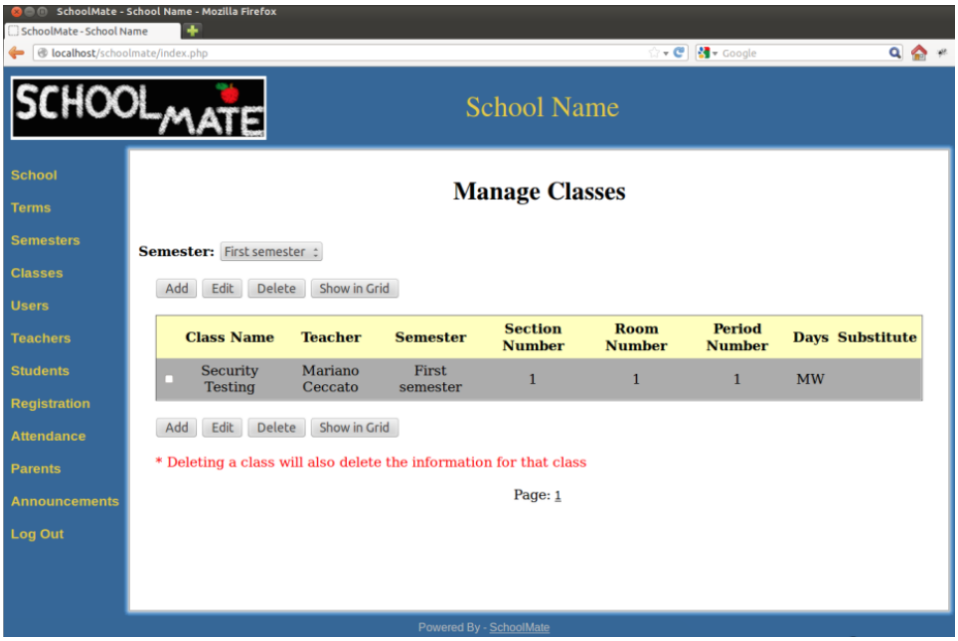

Figure 1: Screenshot of the application on the class management section of the admin

This software can be divided in four main sections that correspond to four users with different roles and permissions:

1. Administrator is the user with obviously more rights in the system. He can add, delete and modify most of the entities in the system such classes, terms, semesters, users and can also publish announcements and edit the school information. Take a look on Figure 1 that represent the page in which an administrator can manage the courses of the school.

2. Teacher: the user with this role can access to his own classes and edit some settings (e.g the grade percentage). Moreover he can add assignment to the class's students and assign grades. Furthermore he can visualize information about the school and read the announcements published by the administrator.

3. Parents can visualize all the data about the school career of the sons and also read general information on the school (e.g announcements, message of the day and other utility fields like phone number and address)

4. Student, exactly like the parents, can visualize data about his own career divided by classes. For every class there are information about assignments and grades available.

2.1 Code

This application is composed of about 70 ".php" files. The more important file is called "index.php" that contains the function used to connect to the database. Another important feature included in the index is the navigation's rule. Indeed this file take the parameters

used to navigate the pages and includes the right file based on the page request and the user session. Every user type has its own "(Type)Main.php" that create the navigation bar on the right with the features available for that role.

Every other file is a specific function of one or more user's types. For example the file "ManageClasses.php" is called by the administrator when he click on the link labelled as "Classes". Another file can be "ViewCourses.php" that is included inside "StudentViewCOurses.php" and "ParentViewCourses.php" that are called based on the type of the user logged in.

So as we said "index.php" is the more important page in the application and in fact every request is handled by this page. About the requests I have noticed that every parameter that is sent from the client to the server is embedded in a "POST" request through form's submit. But the problem with those parameters is that almost all of them are posted, used by the server and then printed into the next page without any kind of sanitization and this is the main threats to this system. This kind of vulnerability can easily be exploited in order to perform a XSS attack.

3 Technologies

In order to analyze this application, find the vulnerabilities and create a proof of concept attack I have used mainly two tools:

First I have run on the application "Pixy[3]" that is a static analysis tool that help into the detection of the web application vulnerabilities. The result of its analysis is a set of graphs with the tainted variables that it has found. Unfortunately this tool is not perfect and sometimes give in the result some tainted variables that are false positive namely they are the result of unfeasible executions or they came out of some source that the tool can't trust as safety. For example if a variable comes from the database the tool label it as tainted because it doesn't know if in the insertion's form there are some sanitizations or not.

Another important tool that I have used for this project is "JWebUnit[6]" that is a Java-based testing framework for web applications. It wraps existing testing frameworks such as "HtmlUnit" and "Selenium" with a unified, simple testing interface to allow the testing of the correctness of your web applications. JWebUnit provides high-level Java API for navigating a web application combined with a set of assertions to verify the application's correctness. So with this framework I have written my proof of concept that show how every vulnerability that I have identified is actually exploitable and therefore dangerous.

4 Analysis

Now as I said in the previous section the first job was run "Pixy" on the application. This tool provide a list of file that with the help of the "dot" command of the package "Graphviz[1]" package can be converted into graphs in format "jpg" so that are readable by a human.

As we can see in Figure 2 every graph contains some variables that are labelled as tainted, the leaf of the tree are the entry points and raising from above we can see the flow of the variable in the page. The dot operator means concatenation and in this examples the three

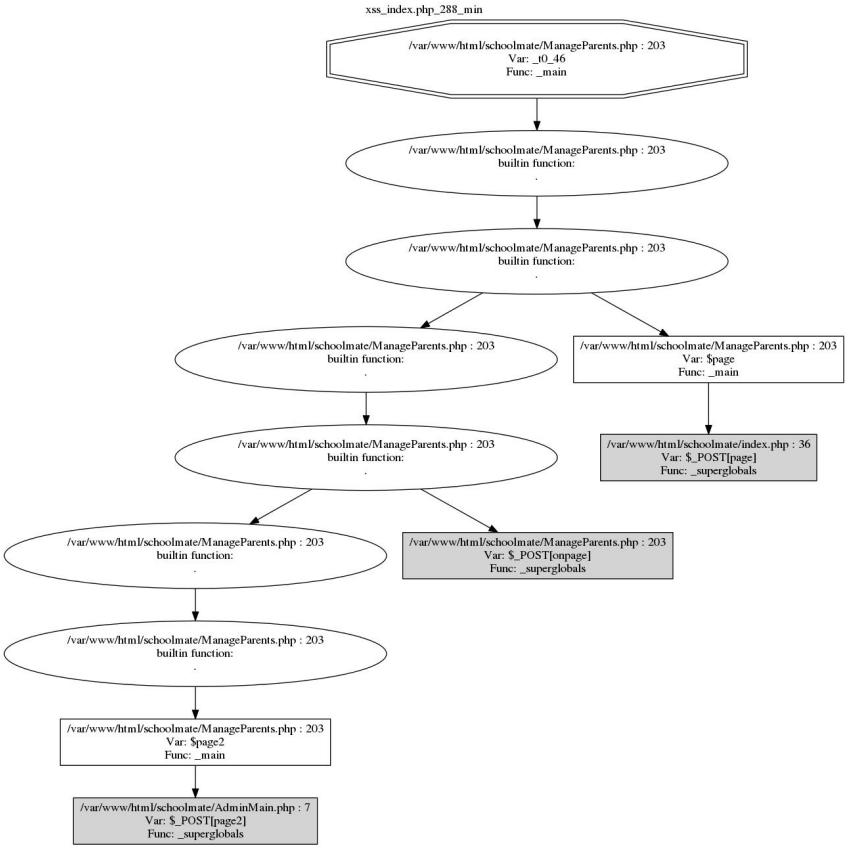

Figure 2: Example of result of pixy: ManageParents.php contains 3 tainted variables named "page", "onpage" and "page2" that have as entry point respectively index at line 36, ManagePage at line 203 and AdminMain at line 7

values starts from different pages and arrive in concatenation into the same file.

So once that I had these graphs I checked everyone looking through the pages indicated if these results are really tainted or if they are false positives. All my checks and decisions are summarized in the table that you can see in the Appendix A. In this file I have written for each variable from which graph is tainted (id column), which files generate it and where is propagated (e.g AddTerm:34 means that it's present at line 34 of the file AddTerm.php). Then I have decided if it's tainted or not and explained why. If it's really tainted I have also tried to exploit it manually and so I have write some annotations in order to remember how I have done it.

5 Proof of Concept Automatization

In order to prove that the vulnerability that I found are really dangerous I have build a project in Java with the framework JWebUnit. So for every tainted variable that I have

build a function that exploit it and inject a link into the page. This little injection is the prove that a cross site scripting attack can occur based on this vulnerability.

Another important feature that these tests provide is that once I have fixed the vulnerabilities using a sanitize function I can run all the tests again and they will highlight if my fix works.

Now let's take a look on the java project that I create to tests the application: I have written a test file for each php file that compose the platform and I divided them into packages badsed on which user's type I have used to perform the operations. For example as we can see in the Listing 1 this is the Java class that test each XSS vulnerability that allows to inject and print some malicious code into the "EditClass.php" file. Into this file we have three main type of function that are labeled with three different annotations. The first function is named "setup" and as we can see is labeled with the "@Before" annotation. This function executes some operations that are needed before every test. In this case we use this "setup" to sign in into the platform with the admin user named "schoolmate".

The most important functions are those labeled with the annotation "@Test". We have five functions of this type and each one is used to prove one vulnerability. The name of the function recall the name of the tainted variable that is exploited. For example the page function, starting from where the setup function have leave the execution, namely after the login, inject in the hidden field that contains the variable "page" a malicious string that contains a link. After the manipulation of the input field we post this parameter with the others navigating towards the "EditClass" page. Now that we arrived in the desired page we check with an assertion if we actually succeed into the injection of a malicious link into this page.

```
1  public class EditClass {
2
3    private WebTester tester;
4
5    @Before
6    public void setup(){
7      tester = new WebTester();
8      tester.setBaseUrl(Manager.getBaseUrl());
9      tester.beginAt("index.php");
10     tester.setTextField("username", "schoolmate");
11     tester.setTextField("password", "schoolmate");
12     tester.submit();
13   }
14
15   @Test
16   public void page(){
17     tester.assertMatch("Manage Classes");
18     tester.setWorkingForm("classes");
19         tester.setHiddenField("page","1'> <a href=www.unitn.it>malicious link
               </a> <br  '");
20         tester.checkCheckbox("delete[]","1");
21         tester.clickButtonWithText("Edit");
22         tester.assertMatch("Edit Class");
23         tester.assertLinkPresentWithText("malicious link");
24   }
25
26   @Test
```

```
27    public void page2(){
28      tester.assertMatch("Manage Classes");
29      tester.setWorkingForm("classes");
30          tester.setHiddenField("page2","11'><a href=www.unitn.it>malicious
                link</a><br ");
31          Manager.addNewSubmitButton("/html//form[@name=\"classes\"]",tester);
32          tester.checkCheckbox("delete[]","1");
33          tester.submit();
34          tester.assertMatch("Edit Class");
35          tester.assertLinkPresentWithText("malicious link");
36    }
37
38    @Test
39    public void id(){
40      tester.assertMatch("Manage Classes");
41      tester.setWorkingForm("classes");
42      tester.getElementByXPath("//input[@type='checkbox' and @value='1']").
            setAttribute("value", "1 -- '><a href=www.unitn.it>malicious link</a><
            br ");
43      tester.checkCheckbox("delete[]");
44      tester.clickButtonWithText("Edit");
45      tester.assertMatch("Edit Class");
46        tester.assertLinkPresentWithText("malicious link");
47    }
48
49    @Test
50    public void id_MysqlError(){
51      tester.assertMatch("Manage Classes");
52      tester.setWorkingForm("classes");
53      tester.getElementByXPath("//input[@type='checkbox' and @value='1']").
            setAttribute("value", "1><a href=www.unitn.it>malicious link</a><br ")
            ;//without closing the sql param
54      tester.checkCheckbox("delete[]");
55      tester.clickButtonWithText("Edit");
56          tester.assertMatch("EditClass.php: Unable to retrieve");
57          tester.assertLinkPresentWithText("malicious link");
58    }
59
60    @Test
61    public void coursename() {
62      tester.assertMatch("Manage Classes");
63      String key = "-1";
64      StoredAttackCreationUtility sacu = new StoredAttackCreationUtility();
65      try {
66        key = sacu.createCoursename("'/><a href=ww>l</a>");
67      } catch (SQLException e) {
68        System.err.println(e + " class creation faild");
69      }
70      tester.clickLinkWithExactText("Classes");
71      tester.assertMatch("Manage Classes");
72      tester.setWorkingForm("classes");
73      tester.checkCheckbox("delete[]", key);
74      tester.clickButtonWithText("Edit");
75      tester.assertMatch("Edit Class");
76      tester.assertLinkPresentWithText("l");
77    }
78
79    @After
80    public void restore() {
81      StoredAttackCreationUtility sacu = new StoredAttackCreationUtility();
```

```
82      try {
83        sacu.restoreCourses();
84      } catch (SQLException e) {
85        System.err.println(e + " restore faild");
86      }
87    }
88
89  }
```

Listing 1: Code for a page that contains XSS attacks and also Stored XSS attacks

Another type of attack is the Stored XSS: instead of inject the code into a page and posting with a form towards the next page we can store some malicious code into the database. So when we arrive in a page that perform a query on the database and then shows the results without check and sanitize we have an injection of malicious code.

For example the "coursename" function at line 61 prove this type of attack. As we can see this feature create a new course with a malicious name using an utility that I have written. Once that this new class is stored into the database this function navigate into the EditClass page targeting this course and check if the link is present.

The "StoredAttack¡something¿Utility" is a set of classes located into the "util" package that I have written that creates the new entity in database using the pages of the application an not a simple sql query. Using the "jdbc" library could be easier and faster but using the pages I can test if once I have fixed the vulnerability is safe to inject also malicious code. In fact once I have create a sanitization for the input before submitting to the database this utility will fail instead if I planted the malicious code directly into the database with a sql statement those sanitizations would be skipped.

Another type of function is the call named "restore" that we can see at the end of the file. This feature labeled with the annotation "@After" is called after the execution of every test. So it's particularly useful to restore the database to the original form after a Stored XSS attacks that have injected some malformed data. In fact as we can notice at line 83 we have another function of my utility class that is supposed to roll back the database to a safety state.

The last type of attack that I have found is the cross site scripting attack based on the information leakage due to a bad error handling. As we can see at line 50 of the Listing 1 we have an injection into an hidden field of a malformed "id". This string is used in the "EditClass.php" file into a query with a simple concatenation. But this injected code is not a valid sql statement. SO when I post this page and this parameter is used in the query we get an error that is checked with an assertion and in this error we can find also the wrong sql statement. If the parameters are not sanitize in the sql error we will find also my injected link printed.

6 Results

Based on the "Pixy"'s results I have identified more or less 170 vulnerability that I have exploited with the "JWebUnint" project. Most of them are quite similar through every page. For example the "page" variable that is used for the navigation is present in every file of the application and is always tainted. In fact this value is posted by every page without any sanitization and then is used by the index.php to redirect the program flow

towards the right page. After this redirect the value will be printed again in the current page always without any check. So it's easy to inject into this hidden fields that are used to print/post this value a malicious string that can be interpreted by the index as a valid value. For example the string *1'>malicious link<br '* can be injected and the index page will use it without problem to redirect the program. As we can see after the 1 we have a quote and a strict inequality symbol. When we post this string those symbols will be interpreted as a 1 in the input field concatenated with a malicious link. So the program works fine but with an added html element.

About the Pixy results as we can see in the attached table (look ate the "True?" column) I found only four false positive on about 170 labeled variables. These values are actually safe even if Pixy labeled them as tainted. Instead while I was looking into the application's files I noticed that there are some exploitable vulnerability that the tool did not find. I have added them (more or less 20) to my JWebUnit project and I have listed them into the second appendix file. Most of them are Stored XSS vulnerability based on the name of courses, semesters and terms. Perhaps those vulnerability are not reported because those fields in the database are at most twenty characters long an so they are unlikely to be used for a real attacks. However I have proved that they can be used to inject small amount of data and this for me is enough to state that they need a fix almost as the others vulnerability.

7 Fix them all

Now that I have evaluate all the Pixy's graphs and I have automated every test to prove that they are all actually exploitable I can proceed to fix this application. The first thing that I noticed when I saw for the first time this platform is the structure on which is builded. When we navigate through the available pages the url does not change. After brief analysis is easy to see that every file except the "Login.php" are included into the application as part of the index page. So we can say that the application consists of a page that use all the other files to compose itself.

But this type of structure united with the fact that every parameter is passed to the server side within a post http request can be exploited to sanitize everything in one single place. The php framework handle the post parameters with an array called "$_POST". So when we make a call to a page and we post some parameters to the server the index page receive this array in which there are every variable that it needs to include the right files and to query the right data. So before that the "index.php" can even take a look to all those values I have placed a sanitization feature that parse every element the array applying the php function "htmlspecialchars()" that encodes strings that contain dangerous characters that can be interpreted as HTML into safety strings.

So now the only thing that I have left to do is to fix the Login.php file where I have sanitize with the same php function into the code the three tainted variables founded here by Pixy.

8 Conclusion

After all the fixes I have run all my JWebUnit tests again on the new version of the application. As I have expected all these cases failed and so I can state that my sanitizations really works. I have also tried to run Pixy again but the tool gave to me exactly the same results as before except for the Login.php where now there are not reported vulnerabilities. Probably the tool doesn't take care of my function that sanitize all the values contained into the post array and so keep to label all the variables as tainted. But based on my tests I can now assume that all those results can be categorized as false positives.

References

[1] Arif Bilgin. Graphviz project. `http://www.graphviz.org`, 2015. [Online; last read 06-May-2015].

[2] The Apache Software Foundation. The apache http server project. `http://httpd.apache.org`, 2015. [Online; last read 06-May-2015].

[3] Oliver Klee Jenad Jovanonic. Pixy project on github. `https://github.com/oliverklee/pixy`, 2015. [Online; last read 06-May-2015].

[4] mrmunkey22. Download schoolmate webapp. `http://sourceforge.net/projects/schoolmate/`, 2013. [Online; last read 06-May-2015].

[5] Andrea Avancini Paolo Tonella, Mariano Ceccato. Security testing unitn. `https://sites.google.com/site/sectestunitn/`, 2015. [Online; last read 06-May-2015].

[6] JWebUnit Team. Jwebunit project page. `http://jwebunit.sourceforge.net`, 2015. [Online; last read 06-May-2015].

Appendix A: analysis on the results of Pixy

ID	VarName	EntryPoint	Contained also in:	True?	Why	PoC
xss_index.php_11_min.dot_1 xss_index.php_13_min.dot_1 xss_index.php_13_min.dot_1 xss_index.php_16_min.dot_1 xss_index.php_18_min.dot_1 xss_index.php_19_min.dot_1 xss_index.php_37_min.dot_1 xss_index.php_41_min.dot_1 xss_index.php_44_min.dot_1 xss_index.php_63_min.dot_1 xss_index.php_70_min.dot_1 xss_index.php_71_min.dot_1 xss_index.php_76_min.dot_1 xss_index.php_85_min.dot_1 xss_index.php_87_min.dot_2 xss_index.php_88_min.dot_2 xss_index.php_89_min.dot_2 xss_index.php_93_min.dot_1 xss_index.php_92_min.dot_1 xss_index.php_105_min.dot_2 xss_index.php_111_min.dot_1 xss_index.php_115_min.dot_1 xss_index.php_126_min.dot_1 xss_index.php_138_min.dot_1 xss_index.php_141_min.dot_1 xss_index.php_142_min.dot_2 xss_index.php_146_min.dot_1 xss_index.php_147_min.dot_1 xss_index.php_148_min.dot_1 xss_index.php_149_min.dot_1 xss_index.php_161_min.dot_1 xss_index.php_165_min.dot_2 xss_index.php_180_min.dot_2 xss_index.php_181_min.dot_1 xss_index.php_183_min.dot_1 xss_index.php_184_min.dot_1 xss_index.php_186_min.dot_1 xss_index.php_191_min.dot_1 xss_index.php_194_min.dot_3 xss_index.php_200_min.dot_1 xss_index.php_201_min.dot_1 xss_index.php_212_min.dot_1 xss_index.php_230_min.dot_1 xss_index.php_238_min.dot_1 xss_index.php_239_min.dot_1 xss_index.php_241_min.dot_1 xss_index.php_257_min.dot_1 xss_index.php_260_min.dot_1 xss_index.php_268_min.dot_1 xss_index.php_269_min.dot_1 xss_index.php_272_min.dot_1 xss_index.php_273_min.dot_1 xss_index.php_283_min.dot_1 xss_index.php_288_min.dot_1 xss_index.php_293_min.dot_1 xss_index.php_299_min.dot_1 xss_index.php_309_min.dot_1 xss_index.php_316_min.dot_1 xss_index.php_320_min.dot_1	$page	index:36	AddAssignment:3 AddAttendance:3 AddAnnouncement:3 AddUser:3 EditAssignment:10 EditAssignment:27 EditAnnouncement:23 EditTerm:23 EditTerm:10 AddTeacher:26 AddStudent:28 AddSemester:28 EditGrade:50 EditGrade:31 EditSemester:43 EditSemester:35 ViewClassSettings:36 ClassSettings:36 ManageSchoolInfo:124 ManageSchoolInfo:37 AddParent:39 Login:45 EditTeacher:49 EditStudent:51 ViewCourses:58 StudentViewCourses:63 AddClass:63 ParentViewCourses:64 ViewAnnouncements:67 EditUser:68 EditParent:73 StudentMain:75 TeacherMain:83 ViewStudents:83 ViewAssignments:85 AdminMain:87 DeficiencyReport:90 ParentMain:93 ViewGrades:102 PointsReport:121 VisualizeClasses:128 VisualizeRegistration:14 2 EditClass:142 GradeReport:144 ManageAnnouncements :161 ManageTerms:164 ManageSemester:175 AddClass:177 ManageAttendance:181 ManageTeacher:181 ManageUser:188 ManageParents:203 ManageStudents:211 Registration:240 ManageAssignments:25 7 ManageGrades:270 ManageClasses:303	yes	This var is taken from the POST in the index and used to redirect the flow to the next page in the application. Then is printed again in the next page in order to be posted for the next navigation. All this post/print are made without any check or sanitization	Craft the injection to submit a real value to the server side concateneted with a malicious link 1'/>link an then navigate somewhere
xss_index.php_11_min.dot_2 xss_index.php_89_min.dot_1 xss_index.php_37_min.dot_2 xss_index.php_87_min.dot_1 xss_index.php_88_min.dot_1 xss_index.php_165_min.dot_1 xss_index.php_180_min.dot_1 xss_index.php_181_min.dot_3 xss_index.php_183_min.dot_2 xss_index.php_184_min.dot_2 xss_index.php_194_min.dot_2 xss_index.php_200_min.dot_3 xss_index.php_201_min.dot_3 xss_index.php_309_min.dot_2 xss_index.php_316_min.dot_3	$_POST["selectclass"]	AddAssignment:3 ClassSettings:36 EditAssignment:27 EditGrade:50 ViewClassSettings:36 StudentMain:75 TeacherMain:83 ViewStudents:83 ViewAssignment:85 ParentMain:93 ViewGrades:102 ManageAssignments:257 ManageGrades:270	EditAssignment:27 EditGrade:10	Yes	This var depends on the class chosen in the previous form by the teacher. It's printed as value of an hidden form and submitted to the server in "POST" without check or sanitization.	Craft the injection to submit a real value to the server side concateneted with a malicious link 1'/>link
xss_index.php_2_min.dot xss_index.php_3_min.dot xss_index.php_4_min.dot xss_index.php_6_min.dot xss_index.php_10_min.dot xss_index.php_53_min.dot xss_index.php_92_min.dot_7	$schoolname	header:6	maketop:2 header:15 ManageSchoolInfo:37	No	This var is tainted if it's tainted its value in the database but when the values is stored into the database (header.php:11) the value is sanitized with the funtion "htmlspecialchars"	
xss_index.php_13_min.dot_3 xss_index.php_194_min.dot_1	$_POST["student"]	AddAttendance:3 ParentViewCourses:64 ParentMain:93	ParentViewCourses:3	Yes	This value is printed in in an hidden field without sanitization. This variable navigate through "POST" to the server side and then is printed without checking to the next page	Craft the injection to submit a real value to the server side concateneted with a malicious link 1'/>link
xss_index.php_13_min.dot_4	$_POST["semester"]	AddAttendance:3		Yes	This value is printed in in an hidden field without sanitization. This variable navigate through "POST" to the server side and then is printed without checking to the next page	Craft the injection to submit a real value to the server side concateneted with a malicious link 1'/>link

xss_index.php_11_min.dot_3 xss_index.php_13_min.dot_2 xss_index.php_13_min.dot_2 xss_index.php_16_min.dot_2 xss_index.php_18_min.dot_2 xss_index.php_19_min.dot_2 xss_index.php_37_min.dot_3 xss_index.php_41_min.dot_2 xss_index.php_44_min.dot_2 xss_index.php_64_min.dot_2 xss_index.php_70_min.dot_2 xss_index.php_71_min.dot_2 xss_index.php_76_min.dot_2 xss_index.php_85_min.dot_2 xss_index.php_87_min.dot_3 xss_index.php_88_min.dot_3 xss_index.php_89_min.dot_3 xss_index.php_92_min.dot_2 xss_index.php_93_min.dot_2 xss_index.php_111_min.dot_2 xss_index.php_115_min.dot_2 xss_index.php_126_min.dot_2 xss_index.php_138_min.dot_2 xss_index.php_141_min.dot_2 xss_index.php_142_min.dot_3 xss_index.php_146_min.dot_3 xss_index.php_147_min.dot_3 xss_index.php_148_min.dot_3 xss_index.php_149_min.dot_2 xss_index.php_161_min.dot_2 xss_index.php_165_min.dot_3 xss_index.php_180_min.dot_3 xss_index.php_181_min.dot_2 xss_index.php_183_min.dot_3 xss_index.php_184_min.dot_4 xss_index.php_186_min.dot_2 xss_index.php_191_min.dot_2 xss_index.php_194_min.dot_4 xss_index.php_200_min.dot_2 xss_index.php_201_min.dot_2 xss_index.php_212_min.dot_2 xss_index.php_230_min.dot_2 xss_index.php_238_min.dot_2 xss_index.php_239_min.dot_2 xss_index.php_241_min.dot_2 xss_index.php_257_min.dot_3 xss_index.php_260_min.dot_3 xss_index.php_268_min.dot_3 xss_index.php_269_min.dot_2 xss_index.php_272_min.dot_2 xss_index.php_281_min.dot_3 xss_index.php_283_min.dot_2 xss_index.php_288_min.dot_3 xss_index.php_293_min.dot_3 xss_index.php_299_min.dot_2 xss_index.php_309_min.dot_4 xss_index.php_316_min.dot_2 xss_index.php_320_min.dot_3	$page2	TeacherMain:8 AdminMain:7 ParentMain:8 StudentMain:8	AddAssignment:3 AddAttendance:3 AddAnnouncement:3 AddUser:3 AddTerm:3 EditAssignment:10 EditAssignment:27 EditAnnouncement:23 EditTerm:23 EditTerm:10 AddTeacher:26 AddStudent:28 AddSemester:28 EditGrade:50 EditGrade:31 EditSemester:43 EditSemester:35 ViewClassSettings:36 ClassSettings:36 ManageSchoolInfo:124 ManageSchoolInfo:37 AddParent:39 EditTeacher:49 EditStudent:51 ViewCourses:58 StudentViewCourses:63 AddClass:63 ParentViewCourses:64 ViewAnnouncements:67 EditUser:68 EditParent:73 StudentMain:73 TeacherMain:83 ViewStudents:83 ViewAssignementst:85 AdminMain:87 DeficiencyReport:90 ParentMain:93 ViewGrades:102 PointsReport:121 VisualizeClasses:128 VisualizeRegistration:14 2 EditClass:142 GradeReport:144 ManageAnnouncements :161 ManageTerms:164 ManageSemesters:175 AddClass:177 ManageAttendance:181 ManageTeacher:181 ManageUser:188 ManageParents:203 ManageStudents:211 Registration:240 ManageAssignments:25 7 ManageGrades:270 ManageClasses:303	yes	This value is stored in some form in an hidden field. This value is used to navigate in the functions of the application. The var $page2 is used to navigate through the functionality that are provided by the application based on the user role. This variable navigate through "POST" to the server side and then is printed without checking to the next page	Craft the injection to submit a real value to the server side concateneted with a malicious link 1'/>link
xss_index.php_30_min.dot xss_index.php_31_min.dot xss_index.php_207_min.dot_1	$coursename	ViewAssignment:7 ManageAssignment:7	ViewAssignment:9 ManageAssignment:15	Yes	This var comes from the database. If we go to the "AddClass" function we are able to inject some html/script code that is displayed in these vulnerabilities without sanitization. However the field "coursename" in the table "class" is only 20 chars long and so I can't inject an alert script (too long) but I can inject "Test" Moreover if I came with a link in the sidebar I can inject \'/>ciao and the mysql error will show them whitout sanitization	Inject in the database some malicious code but be sure that is not longer then 20 chars
xss_index.php_37_min.dot_4 xss_index.php_41_min.dot_3 xss_index.php_44_min.dot_3 xss_index.php_85_min.dot_3 xss_index.php_142_min.dot_1 xss_index.php_149_min.dot_3 xss_index.php_161_min.dot_3 xss_index.php_239_min.dot_3	$id	EditAssignment:2 EditAnnouncement:2 EditTerm:2 EditGrade:2 EditTeacher:2 EditStudent:2 EditUser:2 EditParent:2 EditClass:2	EditAssignment:10 EditAssignment:27 EditAnnouncement:23 EditTerm:23 EditTerm:10 EditGrade:50 EditGrade:10 EditSemester:43 EditSemester:35 EditTeacher:49 EditStudent:51 EditUser:68 EditParent:73 EditParent:142	Yes	Variable printed and posted without sanitization.	Inject a valid id + malicious link to show the link tinto the page. Inject only the link without a valid id to print the link in the 'page ot the mysql error
xss_index.php_54_min.dot	$text	Login:13	Login:15	Yes	This var is tainted if it's tainted its value in the database but when the values is stored into the database (header.php:11) the value is not sanitized	We can insert this line in the database through the ManageSchoolInfo form and this will work. Double quotes to escape sql.<script>alert("ciao");</script>
xss_index.php_76_min.dot_2	$_POST["assignment"]	EditGrade:50	EditGrade:10	Yes	Variable printed and posted without sanitization.	Craft the injection to submit a real value to the server side concateneted with a malicious link 1'/>link
xss_index.php_92_min.dot_3	$numperiods	ManageSchoolInfo:22	ManageSchoolInfo:37	No	This value is displayed in an input field that is editable but come directly from the database. When the information in database are update there is not any kind of sanitization but the dabase field is an integer and so it's impossible create a stored xss attack.	
xss_index.php_92_min.dot_4	$numsemesters	ManageSchoolInfo:17	ManageSchoolInfo:37	No	This value is displayed in an input field that is editable but come directly from the database. When the information in database are update there is not any kind of sanitization but the dabase field is an integer and so it's impossible create a stored xss attack.	
xss_index.php_92_min.dot_5	$phone	ManageSchoolInfo:12	ManageSchoolInfo:37	Yes	This value is displayed in an input field that is editable but come directly from the database. When the information in database are update there is not any kind of sanitization but the dabase field is only 14 chars long	Inject in the database some malicious code that close the input tag and print a link but take care of the length (max 15 chars)
xss_index.php_92_min.dot_6	$address	ManageSchoolInfo:37		Yes	This value is displayed in an input field that is editable but come directly from the database. When the information in database are update there is not any kind of sanitization	We can insert this line in the database through the ManageSchoolInfo form and this will work. Double quotes to escape sql.<script>alert("ciao");</script>

xss_index.php_105_min.dot_1	$message	Login:10	Login:45	Yes	This var is tainted if it's tainted its value in the database but when the values is stored into the database (header.php:11) the value is not sanitized	We can insert this line in the database through the ManageSchoolInfo form and this will work. Double quotes to escape sql.<script>alert("ciao");</script>
xss_index.php_146_min.dot_2 xss_index.php_147_min.dot_2 xss_index.php_148_min.dot_2 xss_index.php_183_min.dot_3 xss_index.php_184_min.dot_3 xss_index.php_257_min.dot_2 xss_index.php_260_min.dot_2 xss_index.php_268_min.dot_2 xss_index.php_273_min.dot_2 xss_index.php_283_min.dot_3 xss_index.php_288_min.dot_2 xss_index.php_293_min.dot_2 xss_index.php_309_min.dot_3 xss_index.php_320_min.dot_2	$_POST["onpage"]	ViewAnnouncements:67 ViewAssignment:85 ManageAnnouncements:161 ManageTerms:164 ManageSemester:175 ManageTeachers:181 ManageUsers:188 ManageParents:203 ManageStudents:211 ManageAssignments:257 ManageClasses:303		yes	Value posted and printed without any knid of sanitization.	Craft the injection to submit a real value to the server side concateneted with a malicious link 1'/>link
xss_index.php_234_min.dot_1	$term	ManageSemestesr:133	ManageSemestesr:139	yes	This value comes from a constant query that provides an id to another constant query. But if this term name is tampered in the database so it's showes without checks	Create in the db a term with name nome this will be shown as a link. Rememeber to craft a short injection max 15 chars in db
xss_index.php_269_min.dot_3	$_POST["fullyear"]	AddClass:177		no	When I load this page at line 2 we have a check on this value, if it's different then 1 it prints a hidden input without values otherwise it prints the value directly from the post but it's 1 for sure	

Appendix B: vulnerability not found by Pixy that I've exploited

TaintedVarName	EntryPoint	True tainted	Why
$sitemessage	ManageSchoolInfo	yes	Stored XSS printed without sanitization
$sitetext	ManageSchoolInfo	yes	Stored XSS printed without sanitization
$class	StudentViewCourses ParentViewCourses EditClass ManageClasses	yes	Field of only 20 chars so I have injected a link without href
$studentid	EditParent ViewGrades	yes	Posted and printed without sanitization
$selectclass	ViewAnnouncements	yes	Posted and printed without sanitization
$semester_name	VisualizeClasses VisualizeRegistration GradeReport EditSemester ManageSemesters	yes	Stored XSS printed without sanitization. We have only 15 chars to store an attack
$page2	AddTerm	yes	Posted and printed without sanitization. This var is in almost every page I think I have not read into pixy repost and I have found by myself
$title (announcement)	EditAnnouncement ManageAnnouncements ViewAnnouncemetns	yes	Stored XSS printed without sanitization. We have only 15 chars to store an attack
$message (announcement)	EditAnnouncement ManageAnnouncements ViewAnnouncemetns	yes	Stored XSS printed without sanitization
$assignment (title and task)	ManageAssignment ViewAssignments	yes	Stored XSS printed without sanitization. We have only 15 chars to store an attack on the title but the content is much bigger
$title	EditAssignment	yes	Stored XSS printed without sanitization. We have only 15 chars to store an attack. There should be also a XSS attack on the assignment content but I think that this page is wrong and doesn't print the right values into the right place
$parent	EditParent ManageParents	yes	Stored XSS printed without sanitization. We have only 15 chars to store an attack. But the variables actually are two. Firstname and secondname
$student	EditStudent ManageStudents	yes	Stored XSS printed without sanitization. We have only 15 chars to store an attack. But the variables actually are two. Firstname and secondname
$teacher	EditTeacher ManageTeachers	yes	Stored XSS printed without sanitization. We have only 15 chars to store an attack. But the variables actually are two. Firstname and secondname
$user	EditUser ManageUsers	yes	Stored XSS printed without sanitization. We have only 15 chars to store an attack into the username field