Security Testing: Assignment #7

Tool Vulnerability Report

Fabrizio Zeni
Student Id: 153465

Abstract

This report analises the xss vulnerabilities detected by *pixy*[1] on the *Schoolmate*[2] web application. This is done by looking at the source code and, sometimes, at the database schema.

Contents

[1] http://pixybox.seclab.tuwien.ac.at/pixy/
[2] http://sourceforge.net/projects/schoolmate/

Vulnerabilities 2,3,4,6,10,53

Brief Analysis

Files: maketop.php,header.php

VARIABLE	RESULT
schoolname	false positive

Explanation

Explanation

```
$query = mysql_query("select schoolname from schoolinfo")
       or die("Unable to retrieve school name: " . mysql_error());

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

Listing 1: header.php load of *schoolname*

As we can see from the query, the field that can be the source of the vulnerability is *schoolname*, so we have to check if and where a injection can be made over that field.

```
$query = mysql_query("UPDATE schoolinfo SET schoolname = \"".htmlspecialchars($_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 2: header.php store of *schoolname*

Inside the application we have only one *UPDATE* statement, which is contained in header.php. However we can notice that the input for schoolname is sanitizied through the **htmlspecialchars()** function call. So no injection is possible and then the vulnerability can be classified as a false positive.

Vulnerabilities(*)

Brief Analysis

VARIABLE	AFFECTED PAGES(*)	RESULT
page	all	positive
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. So an XSS attack can be done by injecting a string inside that POST parameter. In the following subsections I will show the portion of code which is affected by the attack.

page

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

Listing 3: AddAssignment.php load of *page*

page2

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

Listing 4: AddAssignment.php load of *page2*

selectclass

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

Listing 5: AddAssignment.php load of *selectclass*

student

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

Listing 6: AddAttendance.php load of *student*

semester

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

Listing 7: AddAttendance.php load of *student*

delete

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

Listing 8: EditAssignment.php load of *delete*

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

Listing 9: EditAssignment.php read of *delete*

assignment

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

Listing 10: EditGrade.php load of *assignment*

onpage

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

Listing 11: ViewAnnouncements.php load of *onpage*

Vulnerabilities 30,31,207

Brief Analysis

Files: ViewAssignments.php,ManageAssignments.php

VARIABLE	RESULT
coursename	true
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 12: 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 13: 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 14: 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 15: 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 16: ViewAssignment.php load of *coursename*

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

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

Listing 17: 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 18: AddClass.php form field

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

Listing 19: EditClass.php form field

Anyway we know that such restriction can be by-passed by intercepting the requests and modify them on-the-fly. We can even have a closer look to the database structure:

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

Listing 20: Sql structure of the field

In fact, the field *coursename* is restricted to 20 chars even in the database structure. Anyway we can inject a string of the form *<a href>a*, which is **12** chars long, and so satisfies the length constraints.

assignment[5]

This vulnerability was not reported bt pixy, but I want to show that it could be a potential problem. 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 21: 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 22: 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 23: ViewAssignments.php variable read of *assignment[5]*

Vulnerability 53

Brief Analysis

File: header.php

VARIABLE	RESULT
schoolname	false positive

Explanation

The schoolname is taken directly from the database without any validation:

```
$query = mysql_query("select schoolname from schoolinfo")
       or die("Unable to retrieve school name: " . mysql_error());

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

Listing 24: Load of *schoolname*

Anyway, there is just one way to update the schoolname and is through this query:

```
 $query = mysql_query("UPDATE schoolinfo SET schoolname = \"".htmlspecialchars($_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 ")
     ;
 $schoolname = htmlspecialchars($_POST["schoolname"]);
```

Listing 25: update of *schoolname*

In the query the schoolname is sanitized through htmlspecialchars, so no injection is possible.

Vulnerability 54

Brief Analysis

File: Login.php

VARIABLE	RESULT
text	positive

Explanation

The sitetext is taken directly from the database without any validation:

```
$query = mysql_query("select sitetext from schoolinfo");
$text  = mysql_result($query,0);
```

Listing 26: Login.php load of *text*

The problem is represented by the fact that even the update of such entry is not validated:

```
$query = mysql_query("UPDATE schoolinfo SET schoolname = \"".htmlspecialchars($_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 27: header.php store of *text*

Vulnerability 92

Brief Analysis

File: ManageSchoolInfo.php

VARIABLE	RESULT
page	true
page2	true
numperiods	false positive
numsemesters	false positive
phone	false positive
address	true
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 28: 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 29: header.php store of *numperiods-numsemesters-phone-address*

Vulnerability 92 continued on next page...

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 30: 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 31: ManageSchoolInfo.php load of *phone*

The load works as for the two field above, and the result can be addressed as a *false positive* thanks to the database schema (*Listing 30*). In this case the column has type *varchar(14)*, which is more sensitive than int, but the print of the tainted variable is only performed in the ManageSchoolInfo page, inside the inside the textarea. So to get out of the text area and print a malicious link we should inject a string like this *'><a href>a* which has a length of 15 chars and so it cannot be injected in the database.

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 32: 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 105

Brief Analysis

File: Login.php

VARIABLE	RESULT
message	true
page	true

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 33: 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 34: header.php store of *sitemessage*

Vulnerability 234

Brief Analysis

File: ManageSemesters.php

VARIABLE	RESULT
term	true

Explanation

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

Listing 35: ManageSemesters.php load of *term*

In the portion of code above, we can see the core of the vulnerability, infact *title* is taken from the database and granted to be safe.

```
$query = mysql_query("UPDATE `terms` SET `title`='$_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());
```

Listing 36: ManageTerms.php store of *term*

Unfortunately, we have this update operation which can modify *title* and is not validated.

Vulnerability 269

Brief Analysis

File: AddClass.php

VARIABLE	RESULT
page	true
page2	true
fullyear	true

Explanation

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

fullyear

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

Listing 37: AddClass.php load of *fullyear*

This attack is almost the same of the *page* one.

Vulnerability 321

Brief Analysis

File: ReportCards.php

VARIABLE	RESULT
data	false 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))
```

Listing 38: ReportCards.php fetch of data from the database

Here the data from the database are fetched and put into a variable (*$class*), which later is used to print into the pdf document.

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

Listing 39: ReportCards.php load of *data*

```
$data = pdf_get_buffer($pdf);

// Close up the db connection //
mysql_close($dbcnx);

header('Content-type: application/pdf');
header('Content-disposition: attachment; filename=ReportCard.pdf');
header('Content-length: ' . strlen($data));
echo $data;
```

Listing 40: ReportCards.php read of *data*

As long as seen at *Vulnerabilities 30,31,207*, *coursename* can be injected with malicious strings which can lead to an xss vulnerability. In this case the pdf generated can contain such malicious string. Anyway the page is not working properly and the pdf is not generated, so this vulnerability is marked as *false positive*. However, if we want to sanitize this vulnerability, the loads from the database should be checked and cleansed by any potential threat.