Security Testing: Assignments #7-8

Due on Friday, April 19, 2013

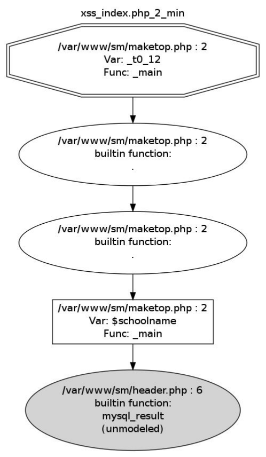
Jones 13:30am

Fabrizio Zeni

Contents

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

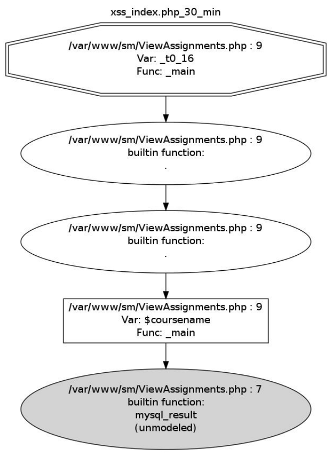
(*) 11: AddAssignment.php — 13: AddAttendance.php —16: AddAnnouncements.php — 18: AddUser.php — 19: AddTerm.php — 37: EditAssignment.php — 41: EditAnnouncements.php — 44: EditTerms.php — 63: AddTeacher.php — 70: AddStudent.php — 71: AddSemester.php — 76: EditGrade.php — 85: EditSemester.php — 87/88: 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. So far, it seems not possible to inject some xss through them.

Vulnerabilities 30,31,207

Brief Analysis


xss_index.php_30_min
/var/www/sm/ViewAssignments.php : 9
Var: _t0_16
Func: _main
/var/www/sm/ViewAssignments.php : 9
builtin function:
.
/var/www/sm/ViewAssignments.php : 9
builtin function:
.
/var/www/sm/ViewAssignments.php : 9
Var: $coursename
Func: _main
/var/www/sm/ViewAssignments.php : 7
builtin function:
mysql_result
(unmodeled)

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 3: 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 4: 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 5: 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 6: 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 7: 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 8: 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 9: AddClass.php form field

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

Listing 10: 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 11: 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 12: 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 13: 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 14: ViewAssignments.php variable read of *assignment[5]*

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 15: 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 16: header.php store of *text*

Testing Code

```
                tester.setTextField("username", "dduck");
                tester.setTextField("password", "duck");
                tester.submit();
                break;
            default:
                break;
        }
    }

    private void click(String linkname,Integer mode){
```

Listing 17: jwebunit test code for *text*

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 18: 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 19: header.php store of *numperiods-numsemesters-phone-address*

Vulnerability 92 continued on next page...

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 20: 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 21: 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 20*). 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 22: 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.

Testing Code

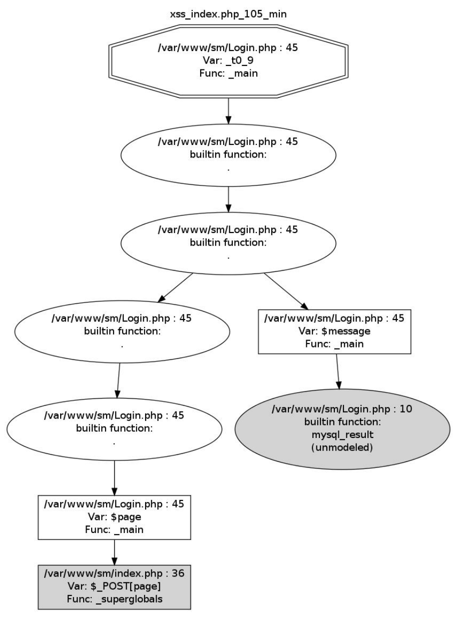
address

```
         * 0: link with text
         * 1: button
         */
        switch(mode){
            case 0:
                tester.clickLinkWithText(linkname);
                break;
            case 1:
                tester.clickButtonWithText(linkname);
```

Listing 23: jwebunit test code for *address*

Vulnerability 105

Brief Analysis


xss_index.php_105_min
/var/www/sm/Login.php : 45 Var: _t0_9 Func: _main
/var/www/sm/Login.php : 45 builtin function: .
/var/www/sm/Login.php : 45 builtin function: .
/var/www/sm/Login.php : 45 builtin function: .
/var/www/sm/Login.php : 45 Var: $message Func: _main
/var/www/sm/Login.php : 45 builtin function: .
/var/www/sm/Login.php : 10 builtin function: mysql_result (unmodeled)
/var/www/sm/Login.php : 45 Var: $page Func: _main
/var/www/sm/index.php : 36 Var: $_POST[page] Func: _superglobals

File: Login.php

VARIABLE	RESULT
message	positive
page	false 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);
```

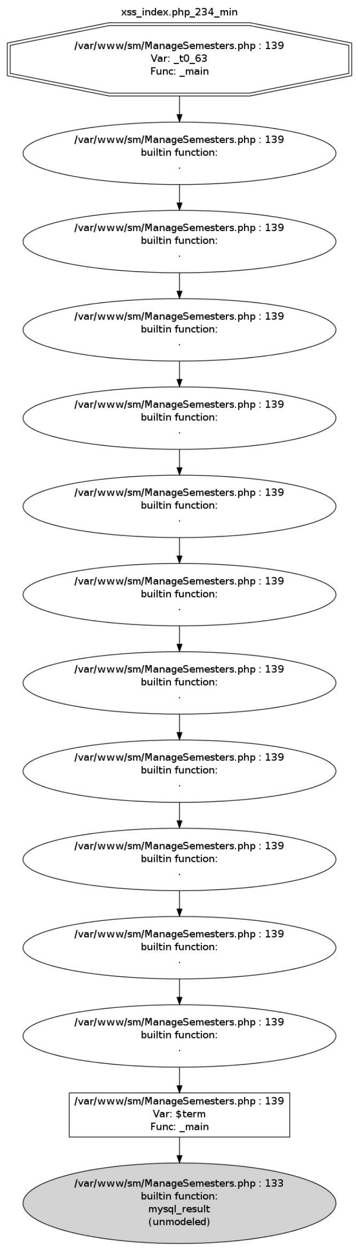
Listing 24: Login.php load of *sitemessage*

```
$query = mysql_query("UPDATE schoolinfo SET schoolname = \"".htmlspecialchars($_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 25: header.php store of *sitemessage*

Vulnerability 234

Brief Analysis


xss_index.php_234_min
/var/www/sm/ManageSemesters.php : 139
Var: _t0_63
Func: _main
/var/www/sm/ManageSemesters.php : 139
builtin function:
.
/var/www/sm/ManageSemesters.php : 139
Var: $term
Func: _main
/var/www/sm/ManageSemesters.php : 133
builtin function:
mysql_result
(unmodeled)

File: ManageSemesters.php

VARIABLE	RESULT
term	positive

Vulnerability 234 continued on next page. . .

Explanation

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

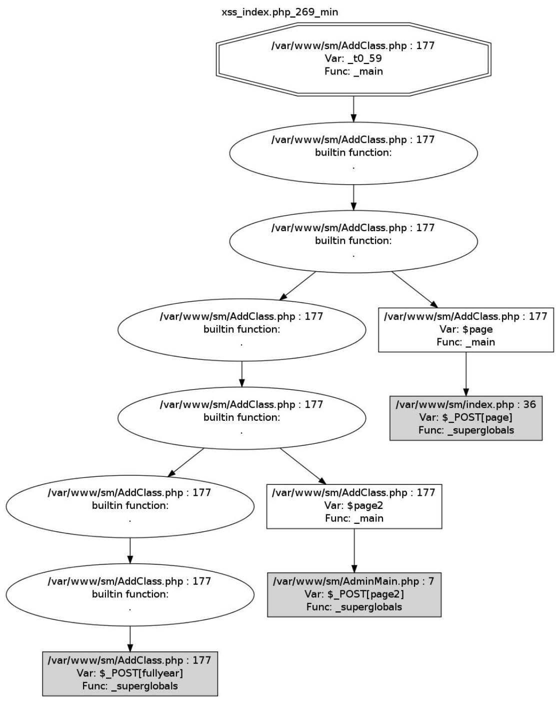
/var/www/sm/ManageSemesters.php

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

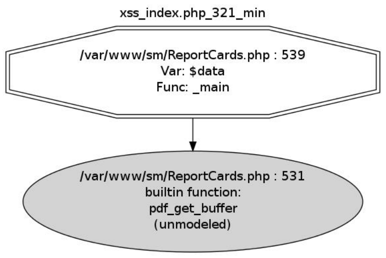
The analysis of the section *Vulnerabilities*$^{(*)}$ can also fit for *page* and *page2*.

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.