Filesync and Era Literaria: Realistic Open Source Webs To Develop Web Security Skills

Jose Manuel Redondo López[1,*] and Leticia Del Valle Varela[2]

[1]*Department of Computer Science, University of Oviedo, C/Calvo Sotelo S/N Oviedo (Asturias), 33007, Spain*
[2]*GADD Grupo Meana S. A., Palacio de Lieres s/n Lieres (Asturias), 33580, Spain*
E-mail: redondojose@uniovi.es; leticiavallevarela@gmail.com
[*]*Corresponding Author*

Received 15 March 2018; Accepted 20 August 2018;
Publication 10 September 2018

Abstract

A great variety of services and applications are currently offered using web sites. Unfortunately, this also caused the proliferation of attacks targeting their potential vulnerabilities. Therefore, the demand for security-trained professionals that identify, prevent and find solutions to security vulnerabilities is greatly increasing. This also increased the need for adequate training tools that show how real attacks are performed and prevented. In this paper we describe the design, implementation and usage examples of two websites designed to facilitate web security training. These websites have a realistic set of features and have been developed using different popular technologies. They deliberately incorporate examples of a large subset of common security vulnerabilities, complemented with learning and training materials. They are also open source to allow the development of customizations and adaptations to different scenarios and facilitate learning secure code development techniques.

Keywords: Web security, pentesting, OWASP, vulnerability, training.

Journal of Web Engineering, Vol. 17_5, 1–22.
doi: 10.13052/jwe1540-9589.1751
© 2018 River Publishers

1 Introduction

The popularity of web browsers, cloud computing and the globalization of economic activities turned the web as one of the preferred platforms to deliver services or products. Web sites are no longer simple static information-delivery tools, but complex and dynamic applications that can model almost any business need. Some web applications are aimed to the general public, like productivity tools [1, 2], or shops [3, 4]; others are aimed to concrete business areas, fulfilling specialized requirements [5, 6].

As websites became more complex the probability of having security vulnerabilities have also increased. Nowadays, attacks that specifically target web sites are a major problem [7, 8]. There is a great variety of web attacks that exploit different vulnerabilities, evolving and acquiring new attack vectors over time. Projects like OWASP [9] study, classify, and rank them.

In general, a web vulnerability can be originated from three different sources:

1. Defects/omissions/bugs in the development platforms or supporting third-party products [10].
2. Unawareness or incorrect application of security mechanisms within the underlying supporting infrastructure (web servers, OS, libraries, frameworks...).
3. Inability to develop proper secure coding practices [11, 12].

Because of this, companies have increased the demand of web security professionals [13]. Their goal is to assess the development of secure web sites, applying a series of techniques that try to avoid the three mentioned sources of problems, so they cannot be used to unfold attacks. They also may test the security of existing web applications to strengthen their security. Normally, the applied techniques are the same as the ones used by malicious attackers, although in a controlled environment. As a result, they create reports describing the problems found and potential solutions. This is commonly known as *pentesting* (penetration testing) [14].

To train pentesters, academic institutions need properly trained professionals and appropriate tools to help students to reach adequate skill levels. Popular tools for this purpose are deliberately vulnerable web applications. They incorporate different sets of known security flaws, so the students can experiment its discovery, potential consequences (exploiting), and/or develop solutions, depending on the training goal.

There are a great number of products of this type [15, 16]; some of them emphasize the educational aspect without simulating a real website,

being a collection of tutorials of specific vulnerabilities or challenge-oriented games. Opposite, we can find tools that simulate a core functionality (image galleries, shops...), with vulnerabilities placed in different parts, but not designed as a realistic web site. Finally, there are realistic training web sites, but sometimes they give the users little clue about its potential vulnerabilities and architecture, so its discovery depends on student skills only. This can limit its applications in security learning environments.

Tutorial-based, game-focused, and simulated web sites are key to achieve proper knowledge of web security concepts. However, once the student training is advanced enough, practicing in environments that are much closer to the real ones facilitates achieving proper skills. Unfortunately, realistic security-training web sites are scarce, as their development is more complex and time-consuming. The effort required to develop a realistic web site is significantly increased when part of the requirements include integrating key security vulnerabilities without compromising their goals. This requires careful planning and implementation, and also develop its educational aspects without sacrificing its realism. An extra effort to develop adequate documentation about the web functionality and its implemented vulnerabilities is necessary. Also, increasing the available number of these type of tools gives more freedom to design different types of educational activities or courses.

In this paper we present two realistic web security training applications, *FileSync* and *Era Literaria*. They are developed with two popular web development technologies [17, 18] using current design, implementation and UI standards. They contain a substantial number of well-known and carefully chosen security vulnerabilities, deliberately introduced at known points. We also provide separate and detailed documentation about its architecture, implemented security problems (including how to find them), potential solutions, and management instructions (installation, maintenance...). This way, we can maintain their educational value without compromising their realism, and allow its usage in more scenarios. They are also open source, so anyone can develop adaptations or extensions that fulfill concrete needs.

This paper is structured as follows: Section 2 describes the web applications, its design and development principles, while Section 3 details their architecture and the implemented vulnerabilities. The next section offers an overview of the possible applications of these tools in common training scenarios, while Section 5 describes the related work. Finally, Section 6 details the conclusions and future work.

2 Website Design

Our web applications were designed following a series of common rules:

- *Realism*: design, implementation and GUI follow current industry standards, as popular websites with similar features [19]. We also use current practices to increase their realism, such as mail confirmation of newly created accounts, anti-CSRF tokens or simulation of merchant services.
- *Functionality*: All their functionalities are fully implemented.
- *Data volume and type*: Both contain a substantial amount of realistic data.
- *Frameworks*: they use modern, well-known and widely used web development frameworks and supporting products.
- *No shared technologies*: both use different development technologies and frameworks, programming languages, hosting web servers and operating systems.
- *Shared vulnerabilities*: examples of the same vulnerabilities can be found in both websites, to show how the same problems can appear even when using totally different supporting infrastructures and technologies.

FileSync allows registered users to upload, download and share files. Its purpose is to be a local file sharing tool with a restricted user community, emulating the file management and sharing behaviors of products like *ShareFile* [20], but with non-anonymous users. Figure 1 shows the interface of this application.

Era Literaria is an online bookshop that implements the classical online shop concepts. It emulates the UI of a real bookshop [21]. Figure 2 shows the interface of this application.

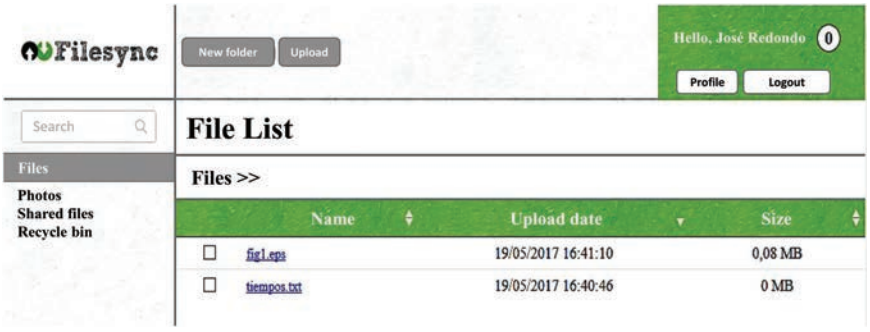

Figure 1 FileSync user interface.

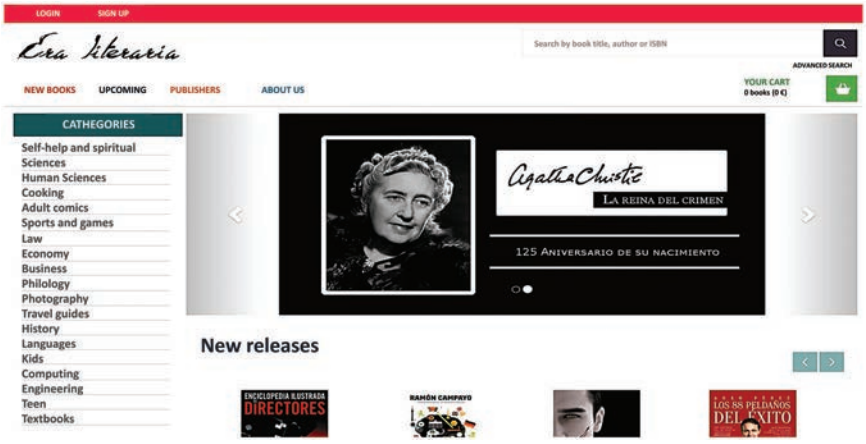

Figure 2 Era Literaria user interface.

2.1 Vulnerability Inclusion Design

Implemented vulnerabilities were chosen and classified following the *OWASP Pentesting guide v4* [22]. Our selection includes several classified as *Top 10* by the OWASP organization [9] (see next subsection). We also provide an educational guide to explain every security vulnerability introduced on each website. For each vulnerability, this guide provides its OWASP name, a description of the problem (referencing official OWASP materials), how we can detect it, in which places of each web is present, and proposed solutions.

Proposed solutions for vulnerabilities focus on correctly applying the built-in security mechanisms of the frameworks or third-party products used to build each website, if available. Therefore, we favor using tried-and-tested security mechanisms over developing custom solutions to try to show the best possible solution to each vulnerability. Applying these mechanisms also requires substantial knowledge of the vulnerability they protect against, so the students know the scope of the problem before learning a proper solution. This also targets the second source of vulnerabilities that was mentioned in the introduction.

3 Website Implementation

Although both websites use different third-party supporting products, they are built using the *Model-View-Controller* (MVC) architectural pattern (see Figure 3).

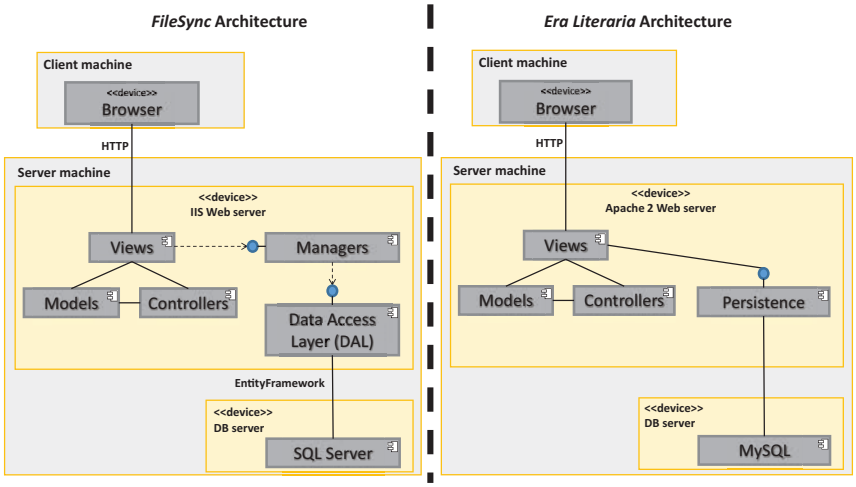

Figure 3 *FileSync* and *Era Literaria* architectures.

- *Era Literaria* can be hosted in any web server able to run *PHP 5* or *PHP 7*. It also uses a *MySQL* database engine, but it can work with other compatible products, such as *MariaDB*. Additionally, it uses *PHPMyAdmin* to manage the database. The web was built using the *Yii* framework [23]. In our tests we used the *Apache 2* web server and *Yii* version 1.1.16.
- *FileSync* is coded in *C#* and targets *Windows* operating systems. It can be hosted in *IIS 8* or higher. It uses a *SQL Server* compatible database engine (2012 version or higher) and the *MVC* framework [24]. For our tests we used the free *SQL Server Express* database engine and the *MVC 5* framework.

3.1 Chosen Vulnerabilities

We chose the vulnerabilities we considered most important to address to attain a high security level on any website from the OWASP classification. There are a total of 21 vulnerabilities implemented on each website. Some of them are caused by faulty administration procedures, others are due programming errors or wrong usage of the underlying frameworks and/or third-party products. This way, the covered vulnerabilities are more varied, so students using *FileSync* and *Era Literaria* for training must develop a wider range of security skills to implement proper solutions. The Table 1 shows a description of each vulnerability, its common OWASP name and classification code [22].

Table 1 Vulnerabilities implemented in *FileSync* and *Era Literaria*

Vulnerability	**OWASP Code**	**Description**
Information Gathering		
Fingerprint Web Server	OTG-INFO-002	Determine the type and version of the installed web server (including other third-party software if possible), to identify its potential vulnerabilities in public data sources [10]
Review Webserver Metafiles for Information Leakage	OTG-INFO-003	Determine if the presence of certain files (`robots.txt`) or meta-information tags in the server gives attackers too much information
Fingerprint Web Application	OTG-INFO-009	Determine known third-party products installed to support web site functionalities (like *WordPress*), with the same final goal as the previous point
Identity Management Testing		
Testing for Account Enumeration and Guessable User Account	OTG-IDENT-004	Guess if it is possible to obtain a valid user list by exploring the login system of a web application. This can be used to unfold other types of attacks, like brute-force password enumeration of *Advanced Persistent Threats* (APTs)
Authentication Testing		
Testing for Credentials Transported over an Encrypted Channel	OTG-AUTHN-001	Verify that the transmission of the authentication data uses a properly encrypted channel [25], thus avoiding data leaks
Testing for Weak lock out mechanism	OTG-AUTHN-003	Identify problems in the automatic account blocking mechanisms implemented to avoid brute-force password guessing attacks, if present
Testing for Browser cache weakness	OTG-AUTHN-006	Determine if the application is showing private or sensible data by incorrect browser cache management
Testing for Weak password policy	OTG-AUTHN-007	Test the web site password policy

(*Continued*)

Table 1 Continued

Vulnerability	OWASP Code	Description
Session Management Testing		
Testing for Bypassing Session Management Schema	OTG-SESS-001	Check the session token creation policy of the application, and if these tokens can be guessed or predicted
Testing for Cookies attributes	OTG-SESS-002	Check the existence and proper usage of certain security-related cookie attributes [26]
Testing for Session Fixation	OTG-SESS-003	Determine if it is possible to assign forged cookies to existing users to steal valid user sessions
Testing for Exposed Session Variables	OTG-SESS-004	Determine if the application properly protects important session variables
Testing for Cross Site Request Forgery (CSRF)	OTG-SESS-005	Determine if the web has CSRF (*Cross-Site Request Forgery*) vulnerabilities
Testing for logout functionality	OTG-SESS-006	Check for common errors in the logout functionality of the application
Test Session Timeout	OTG-SESS-007	Determine if the web site has automatic logout functionality due to inactivity
Input Validation Testing		
Testing for Reflected Cross Site Scripting	OTG-INPVAL-001	Determine if the application is vulnerable to *reflected XSS* attacks
Testing for Stored Cross Site Scripting	OTG-INPVAL-002	Determine if the application is vulnerable to *stored XSS* attacks
Testing for HTTP Verb Tampering	OTG-INPVAL-003	Check the web server response to forged or not typical HTTP verbs present in the request
Testing for SQL Injection	OTG-INPVAL-005	Test if the application is vulnerable to various forms of injection attacks
Testing for ORM Injection	OTG-INPVAL-007	Test if the application is vulnerable to various forms of injection attacks
Testing for Error Handling		
Analysis of Error Codes	OTG-ERR-001	Determine if error messages give too much information to potential attackers. Attackers may use this information to perform the same operations detailed in OTG-INFO-002
Business Logic Testing		
Test Upload of Unexpected File Types	OTG-BUSLOGIC-008	Test if the application properly manages the files that different users may upload

3.2 Implementation of Vulnerabilities

As we said, the educational guide supplied with *FileSync* and *Era Literaria* describes the nature of each implemented vulnerability. Additionally, it details three important aspects of every vulnerability on each application: how they have been implemented, how they can be found, and suggested steps to fix the problems they cause. This was done to facilitate the development of security programming skills. As an example, in this section we will review these three aspects of the two vulnerabilities used in the next section to show usage scenarios.

The first security vulnerability is OTG-INPVAL-002 (*Testing for Stored Cross Site Scripting*, see Table 1) in *Era Literaria*. This vulnerability is caused by an incorrect handling of user supplied input. To avoid this vulnerability, a web application must check every possible user input to see if the data follows the expected format or type. This not only includes URL-supplied parameters in typical `HTTP GET` requests made to the different application sections, such as `GET Eraliteraria/web/book/categories/2/Autoayuda_y_Espiritualidad`, but also `POST` parameters or data read form cookies.

The *Yii* framework used to develop *Era Literaria* has no automatic protection for *XSS* attacks. The user must manually check every possible user input with the appropriate framework features to see if it might be used to perform an attack. In the case of *Era Literaria*, these manual checks have been omitted deliberately from certain application parts. This models a frequent code injection security problem, that happens when programmers forget to properly validate specific input sources of a web page [27], even if other parts of the same page are correctly validated.

One way of finding this vulnerability is through the *modify user profile* functionality. User name changes are not validated, and therefore a session-stealing script can be introduced easily via *XSS* in the user name field. Therefore, every user seeing this malicious user name will be victim of this attack. This is critical if we consider that the `admin` user can view the list of users as part of its private administration interface. As the user name is displayed in this user list, the injected script can be used to steal the administrator session token, and therefore impersonate it. This will enable the malicious user to tamper or destroy information of the application.

This attack vector also allows more quiet attack types. For example, the `admin` can add new books to the store. As this is a restricted feature, no security validation is performed to the data when adding new books. As the malicious user can now impersonate the `admin`, new books with malicious scripts within

their data can now be added. Therefore, a great amount of book related requests can now be used to compromise user data: just viewing a detailed profile of a book (`GET Eraliteraria/web/book/view/9788416029297`) could now be used to perform a wide variety of attacks: session stealing of standard users, clickjacking attacks, giving false information...

The *book review* feature is another potential source of this vulnerability. Book reviews are also not validated, so any user may input scripts within the review text. As reviews are displayed to the users that check the detailed information of a book, any user (even the administrator) may become a victim when viewing a book with a malicious review. This will be shown on the next section.

To avoid this kind of attacks all user input must be sanitized. As we said, we need to do this manually on every input, with the help of the `CHtmlPurifier` library of the *Yii* framework. Every time a user input is shown or captured in a form, parsing the input with the features of this library will ensure no malicious input is displayed to the users. However, we must not forget to call this library on every possible attack vector, so a thorough analysis of every possible application form in the `View` package (see Figure 3) and input parameters must be performed. In *Era Literaria*, affected forms are:

- User registration
- Add a book review
- User profile data modification
- Change the address of a shipment
- Add/Edit a book

Regarding *FileSync*, one of the most important implemented vulnerabilities is OTG-BUSLOGIC-008 (*Test Upload of Unexpected File Types*, see Table 1). This vulnerability can be found by examining its *upload* functionality, that do not restrict or parse uploaded files. This happens because *FileSync* is designed as an all-purpose file sharing application, conceived to allow authorized users to share as many types of contents as possible. As users are identified, programmers trusted that they will not perform malicious operations because they can be tracked, and this way possible shared content types are totally open. However, malicious attacks using a stolen user id are possible, and therefore users cannot be really blamed for uploading malicious files without further investigation. This scenario is a typical case in which data turns into executable code: if a user uploads files with code that can be executed in a web client, the contents of the file will not be displayed or downloaded when accessed but executed as part of the application source code. Because of this, the code in the uploaded files will have the same rights as the code of the application,

so the effects of this type of vulnerability can be critical. Additionally, the vulnerability is not only restricted to code execution, as uploading server key files such as `web.config` may even change the server configuration.

Solving this vulnerability in *FileSync* requires careful planning. Should we sacrifice part of the application functionality by filtering allowed file types? Should we maintain allowed file types unrestricted by parsing potentially dangerous ones? Implementing the latter is more difficult, as a successful parsing method for every type of uploaded file must be implemented without corrupting the file contents [28]. In our educational guide we decided to restrict the file types to be uploaded, which is easier to understand and faster to implement, although it makes *FileSync* less versatile. To restrict uploaded files, OWASP recommends filtering allowed file types using a while-list or a black-list approach [29]. This cannot be done only by consulting the `Content-Type` HTTP Header, as it can be modified by the user using an HTTP proxy [30]. Normally a white-list approach is more advisable, as we cannot ban every possible file type that may turn into executable code, because new file types may be capable of this in the future. Whitelisting files has also its own problems, as file extensions may be altered to circumvent filters. Typical methods to circumvent file filters include adding a dot at the end of the uploaded file (`file.php.`) or using multiple file extensions (`file.php.jpg`). Therefore, fixing this problem in *FileSync* requires implementing multiple measures:

- A white list will be defined in the application, accepting common image types (`.jpg`, `.bmp...`), documents (`.pdf`, `.docx...`), multimedia files (`.mp3`, `.avi...`) and compressed files (`.zip`, `.rar...`), among other file types considered non harmful.
- All uploaded files must carry a valid single extension included in the previous white list.
- The `Upload` method of the `File` controller (see Figure 3) is extended to check the file extension against the white list, to ensure that it belongs to an allowed type.
- *FileSync* allows users to change file names and extensions. This can be used to circumvent the upload filter. Therefore, the `ChangeName` method of the `File` controller (see Figure 3) is also modified to use the white list.
- File names must be composed by just alphanumeric characters and dots. Empty file names or names longer than 255 characters are not allowed. This can be implemented in the `CheckName` method of the `FileManager` class. This class belongs to the `Managers` package, as can be seen in Figure 3.

- Files will be uploaded to a folder whose name is the same as the uploading user name. This folder will not have execution permissions assigned. This also avoids problems if files are uploaded to the web root folder overwriting important application files.
- File size is limited to 20Mb to avoid a potential *Denial-Of-Service* scenario.
- File uploads are only allowed via the HTTP `POST` method.
- Finally, compressed files are not opened or examined, to avoid a possible zip-bomb attack vector [31].

Therefore, these two examples show how both web applications have been implemented and how the vulnerabilities have been included in their code and how they can be found. Thanks to the educational guide and the open source nature of the applications, they can be also used as training tools of secure programming practices. Users can experiment common security vulnerabilities in action, understand, and fix them. This enhances the educational capabilities of the applications, covering a broader variety of educational needs and not only pure pentesting-related activities (see next section).

4 Sample Work Scenarios

This section describes some usage scenarios of our web sites as adaptable educational or training tools.

4.1 Black Box Pentest (Blind Pentest)

We will give each student an *Ubuntu Linux* server with *Era Literaria* installed over the *Apache 2* web server in a non-conventional port. The students must first discover the existence of the web application, explore it, and test any vulnerability detection technique that they have learned. During this vulnerability discovery phase, they will reach the web page that allows to leave comments and opinions about books. As we saw in Section 3.2, this page is vulnerable to *stored XSS* attacks (OTG-INPVAL-002), as the programmers forgot to apply proper filtering to the user-delivered content. Figure 4 shows how this attack can be used to show the session ID of the user that views the malicious comment. One possible exploit of this vulnerability is remote-controlling the user session. For example, by using *BeeF* [32], an attacker can open a remote administration panel to modify the user cookie and see the actions of any user that views a malicious book comment. This attack unfolds by injecting the *BeeF hook.js* script into the comment text box of the web

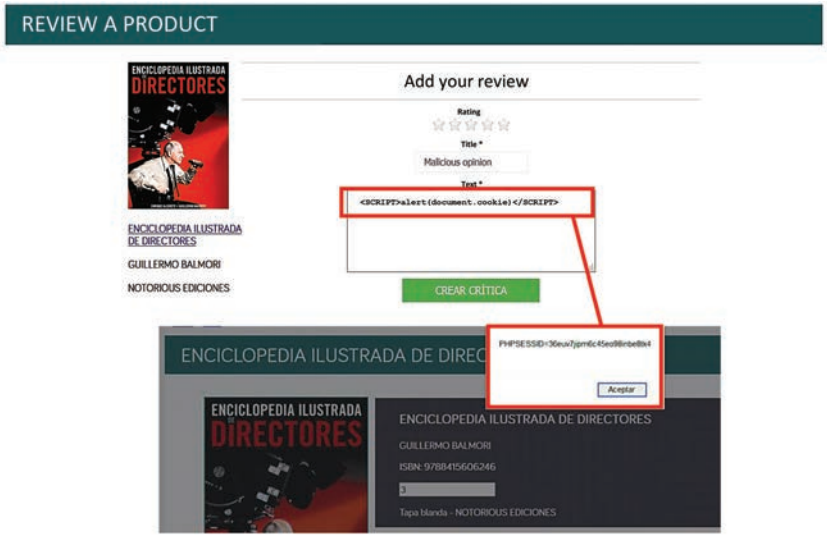

Figure 4 Stored XSS attack in *Era Literaria*.

page, using the same injection method shown in Figure 4. A full description of this attack is detailed in the guide that accompanies *Era Literaria*.

This scenario illustrates the dangers and the sophistication level of an *XSS* attack, showing the importance of avoiding this kind of vulnerabilities. Scenarios like this are proposed to be used as training activities in the *Web Security Systems* course of the Spanish official *Master on Web Engineering* of the *University of Oviedo* [33].

4.2 White Box Pentest

We provide the students a *Windows 2016* server with *FileSync* installed in a known location. Additionally, we also give them the source code of the application and its user manual. With all these materials, the students must perform code reviews and vulnerability discovery procedures, performing all the tests they know. During this process, the students may find that the *file upload* functionality of this web page do not perform proper file type checking, as described in Section 3.2. Therefore, any user can upload an HTML web page with malicious *Javascript* code, as the one shown in Figure 5.

Now, if the attacker shares this malicious file with other registered users, its contents will be executed under their identity. As we said, this can have a great amount of potential consequences (session hijacking, web content modification. . .).

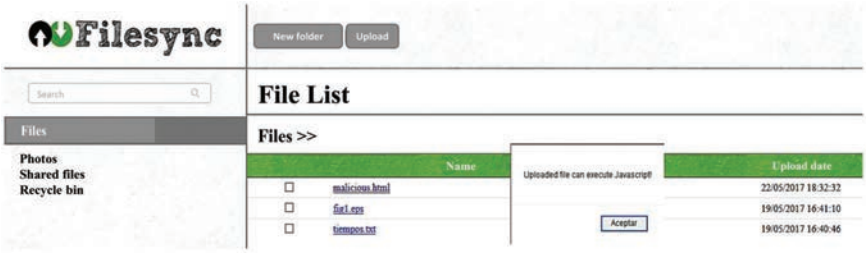

Figure 5 Uploading a malicious web page in *FileSync*.

4.3 Other Usage Scenarios

This section describes other potential usage scenarios. The combination of open source code, detailed documentation, and implementing a wide range of vulnerabilities allows us to adapt our tools to different course types, environments, or student profiles:

- *Find only concrete types of vulnerabilities*, to use them to launch certain predetermined exploits (UI redressing, private information capture, keylogging . . .).
- *Develop countermeasures* to selected vulnerabilities (discovered by the users or just initially explained to the students). As we said, depending on the targeted vulnerabilities this requires learning proper usage of the frameworks, developing custom secure code, or administering the underlying infrastructure (enabling encryption, modifying HTTP responses. . .), giving focus to secure programming procedures.
- *Develop proactive responses* to stop certain types of attack when they are detected. This requires modifying certain parts of the web source code. Countermeasures can range from IP blocking, warning messages, logging . . .
- *Perform web and server vulnerabilities information gathering tasks only*. The goal is to perform a comprehensive report of vulnerabilities and solutions, not launching any kind of exploit.
- *Mixed gray-box pentesting scenarios*. The students have initially a restricted amount of information about the web and its infrastructure, so their first step will be to try to obtain the rest of information they need to continue performing vulnerability discovery tasks.
- *Install the web sites in web servers that are not properly secured* and try to improve the security of the whole infrastructure by using adequate third party products (like `mod_security` [34]) without modifying the

source code of the installed webs. This scenario is proposed to be used in the *Web Server Administration* course of the Spanish official *Master on Web Engineering* of the *University of Oviedo* [33].

- *Test the precision of automatic vulnerability discovery tools* [35, 36] by using them against our web applications [37]. This scenario is also proposed to be used in the previously mentioned course.

Another very interesting usage scenario for *FileSync* and *Era Literaria* is *incorporating end-to-end automated test suites* [38]. Test automation allows from fast, unattended execution of a set of tests. This approach is much more efficient than the traditional manual interaction through the web application to check if it behaves correctly. Normally, end-to-end web testing is one of the main approaches for assuring the quality of web applications. As its goal to detect failures (deviations from expected behaviors), they can be also used when these failures are provoked by unfolding attacks, instead of being a consequence of source code errors. This also allows designing activities closer to game or challenge-based tasks, similar to the approach followed by some of the related web sites that will be described on the next section.

Test suites can be designed to operate GUI components, retrieve information and compare it with expected behaviors using assertions. GUI operation can be performed via *Capture-Relay* web testing or using a *Programmable* approach. The first approach is cheaper to write [38], and their potential maintenance problems are counteracted in our scenario because *FileSync* and *Era Literaria* do not really evolve over time. This also removes the necessity of repairing test cases due to logical or structural changes. The second approach eases the creation of test cases that perform multiple calls with different arguments and also facilitates creating more flexible test cases. To locate visual elements to be tested, a *DOM-based* or *Visual* approach can be used. As *Era Literaria* and *FileSync* do not evolve, choosing the appropriate test case development method or element location technique largely depends on the goals of the test cases to be designed. Finally, test cases have not to be comprehensive, but target only those areas that are expected to be attacked to check their behavior and detect the effects of a vulnerability.

This way, several different security-related end-to-end test cases can be proposed to check if the web application security have been violated in different ways within a specified time frame. Once the time ends, students or teams of students can be assigned a score depending on the amount of test cases of the suite that have been broken. This approach has multiple advantages:

- A wide variety of different test cases can be created based on real security problems, thanks to the realistic nature of the webs.
- A competitive element is introduced if this is part of the assessment of a security course. This can be done by letting the students know in advance the description of the test cases they have to fulfill, their expected difficulty, their score, and how many different test cases are available. Introducing a certain degree of competition have been proven to be a successful tool to improve the assessment of security-related courses [39].
- They can be an objective and fast way of assigning scores to students.
- They can be structured to be something like the highly popular CTF competitions [40] among students.

Some ideas for potential test cases regarding altering the functionality of the applications through attacks are:

- Disable the access of registered users by changing their passwords, with a special score assigned to disabling the access to the `admin` user. This can be checked with test cases that try to log in with a known valid user id/password but find that the access is forbidden.
- Artificially change the review score of a certain book in *Era Literaria* to 0 or 5 starts, by injecting scripts that alter this information as part of an attack. This can be checked with test cases searching for certain scores in predetermined books, showing the effects of putting false information on web sites through attacks.
- Upload a malicious file in *FileSync* that creates a `.txt` file accessible via URL with private user or system information. The test case should check that this file exists to validate that the attack has been successful, and may also check if it contains predetermined sensible information.

Test cases to know if the students have implemented valid countermeasures against known vulnerabilities can be also designed: try to check if dangerous meta files have been removed, known weak user password have been detected and replaced with stronger ones, or error pages that give too much information have been converted to more suitable ones. Therefore, a great deal of different test cases can be developed following different approaches depending on the course goals and contents, so we think that this alternative way of using both web sites is very promising.

5 Related Work

There is a great amount of deliberately vulnerable web sites that are used as training web pentesting tools [15, 16]. We classified these tools in three different groups: *pure web learning tools*, *simulated web sites*, and *realistic web sites*.

Pure web learning tools dedicate each of its subsections to test and study a concrete type of vulnerability. These websites are collections of these subsections, which typically are independent. This way, we can find web sites with a separate *XSS* and a *SQL Injection* section, but each one manages different data, have heterogeneous UI, or functionality design. Therefore, these tools are not realistic in its behavior and interface. They allow its users to familiarize with different types of vulnerabilities but does not focus in how to discover them examining the behavior of a real website. Typically, users already know the type of errors that they may find in each section, so they must locate and exploit them. In this category we can also include challenge or game-based web sites. Examples of these types of applications are *Damn Vulnerable Web App* (DVWA) [41], *bWapp* [42], *OWASP Bricks* [43], *WebGoat* [44], the security challenges system *OWASP Hackademic* [45] or the game-based *Game of Hacks* [46].

Simulated web sites focus on simulating a coherent purpose among all their subsections but lacks some features that makes them feel unrealistic. For example, their complexity may be low, data persistence is non-present (the same data are re-created on successive executions, changes are not persisted), or some functionalities are either not implemented or compromise its realism to better illustrate a concrete type or vulnerability (for example, embedding educational explanations of vulnerabilities in the GUI). Examples of this type of web sites are *BodgeIt Store* (a typical shop) [47], *Wacko Picko* (an image gallery) [48], *Hacme Casino* [49] (online casino), or *Peruggia* (another image gallery) [50].

Moreover, *realistic web sites* are much closer to the ones described in this paper. They are full-size realistic web sites, designed with a credible main purpose, persistent data and real DBMS system backing it. Working with them is almost identical to work with a production web application with the same goals. The educational material and documentation of these websites are usually placed in separate documents or sites, in order to not to compromise its realism. Sometimes discovering vulnerabilities requires pure user skill, as no documentation is provided. In this category we can mention *Hackazon* from *Rapid7* [51], that simulates a realistic and professional web shop front-end

and also allows an authorized user to inject certain types of vulnerabilities on some parts. We can also mention two webs developed by *Intel Security*: *Hacme Bank* [52] (online banking) and *Hacme Books* [53] (book shop). Their main disadvantage is that they use old development technologies with more than 10 years old. Apart from installation problems, this means that some of their vulnerabilities or attack vectors are not common nowadays. This also happens with *Hacme Casino* [49].

Finally, the importance of these tools also encourages projects that reunite multiple vulnerable web sites in the same virtual appliance to be used in security courses. Examples are the *OWASP Broken Web Applications Project* [54], or *Vulnerable Web Apps* [55].

6 Conclusions and Future Work

In this paper we describe how we created two realistic and deliberately vulnerable websites to facilitate educational and training tasks on the web security field. These sites implement a large set of the currently most critical security vulnerabilities. They also use different popular development technologies and third-party supporting tools. We complement them with detailed documentation about the introduced vulnerabilities, how they work, and how to find and solve them. The open source character of the webs also allows their users to adapt them to any particular need, develop new versions, train secure programming skills, or incorporate new functionality. Therefore, we think they are very adequate platforms to train security skills in widely different scenarios.

Future work will fully translate both webs and complementary documentation to English. We also plan to introduce brand new vulnerabilities extracted from the OWASP classification, enhancing their functionality. We are also planning to develop new websites using the same design principles based on *Java* technologies and *Node.js*. Another line of work will be creating end-to-end test suites to use the webs as learning tools that follow a more game or challenge-based approach. Finally, these tools are proposed to be used in the *Computer System Security* course of the *Degree on Software Engineering* of the *University of Oviedo* [56], the *Web Security Systems* and the *Web Server Administration* courses of the *Master on Web Engineering* [33], and in some courses of the future *Master on Cybersecurity in Software Engineering* [57], of the same university.

Source code, installation and operations manuals, and the educational guide of these webs is accessible through *GitHub*, in the following URL: https://github.com/jose-r-lopez/SecureWebs.

Acknowledgements

This work was supported in part by the European Union through the European Regional Development Funds and in part by the Principality of Asturias through Its Science, Technology, and Innovation Plan under Grant GRUPIN14-100.

References

[1] Microsoft (2018). *The official Office 365 home page*, https://products.office.com/es-ES, Accessed: 2018-05-21.

[2] Google Inc. (2018). *The official Gmail home page*, https://www.google.com/intl/es/gmail/about, Accessed: 2018-05-21.

[3] Amazon (2018). *The official Amazon home page*, https://www.amazon.com, Accessed: 2018-05-21.

[4] Ebay (2018). *The official Ebay home page*, http://www.ebay.com, Accessed: 2018-05-21.

[5] Redondo, J. M., and Ortin, F. (2017). *A SaaS Framework for Credit Risk Analysis Services*, IEEE Latin America Transactions, Vol. 15, No. 3.

[6] Microsoft (2018). *The official Microsoft Dynamics home page*, http://www.dynamics-crm.es, Accessed: 2018-05-21.

[7] Lonescu, P. (2015). *The 10 Most Common Application Attacks in Action*, https://securityintelligence.com/the-10-most-common-application-attacks-in-action/, Accessed: 2018-05-21.

[8] ICT S. Magazine (2018). *2017 Data Breach Investigations Report 10th Edition*, http://www.ictsecuritymagazine.com/wp-content/uploads/2017-Data-Breach-Investigations-Report.pdf, Accessed: 2018-05-15.

[9] OWASP (2018). *OWASP Top Ten Project*, https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, Accessed: 2018-05-21.

[10] MITRE Corporation (2018). *CVE Details: The ultimate security vulnerability data-source*, https://www.cvedetails.com/, Accessed: 2018-05-21.

[11] OWASP (2017). *OWASP Secure Coding Practices - Quick Reference Guide*, https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide, Accessed: 2018-05-21.

[12] Mozilla Corporation (2012). *WebAppSec/Secure Coding Guidelines*, https://wiki.mozilla.org/WebAppSec/Secure_Coding_Guidelines, Accessed: 2018-05-21.

[13] Morgan, S. (2016). *One Million Cybersecurity Job Openings In 2016*, http://www.forbes.com/sites/stevemorgan/2016/01/02/one-million-cybersecurity-job-openings-in-2016/, Accessed: 2018-05-21.

[14] Bacudio, A. G., Yuan, X., Chu, B. B., and Jones, M. (2011). *An Overview of Penetration Testing*, International Journal of Network Security and Its Applications, Vol. 3, No. 6
[15] Vonnegut, S. (2015). *15 Vulnerable Sites To (Legally) Practice Your Hacking Skills*, https://www.checkmarx.com/2015/04/16/15-vulnerable-sites-to-legally-practice-your-hacking-skills, Accessed: 2018-05-21.
[16] S. Vonnegut (2015). *13 More Hacking Sites to (Legally) Practice Your InfoSec Skills*, https://www.checkmarx.com/2015/11/06/13-more-hacking-sites-to-legally-practice-your-infosec-skills, Accessed: 2018-05-21.
[17] Tiobe (2018). *Tiobe Index official home page*, http://www.tiobe.com/tiobe-index, Accessed: 2018-05-21.
[18] PYPL (2018). *PYPL PopularitY of Programming Language*, http://pypl.github.io/PYPL.html, Accessed: 2018-05-21.
[19] W3 Consortium (2008). *Web Content Accessibility Guidelines (WCAG) 2.0*, https://www.w3.org/TR/WCAG20, Accessed: 2018-05-21.
[20] Citrix (2018). *Sharefile: The Simple, Secure Way to Collaborate*, https://www.sharefile.com, Accessed: 2018-05-21.
[21] El Corte Ingles (2018). *Libreria El Corte Ingles*, https://www.elcorteingles.es/libros, Accessed: 2018-05-21.
[22] OWASP (2016). *OWASP Testing Guide v4*, https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents, Accessed: 2018-05-21.
[23] Yii Software LLC (2018). *Yii framework: The Fast, Secure and Professional PHP Framework*, http://www.yiiframework.com, Accessed: 2018-05-21.
[24] Microsoft (2014). *ASP.NET MVC 5 official home page*, https://www.asp.net/mvc/mvc5, Accessed: 2018-05-21.
[25] Qualys SSLabs (2018). *How well do you know SSL?*, https://www.ssllabs.com/, Accessed: 2018-05-20.
[26] OWASP (2017). *Session Management Cheat Sheet*, https://www.owasp.org/index.php/Session_Management_Cheat_Sheet, Accessed: 2018-05-20.
[27] N. Kaur, P. Kaur (2014). *Input Validation Vulnerabilities in Web Applications*, Journal of Software Engineering, Vol. 8(3), pp. 116–126.
[28] OWASP (2018). *Unrestricted File Upload*, https://www.owasp.org/index.php/Protect_FileUpload_Against_Malicious_File, Accessed: 2018-05-15.
[29] OWASP (2018). *Protect FileUpload Against Malicious File*, https://www.owasp.org/index.php/Unrestricted_File_Upload, Accessed: 2018-05-15.
[30] Portswigger (2018). *Burp Suite*, https://portswigger.net/burp, Accessed: 2018-05-15.
[31] XeusHack (2018). *Zip Bomb*, https://xeushack.com/zip-bomb/, Accessed: 2018-05-15.
[32] BeeF (2018). *BeeF: The Browser Exploitation Framework Project*, http://beefproject.com, Accessed: 2018-05-21.
[33] de Oviedo, U. (2018). *Master Universitario en Ingenieria Web*, https://ingenieriainformatica.uniovi.es/infoacademica/masterydoctorado, Accessed: 2018-05-21.
[34] Trustwave SpiderLabs (2018). *ModSecurity: Open Source Application Firewall*, https://modsecurity.org, Accessed: 2018-05-21.
[35] OpenVAS (2018). *Open Vulnerability Assessment System*, http://www.openvas.org, Accessed: 2018-05-21.

[36] Tenable (2018). *Nessus Vulnerability Scanner*, https://www.tenable.com/products/nessus-vulnerability-scanner, Accessed: 2018-05-21.

[37] Doupe, A. Cova, M. and Vigna, G. (2010). *Why Johnny Cannot Pentest: An Analysis of Black-Box Web Vulnerability Scanners*, Lect. Noter on Comp. Science, Vol. 6201, pp. 111–131.

[38] Leotta, M. Clerissi, D. Ricca, F. Tonella, P. (2016). *Approaches and Tools for Automated End-to-End Web Testing*, Advances in Computers, Vol. 101, pp. 193–237.

[39] Redondo, J. M. (2018). *Improving Student Assessment of a Server Administration Course Promoting Flexibility and Competitiveness*, IEEE Transactions on Education, Article in Press, doi: 10.1109/TE.2018.2816571

[40] CTFTime (2018). *CTFs*, https://ctftime.org/ctfs, Accessed: 2018-05-17.

[41] DVWA (2018). *Damn Vulnerable Web Application*, http://www.dvwa.co.uk, Accessed: 2018-05-21.

[42] Mesellem, M. (2014). *bWapp: an extremely buggy web app!*, http://www.itsecgames.com, Accessed: 2018-05-21.

[43] OWASP (2016). *OWASP Bricks*, https://www.owasp.org/index.php/OWASP_Bricks, Accessed: 2018-05-21.

[44] OWASP (2018). *OWASP WebGoat project*, https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project, Accessed: 2018-05-21.

[45] OWASP (2015). *OWASP Hackademic Challenges Project*, https://www.owasp.org/index.php/OWASP_Hackademic_Challenges_Project, Accessed: 2018-05-21.

[46] CheckMarx (2018). *Game of Hacks: See how good you are*, http://www.gameofhacks.com/, Accessed: 2018-05-21.

[47] Bennets, S. (2016). *BodgeIt store*, https://github.com/psiinon/bodgeit, Accessed: 2018-05-21.

[48] Doupe, A. (2013). *WackoPicko web site*, https://github.com/adamdoupe/WackoPicko, Accessed: 2018-05-21.

[49] Intel Security (2006). *Hacme Casino v1.0*, https://www.mcafee.com/es/downloads/free-tools/hacme-casino.aspx, Accessed: 2018-05-21.

[50] Kramer, A. (2015). *Peruggia web site*, https://sourceforge.net/projects/peruggia, Accessed: 2018-05-21.

[51] Davis, S. L. (2016). *Hackazon: a modern vulnerable web app*, https://github.com/rapid7/hackazon, Accessed: 2018-05-21.

[52] Intel Security (2006). *Hacme Bank v2.0*, https://www.mcafee.com/es/downloads/free-tools/hacme-bank.aspx, Accessed: 2018-05-21.

[53] Intel Security (2006). *Hacme Books v2.0*, https://www.mcafee.com/es/downloads/free-tools/hacmebooks.aspx, Accessed: 2018-05-21.

[54] OWASP (2017). *OWASP Broken Web Applications Project*, https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project, Accessed: 2018-05-21.

[55] Santander, M. (2015). *Vulnerable Web Apps VMWare appliance*, https://sourceforge.net/projects/vapps/files, Accessed: 2018-05-21.

[56] de Oviedo, U. (2018). *Escuela de Ingenieria Informatica*, https://ingenieriainformatica.uniovi.es/infoacademica/grado, Accessed: 2018-05-21.

[57] de Oviedo, U. (2018). *Master en Ciberseguridad en Ingenieria del Software*, http://www.mcis.uniovi.es/, Accessed: 2018-05-15.

Biographies

Jose Manuel Redondo López is an Assistant Professor in the University of Oviedo, Spain since November 2003. Received his B.Sc., M.Sc., and Ph.D. degrees in computer engineering from the same university in 2000, 2002, and 2007, respectively. He participated in various research projects funded by Microsoft Research and the Spanish Department of Science and Innovation. He has authored three books and over 20 articles. His research interests include dynamic languages, computational reflection, and computer security.

Leticia del Valle Varela is a .Net application developer for the Spanish public administration in GADD Grupo Meana S. A. Received his B.Sc. and M.Sc. in computer engineering from the University of Oviedo (Spain) in 2013 and 2015 respectively. He participated in various development projects with a focus in security issues in the University of Oviedo. Also, she was a researcher in the official Computational Reflection research group (University of Oviedo, Spain). His research interests include web engineering and computer security.