Journal of Web Engineering, Vol. 0, No. 0 (2003) 000–000

FILESYNC AND ERA LITERARIA: REALISTIC OPEN SOURCE WEBS TO DEVELOP WEB SECURITY SKILLS

JOSE MANUEL REDONDO
Department of Computer Science, University of Oviedo, C/Calvo Sotelo S/N
Oviedo (Asturias), 33007, Spain
redondojose@uniovi.es

LETICIA DEL VALLE
GADD Grupo Meana S. A., Palacio de Lieres s/n
Lieres (Asturias), 33580, Spain
leticiavallevarela@gmail.com

Received (received date)
Revised (revised date)

Web sites are one of the currently preferred ways to distribute applications and services. Unfortunately, this also caused the proliferation of a wide variety of attacks targeting their potential vulnerabilities. Therefore, the demand for security-trained engineers that can identify, prevent and find solutions to them 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 several usage examples of two websites designed to facilitate web security training. These websites have a realistic set of features and are developed using different popular technologies. They incorporate examples of a large subset of common security vulnerabilities deliberately, complemented with learning and training materials. They are also open source, in order to allow the development of customizations and adaptations to different scenarios.

Keywords: Web security, pentesting, OWASP, vulnerability, training
Communicated by: to be filled by the Editorial

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].

Concurrently, potential security problems of the web have also greatly increased. As websites became more complex they also increase the probability of having security vulnerabilities, and nowadays web attacks are a major problem that must be addressed [7]. There is a great variety of web attacks that exploit different vulnerabilities, evolving and acquiring new attack vectors over time. Projects like OWASP [8] study, classify, and rank them.

In general, the origin of web vulnerabilities can be attributed to:

1. Defects/omissions/bugs in the development platforms or supporting third-party products [9].
2. Unawareness or incorrect application of the security mechanisms of the underlying infrastructure.
3. Inability to develop proper secure coding practices [10, 11].

Companies have increased the demand of web security professionals [12]. Their goal is to assess the development of secure code, applying a series of techniques to avoid exploiting parts of the web source code to unfold attacks. They also may test the security of existing web applications, applying the same techniques as malicious attackers. As a result, they create reports describing the found problems and their potential solutions. This is commonly known as *pentesting* (penetration testing) [13].

In order 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 a great number of products of this type [14, 15]; 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, 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 real ones will be very beneficial to achieve proper skills. Unfortunately, realistic security-training web sites are scarce, as their development is more complex and time-consuming. The complexity of integrating key security vulnerabilities without compromising their goals is added to the development effort. This requires careful planning and implementation, and also develop its educational aspects without sacrificing its realism. 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. They are developed with two popular web development technologies [16, 17] 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 security problems, their possible solutions, and management instructions (installation, maintenance...). This way, we can maintain its educational value without compromising its realism and also 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 its architecture and the vulnerabilities

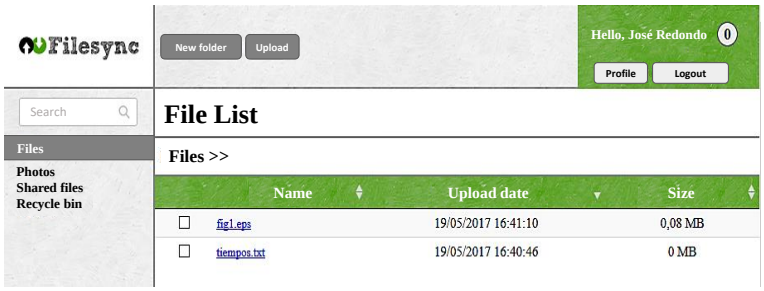

Fig. 1. FileSync interface.

introduced. 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 (*FileSync* and *Era Literaria*) were designed following a series of common rules:

- *Realism*: design, implementation and UI follow current industry standards, as current websites in production with similar goals [18]. 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 infrastructure 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. Fig. 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]. Fig. 2 shows the interface of this application.

2.1. *Vulnerability inclusion*

Implemented vulnerabilities were chosen and classified following the *OWASP Pentesting guide v4* [23]. Our selection includes several classified as *Top 10* by the OWASP organization [8] (see next subsection). Proposed solutions for vulnerabilities focus on correctly applying

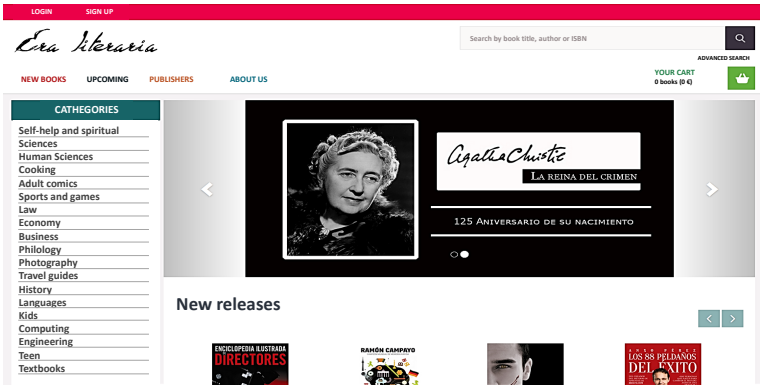

Fig. 2. Era Literaria interface.

the built-in security mechanisms of the frameworks or third party products used to build each website, if available. This way we favor using tried-and-tested security mechanisms over developing custom solutions, to guarantee the implementation of a proper solution to each vulnerability. Applying these mechanisms also requires a complete knowledge of the vulnerability they protect against, so we ensure that the student learn the most appropriate solution available once they know the scope of the problem. This also targets the second source of vulnerabilities that was mentioned in the introduction.

Finally, 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 possible solutions.

3. Website implementation

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

- *Era Literaria* is hosted in the *Apache 2* web server, using a *MySQL* database engine and *PHP 5* or *PHP 7*. Additionally, is uses *PHPMyAdmin* to manage the database. The web was built using the version 1.1.16 of the *Yii* framework.
- *FileSync* targets *Windows* operating systems, hosted in *IIS 8* (or higher) web servers. It uses a *SQL Server 2012* database engine, although it can run with the free *SQL Server Express*). Its *C#* source code uses the *MVC 5* framework.

3.1. *Included vulnerabilities*

We chose 21 vulnerabilities from the OWASP classification we considered most important to address to attain a high level of security on any 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 vulnerabilities are more varied, so students have to develop a wider range of skills to implement solutions when using

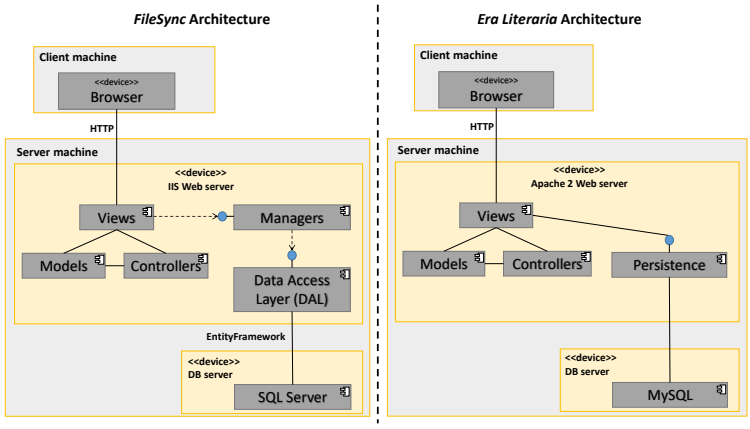

Fig. 3. *FileSync* and *EraLiteraria* architectures.

this tools. The Table 1 shows a description of each vulnerability, its common OWASP name and classification code [23].

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 [9]
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 (i. e. *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 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) [24]
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, 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
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
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

Table 1: Description of the vulnerabilities included in the websites

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 *Apache 2* 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 know. During this vulnerability discovery phase, they will reach the web page that allows to leave comments and opinions about books. This page is vulnerable to *stored XSS* attacks (OTG-INPVAL-002), as the programmers forgot to apply proper filtering to the user-delivered content in this precise point. 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* [25], an attacker can open a remote administration panel to modify the user cookie and see the actions of any user that views an attacked book comment. This attack unfolds by injecting the *BeeF hook.js* script into the comment text box of the web page. Description of this attack can be shown in the learning guide that accompanies the web page.

This scenario illustrates the dangers and the sophistication level of an XSS attack, showing the importance of avoiding these kind of vulnerabilities. This scenario is proposed to be used in the *Web Security Systems* course of the Spanish official *Master on Web Engineering* of the *University of Oviedo* [45]

4.2. *White box pentest*

We provide the students a *Windows 2016* server with *FileSync* installed in a known port and address. Additionally, we also provide them the source code of the application and its user manual. With all these materials, the students have to perform code reviews and vulnerability

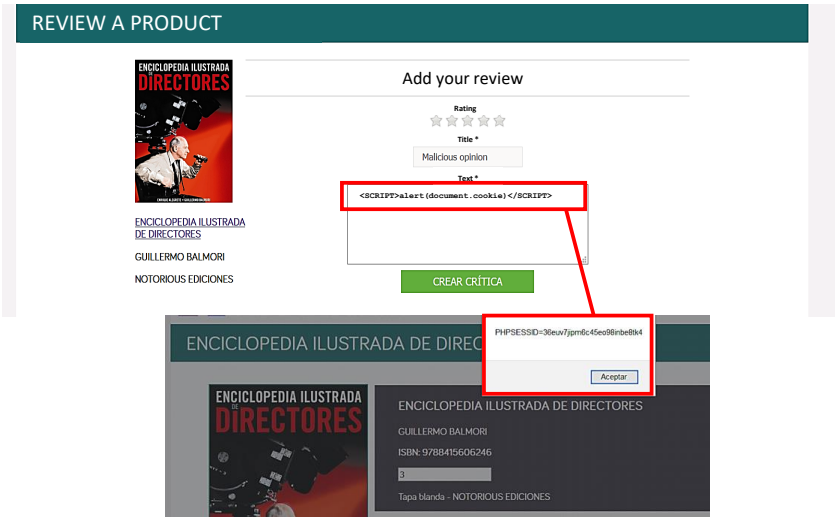

Fig. 4. Stored XSS in *EraLiteraria*.

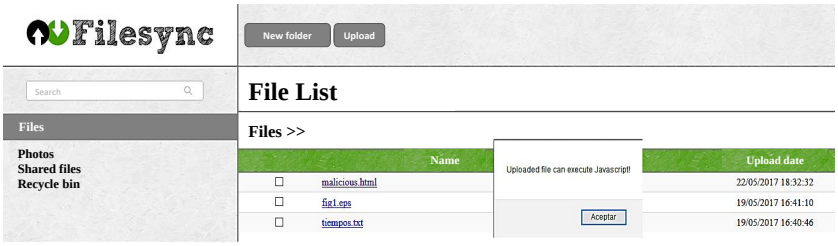

Fig. 5. Uploading a malicious web page.

discovery procedures, performing all the tests they know to determine if there are any potential ones. During this process, the students may find that the file upload functionality of this web page do not perform proper file type checking. Therefore, this page has an *Upload of Unexpected files* (OTG-BUSLOGIC-008) vulnerability, so any user can upload an HTML web page with malicious *Javascript* code, as shown in Fig. 5.

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

4.3. *Other usage scenarios*

The following list shows a description of 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 courses, 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 or developing custom secure code (enabling encryption, modifying the response...), giving focus to secure programming procedures.

- *Develop proactive responses* when certain types of attack are detected to stop them. Active countermeasuring of attacks requires the students to modify 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, rather than launch exploits.
- *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* [26]) 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* [45].
- *Test the precision of automatic vulnerability discovery tools* [27, 28] by using them against our web applications [29]. This scenario is proposed to be used in the same course than the previous one.

5. Related work

There is a great amount of deliberately vulnerable web sites that are used as training web pentesting tools [14, 15]. 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 have to focus into locating and exploiting them. In this category we can also mention challenge or game-based web sites. Examples of these types of applications are *Damn Vulnerable Web App* (DVWA) [30], *bWapp* [31], *OWASP Bricks* [32], *WebGoat* [33], the security challenges system *OWASP Hackademic* [34] 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) [35], *Wacko Picko* (an image gallery) [36], *Hacme*

Casino [39] (online casino), or *Peruggia* (another image gallery) [37].

Finally, *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 [38], 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* [40] (online banking) and *Hacme Books* [41] (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* [39].

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* [42], or *Vulnerable Web Apps* [43].

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 solve them. The open source character of the webs also allows their users to adapt them to any particular need, develop new versions, 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 both web sites functionality. We are also planning to develop new websites using the same design principles. Concretely, we are planning to develop two realistic websites based on *Java* technologies and *Node.js*. 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* [44], and also in the *Web Security Systems* and the *Web Server Administration* courses of the *Master on Web Engineering*, of the same university [45].

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

Acknowledgements

This work has been funded by the European Union, through the European Regional Development Funds (ERDF); and the Principality of Asturias, through its Science, Technology and Innovation Plan (Grant GRUPIN14-100)

References

1. Microsoft (2017), *The official Office 365 home page*, `https://products.office.com/es-ES`, Accessed: 2017-11-22.

2. Google Inc. (2017), *The official Gmail home page*, https://www.google.com/intl/es/gmail/about, Accessed: 2017-11-22.
3. Amazon (2017), *The official Amazon home page*, https://www.amazon.com, Accessed: 2017-11-22.
4. Ebay (2017), *The official Ebay home page*, http://www.ebay.com, Accessed: 2017-11-22.
5. J. M. Redondo and F. Ortin (2017), *A SaaS Framework for Credit Risk Analysis Services*, IEEE Latin America Transactions, Vol. 15, No. 3.
6. Microsoft (2017), *The official Microsoft Dynamics home page*, http://www.dynamics-crm.es, Accessed: 2017-11-22.
7. P. Lonescu (2015), *The 10 Most Common Application Attacks in Action*, https://securityintelligence.com/the-10-most-common-application-attacks-in-action/, Accessed: 2017-11-22.
8. OWASP (2017), *OWASP Top Ten Project*, https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, Accessed: 2017-11-22.
9. MITRE Corporation (2017), *CVE Details: The ultimate security vulnerability datasource*, https://www.cvedetails.com/, Accessed: 2017-11-22.
10. OWASP (2017), *OWASP Secure Coding Practices - Quick Reference Guide*, https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide, Accessed: 2017-11-22.
11. Mozilla Corporation (2017), *WebAppSec/Secure Coding Guidelines*, https://wiki.mozilla.org/WebAppSec/Secure_Coding_Guidelines, Accessed: 2017-11-22.
12. S. Morgan (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: 2017-11-22.
13. A. G. Bacudio, X. Yuan, B. B. Chu and M. Jones (2011), *An Overview of Penetration Testing*, International Journal of Network Security and Its Applications, Vol. 3, No. 6
14. S. Vonnegut (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: 2017-11-22.
15. 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: 2017-11-22.
16. Tiobe (2017), *Tiobe Index official home page*, http://www.tiobe.com/tiobe-index, Accessed: 2017-11-22.
17. PYPL (2017), *PYPL PopularitY of Programming Language*, http://pypl.github.io/PYPL.html, Accessed: 2017-11-22.
18. W3 Consortium (2017), *Web Content Accessibility Guidelines (WCAG) 2.0*, https://www.w3.org/TR/WCAG20, Accessed: 2017-11-22.
19. Microsoft (2017), *ASP.NET MVC 5 official home page*, https://www.asp.net/mvc/mvc5, Accessed: 2017-11-22.
20. Citrix (2017), *Sharefile: The Simple, Secure Way to Collaborate*, https://www.sharefile.com, Accessed: 2017-11-22.
21. El Corte Ingles (2017), *Libreria El Corte Ingles*, https://www.elcorteingles.es/libros, Accessed: 2017-11-22.
22. Yii Software LLC (2017), *Yii framework: The Fast, Secure and Professional PHP Framework*, http://www.yiiframework.com, Accessed: 2017-11-22.
23. OWASP (2017), *OWASP Testing Guide v4*, https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents, Accessed: 2017-11-22.
24. Symantec (2017), *Advanced Persistent Threats: How They Work*, https://www.symantec.com/theme.jsp?themeid=apt-infographic-1, Accessed: 2017-11-22.
25. BeeF (2017), *BeeF: The Browser Exploitation Framework Project*, http://beefproject.com, Accessed: 2017-11-22.
26. Trustwave SpiderLabs (2017), *ModSecurity: Open Source Application Firewall*, https://

modsecurity.org, Accessed: 2017-11-22.
27. OpenVAS (2017), *Open Vulnerability Assessment System*, http://www.openvas.org, Accessed: 2017-11-22.
28. Tenable (2017), *Nessus Vulnerability Scanner*, https://www.tenable.com/products/nessus-vulnerability-scanner, Accessed: 2017-11-22.
29. A. Doupe, M. Cova and G. Vigna (2010), *Why Johnny Cant Pentest: An Analysis of Black-Box Web Vulnerability Scanners*, Lect. Noter on Comp. Science, Vol. 6201, pp. 111–131
30. DVWA (2017), *Damn Vulnerable Web Application*, http://www.dvwa.co.uk, Accessed: 2017-11-22.
31. M. Mesellem (2014), *bWapp: an extremely buggy web app!*, http://www.itsecgames.com, Accessed: 2017-11-22.
32. OWASP (2013), *OWASP Bricks*, https://www.owasp.org/index.php/OWASP_Bricks, Accessed: 2017-11-22.
33. OWASP (2017), *OWASP WebGoat project*, https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project, Accessed: 2017-11-22.
34. OWASP (2016), *OWASP Hackademic Challenges Project*, https://www.owasp.org/index.php/OWASP_Hackademic_Challenges_Project, Accessed: 2017-11-22.
35. S. Bennets (2016), *BodgeIt store*, https://github.com/psiinon/bodgeit, Accessed: 2017-11-22.
36. A. Doupe (2013), *WackoPicko web site*, https://github.com/adamdoupe/WackoPicko, Accessed: 2017-11-22.
37. A. Kramer (2015), *Peruggia web site*, https://sourceforge.net/projects/peruggia, Accessed: 2017-11-22.
38. S. L. Davis (2016), *Hackazon: a modern vulnerable web app*, https://github.com/rapid7/hackazon, Accessed: 2017-11-22.
39. Intel Security (2006), *Hacme Casino v1.0*, https://www.mcafee.com/es/downloads/free-tools/hacme-casino.aspx, Accessed: 2017-11-22.
40. Intel Security (2006), *Hacme Bank v2.0*, https://www.mcafee.com/es/downloads/free-tools/hacme-bank.aspx, Accessed: 2017-11-22.
41. Intel Security (2006), *Hacme Books v2.0*, https://www.mcafee.com/es/downloads/free-tools/hacmebooks.aspx, Accessed: 2017-11-22.
42. OWASP (2017), *OWASP Broken Web Applications Project*, https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project, Accessed: 2017-11-22.
43. M. Santander (2015), *Vulnerable Web Apps VMWare appliance*, https://sourceforge.net/projects/vapps/files, Accessed: 2017-11-22.
44. U. de Oviedo (2017), *Escuela de Ingenieria Informatica*, https://ingenieriainformatica.uniovi.es/infoacademica/grado, Accessed: 2017-11-22.
45. U. de Oviedo (2017), *Master Universitario en Ingenieria Web*, https://ingenieriainformatica.uniovi.es/infoacademica/masterydoctorado, Accessed: 2017-11-22.
46. CheckMarx (2017), *Game of Hacks: See how good you are*, http://www.gameofhacks.com/, Accessed: 2017-11-22.