# Security testing project report

Gianluca Bortoli
DISI - University of Trento
Student id: 179816
gianluca.bortoli@studenti.unitn.it

## 1. INTRODUCTION

This work aims at performing a security analysis study on *Schoolmate*. This web application is PHP/MySQL solution for elementary, middle and high schools where different type of users can manage all the needed information to fulfill their everyday tasks.

This report is structured as follows. Section 2 describes the naming convention used for the test cases, how they are structured into packages and shows the vulnerabilities that have been found on the subject application. Section 3 depicts how to fix these security braches. Section 4 reports the outcomes of the Proof-of-Concept (PoC) attacks described in Section 2. Finally, Section 5 depicts the steps that have to be taken in order to be able to run all the tests from scratch.

## 2. SECURITY TEST CASES

This security review uses a static analysis tool to scan the application's source code in order to spot security vulnerabilities. *Pixy* is one of them and it scans PHP applications for security vulnerabilities. This software is used in order to spot all the possible vulnerabilities in *Schoolmate*, the subject web application.

The workflow followed to develop this work is the following:

1. Pixy is run on the application source code to produce vulnerability reports regarding Cross Site Scripting attacks (XSS).
2. all the reports are manually analyzed to perform a first classification of the resulting vulnerabilities. Thus, each of them has to be categorized either as **false positive**, **reflected XSS** or **stored XSS** (please refer to the *xss_classification.pdf* attachment for a full description of all the vulnerabilities found in step 1).
3. a test cases for all the true positives (TP)[1] is written in the form of a PoC attack against the application under test.
4. every test case is checked to be failing on the bugged application to demonstrate *Schoolmate* has such security vulnerabilities and that they can be exploited.
5. the *Schoolmate*'s PHP code is modified to fix the vulnerabilities found in step 2.
6. the whole test suite is run again to check all the tests pass. This means that the PoC attack implemented in each test case is no longer possible after the application is fixed.
7. Pixy is run again on the new application source code to check the vulnerabilities are really fixed and to make sure step 5 did not introduce any additional security breach.

The most effective way to implement a security test case for a XSS vulnerability is to be able to inject some malicious code inside the HTML of the page that is shown to the user. All the test implemented in this work inject a *malicious link* and then checks whether it is found on the target web page.

### 2.1 Test case code structure

The test suite developed to demonstrate that the application under revision has security concerns makes use of JWebUnit, a Java-based testing framework for web applications.

The test suite's source code is structured as follows:

- the **src/main/java** folder contains a Java file for each Pixy report which has at least one TP. The test-case file naming follows the convention *Test<pixy_report_number>.java* so that it is possible to easily match the test case with its Pixy report counterpart[2].
- the **src/main/java/common** folder contains three packages that implement utility functions that are available to all the test cases. These functions allows not to soil the actual test case implementation, separating all the sequences of operations that has to be done in many tests in order to navigate the web page depending on the situation. In this way, it is possible to maximize the code reuse across multiple test cases and to better modularize the project.

A noteworthy aspect is how tests' assertions are written. When dealing with security PoC test cases, the security expert writes a test that is thought to fail if the vulnerability can be exploited and it should pass otherwise. Hence, in this scenario a failing test (and its assertion) means that the vulnerability is still present, while a passing one means that the vulnerability is not present any more. For the seek of generality all the test cases can run in any order. To achieve this, every test cases involving a stored XSS vulnerability

[1] A single test can cover either a reflected or a stored XSS vulnerability. Furthermore, each Pixy report may include more than one vulnerability for each report.

[2] This is done to simplify the revision work, since Pixy identifies every report with a number in its filename.

has a *cleanup* function called after the test case execution which removes potentially dirty data from the database.

Furthermore, most of the tests perform three types of action on the web page. They can *navigate* through the pages of the web application clicking buttons and/or links, they can *input* some values in the available forms and text fields and finally they *make assertions* about the content of the page (eg. that the malicious link is not present in the page).

Last but not least, some of the test cases (ie. the ones involving the variable *page*2) required to modify the structure of the Document Object Model (DOM) of the webpage, since the submit buttons of the forms automatically set values for some of the form's hidden fields using some javascript code connected to the *onClick* action. To overcome this issue, a new ad-hoc submit button is created inside the form and this one is used to perform the action instead of the original one. This functionality is implemented inside the *src/common/utils.java* file with the *addSubmitButton* function.

## 3. SOURCE CODE FIXES

All the vulnerabilities explained in Section 2 are exploited in the test cases inserting a malicious link that appears on the page. This succeeds because the application does not perform any kind of validation and/or sanitization on the variables that are shown on the page. Unfortunately, many of these variables are (or contain) input under the control of the user. Thus, performing sanitization steps is not an option in this case.

The sanitization phase can be performed either before the user input is stored in the MySQL database or when the value is retrieved from the storage to be displayed on the web page. This is up to the developer and highly depends on the application structure. To fix the PHP source code of *Schoolmate*, this work makes use of the first approach.

The *index.php* file is the one responsible for handling every request coming from every other page in the application and it is the one including all the other ones depending on the values of *page* and *page*2. This architectural decision can be exploited in order to fix all the vulnerabilities affecting the entire application in a single point. The portion code shown below performs the sanitization step of all the POST parameters reaching the application.

**Listing 1: Fixing all the vulnerabilities in index.php**

```
<?php
foreach ($_POST as $k => $val) {
 $_POST[$k] =
    htmlentities($val,ENT_QUOTES,"UTF-8");
}
...
?>
```

The decision to apply the **htmlentities** function to all the items in the POST array allows to eliminate the possibility to modify the page's content by means of malicious tags in the input text fields that are later printed on the page, since the DOM is modified my the attacker. This function converts all the characters that has a special meaning in HTML to its "safe" encoded version. The optional parameter *ENT_QUOTES* is used to convert both double and single quotes, while the *UTF-8* specities the encoding to be used when converting the characters.

Finally, the addition of this code snippet at the very beginning of the *index.php* file allows to fix both all the stored and reflected XSS vulnerabilities in *Schoolmate*, since these were the only vulnerabilities of interest in this security revision.

## 4. CONCLUSIONS

The test cases developed for this work and explained in Section 2 covers all the vulnerabilities found by Pixy which are classified as true positives. All the tests failed when the original code of *Schoolmate* is deployed on the webserver, while they all successfully pass once the fixes explained in Section 3 are applied.

In order to check that the patch does not introduce any new vulnerability and fixes the old ones, the static analysis tool is run again on the fixed version of *Schoolmate*.
After this final step, Pixy revealed the same vulnerabilities discussed in Section 2. Despite the outcome of the tool, all the PoC tests successfully pass after patching the application. Hence, it is possible to claim that all the reports created by Pixy are false positives. This is due to the fact that static anaysis tools perform an overestimation of the paths that can be followed during an actual execution of the application[3]. Thus, based on the outcome of the test suite, it is possible to state that the vulnerabilities are fixed.

## 5. PRELIMINARY STEPS

In order to run all the test cases, the following preliminary steps has to be followed to fullfil the application's requirements:

1. install and run a webserver (eg. Apache), PHP and MySQL.
2. edit the $*dbuser* and $*dbpass* values in the *index.php* file in order to match the ones of the MySQL instance to be used.
3. copy the *Schoolmate* source files inside the webserver directory.
4. import the *dbDump.sql* script in the MySQL instance. It takes care of creating the a database with the right name and populating it with some initial values. All the queries on the database should succeed.
5. import the tests' source code as a Maven project in your IDE so that all the needed dependencies can be automatically downloaded from the remote repository and included in the project.

After all the abovementioned steps are completed successfully, Schoolmate is up and running and the test suite can be run.

[3]As explained in Section 2, this is the reason why an initial dinstinction between TP and FP has to be done in advance before implementing the PoC attacks.