An Overview Over the Open Source Resources for Web Applications Security

Emerson Assis de Carvalho[1], **Fernanda Ramos de Carvalho**[2], **Lucyara Silva Ribeiro**[2], **Germano Estevam Simão Pereira**[2], **Tulio Cesar Lopes Alves**[2]

[1]Departamento de Informátiva
IFSULDEMINAS - Campus Passos
Rua Mario Ribola, 409 - Penha II - Passos/MG

[2]Departamento de Ciência da Computação
Universidade José do Rosário Vellano
Rod. MG 179, Km 0, Câmpus Universitário - Alfenas/MG

emerson.carvalho@ifsuldeminas.edu.br

{frcarvalho,lucyarasr,germanoesp}@gmail.com,tuliocesar@hotmail.com

Abstract. *This work presents a web application security overview, presenting its main concepts and areas, the open source resources available, the most common web security vulnerabilities and how to prevent them. We also have used some open source web application security scanners to test the security of a simple web application. We have used more than one scanner, aiming to have a complete report over the vulnerabilities and to make a comparison between them. We have used a web application previously developed without any concern about security. Our reports were on the vulnerabilities found and how much was easy or not to interpret and fix them.*

Resumo. *Este trabalho apresenta uma análise sobre a segurança de aplicatições web, apresentando os principais conceitos, os recursos open source disponíveis, as vulnerabilidades mais comuns em aplicações web e como se previnir delas. Foram utilizados alguns scanners open source para testar a segurança de uma aplicação web simples. A utlização de mais de um scanner teve como objetivo alcançar um alto nível de detecção de problemas de segurança, bem como realizar uma comparação entre as ferramentas utilizadas. Para os testes, foi utilizada uma aplicação web desenvolvida previamente sem nenhuma precaução sobre segurança. Nossos resultados focaram nas vulnerabilidades encontradas e no trabalho de interpretação e correção das mesmas.*

1. Introduction

Nowadays, Computer Networks is one of the key technologies. It allows connections between elements ranging from routers and servers that host websites to small mobile devices. The global connectivity is the core principle of our information age and vital for our economy today (O'NEILL, 2014).

Our heavy dependence of web applications make a security breach in these kind of applications, that may range from financial losses to dangers to human life. Information

Security is the preservation of the confidentiality, integrity and availability of information, no matter its form, it can be printed on paper, stored in a electronic device, transmitted by a network and more (ISO/IEC, 2005). It is the aspects regarding to defining and maintaining the confidentiality, integrity, availability, non-repudiation, accountability, authenticity, and reliability of information technology resources (ISO/IEC, 2004; SOLMS; NIEKERK, 2013). The main objective of Information Security is to properly protect information from unauthorized access, use, disclosure, disruption, modification and destruction (MELLADO et al., 2010).

In the era of cloud computing and mobile devices, so many people, schools and companies are doing business, storing critical information, conducting researches, cooperating with each other and publishing various types of contents through web applications. Web applications are the Achilles heel of information technology security, once users can run sophisticated web applications from any PC, netbook, tablet or smartphone. Web applications are now the most prevalent of all server vulnerability disclosures (SHEMA, 2011).

Web applications are easy to access, easy to deploy and, unfortunately, they also are easy to attack (THOMÉ; GORLA; ZELLER, 2014). They contain many vulnerabilities, which may lead to security breaches such as stealing of confidential information and denial of services. To protect against these security risks, it is necessary to understand the steps of attacks and the pros and cons of existing defense mechanisms (SHAHRIAR, 2013). Many industry incident reports, such as, the Verizon Data Breach Incidents Report and the FireHost Attack Report, are in alignment that websites and web applications remain one of the leading targets of cyber-attacks. A recent research of WhiteHat Secutiry has showed that, from a security perspective, knowing the security mechanisms and techniques is more important than the language/platform choice (WHITEHATSECURITY, 2014).

The mechanisms to protect web applications are different from those used to protect networks and internal users systems. Techniques like firewalls, cryptography and Intrusion Detection Systems do little to protect web applications, since organizations have to allow customers access their networks to interact with their web applications (bank, shopping, entertainment, news etc). The security systems cannot block network connections at the firewall and encrypting connections to the web application can not prevent attacks like cross-site scripting (XSS) or SQL injection. Web applications security need to be treated at the application layer, either designing and coding applications trying to avoid vulnerabilities and protecting the application with tools like web application firewalls (WALDEN, 2008).

Web vulnerability scanners are a very popular choice for finding security vulnerabilities in web applications. These tools are capable to test any web application, regardless of the back-end language, finding common security vulnerabilities (DOUPÉ et al., 2012). These tools operate by simulating attacks against web applications and observing its responses. They perform a black-box testing, when the source code is not examined directly, instead, special inputs are generated and sent to the application and the results are analyzed for unexpected behaviors that indicate possible errors or vulnerabilities (KALS et al., 2006).

This paper aims to explain the most common vulnerabilities found in web applications, how to avoid such vulnerabilities, as well as how to detect this security vulnerabilities using scanning tools. Since there are many vulnerability scanners and it is difficult to choice the appropriate one, this paper also presents a comparison over some of the main open source scanning tools. The next section presents the most common web application vulnerabilities. Section III presents the web application security standards designed to prevent such vulnerabilities. Section IV presents the open source tools available to test web application security, while Section V shows some web application scanning results and Section VI comes with our last considerations.

2. The most common web application vulnerabilities

Vulnerabilities are flaws found in softwares as a result of possible errors in theirs design, implementation or settings. They are the paths that attackers can potentially use through applications to spoil people or organization, as well as the business impact related to each exploited risk (OWASP, 2013). The main goal of the web security is to test the security of all web application's functionality (SHEMA, 2011). This mean that is necessary to code with security in mind, knowing the web application security standards and practices and the most common application security risks, as well as the well known techniques and recommended remediation mechanisms.

Security in web applications is the safety of implementation code, third part libraries and also the own web servers. There are several organizations that establishes a baseline of security requirements aiming to provide coding standards, which are based on accepted industry practices, to treat security exploits due to improper coding practices. These organization also provides references about common web security vulnerabilities and how to remediate them properly. Among these organizations we can cite:

- Open Web Application Security Project (OWASP): an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted. All tools, documents, forums, and chapters are free and open to anyone interested in improving application security (OWASP, 2015c).
- SysAdmin, Audit, Network, Security Institute (SANS): a cooperative research and education organization with a range of individuals from auditors and network administrators, to chief information security officers that are sharing the lessons they learn and are jointly finding solutions to the challenges they face. At the heart of SANS are the many security practitioners in varied global organizations from corporations to universities working together to help the entire information security community (SANS, 2015).
- Web Application Security Consortium (WASC): an international group of experts, industry practitioners, and organizational representatives who produce open source and widely agreed upon best-practice security standards for the web. Consistently releases technical information, contributed articles, security guidelines, and other useful documentation. Businesses, educational institutions, governments, application developers, security professionals and software vendors use their materials to assist with the challenges presented by web application security (WASC, 2015).

- Common Weakness Enumeration (CWE): provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design (CWE, 2015).
- Software Assurance Forum for Excellence in Code (SAFECode): a non-profit organization exclusively dedicated to increasing trust in information and communications technology products and services through the advancement of effective software assurance methods. It is a global industry-led effort to identify and promote best practices for developing and delivering more secure and reliable software, hardware and services (SAFECODE, 2015b).
- Common Vulnerabilities and Exposures (CVE-MITRE): a dictionary of common names (CVE Identifiers) for publicly known information security vulnerabilities. These identifiers make easier to share data across separate network security databases and tools, and provide a baseline for evaluating the coverage of an organization security tools (CVE, 2015) and (MITRE, 2016).
- National Vulnerability Database (NVD-NIST): the U.S. government repository of standards based vulnerability management data. Enables automation of vulnerability management, security measurement, and compliance. Includes databases of security checklists, security related software flaws, misconfigurations, product names and impact metrics (NIST, 2015b);
- Center for Internet Security (CIS): a non-profit organization focused on enhancing the cyber security readiness and response of public and private sector entities, with a commitment to excellence through collaboration. Produces consensus based best practice secure configuration benchmarks and security automation content. Serves as the key cyber security resource for state, local, territorial and tribal governments (CIS, 2015).

The OWASP Top 10 Web Application Security Risks document (OWASP, 2013) has identified the ten most critical web application security risks and suggested remediation to them. These risks will be explained following OWASP ordering.

2.1. Injection

Occur when an external entity sends untrusted data to an internal interpreter. Consider anyone who can send untrusted data to the system. This can include external users, internal users, and administrators. They are often found in SQL and/or NoSQL queries, Xpath expressions, OS commands, XML parsers, SMTP/HTTP headers and more. Injections are easy to discover examining the source code, but hard to discover via testing. Automated scanners can help finding injection flaws and can be used by security engineers or attackers.

2.2. Broken Authentication and Session Management

Occur when attackers exploit flaws in the authentication mechanisms (exposed accounts, users, passwords, session IDs) to impersonate users. To build custom authentication management schemes is hard, leading to flaws in areas such as logout, password management, timeouts, remember me functions and more. Finding these kind of flaws can be difficult, due different implementations schemes.

2.3. Cross-Site Scripting (XSS)

Occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping. There are three types of XSS: 1) **stored**, those where the injected script is permanently stored on the target, 2) **reflected**, those where the injected script is reflected off the web server, such as clicking on a malicious link, and 3) **DOM based XSS**, those where the entire tainted data flow takes place in the browser. Finding XSS flaws can be easy via testing or code analysis.

2.4. Insecure Direct Object References

Occur when applications use the actual name or key of an object when generating web pages and do not verify if the user have authorized access for the target object. Such flaws can compromise all the data that can be referenced by a parameter. Code analysis shows when authorization is properly verified.

2.5. Security Misconfiguration

Occur when the application stack has some errors in its configuration, such as platform, web server, application server and database settings. Some examples are unprotected files, the use of default accounts and passwords. Automated scanners are useful for detecting these kind of flaws.

2.6. Sensitive Data Exposure

Occur when there is no encrypting for sensitive data or when cryptography is employed with: (1) weak key generation and management, (2) weak algorithms or (3) weak password hashing techniques. Attackers generally do not break cryptography directly, they break other mechanisms before, like steal keys or clear text data off the server. It is have difficulty to detect server side flaws due to limited access and they are also usually hard to exploit.

2.7. Missing Function Level Access Control

Occur when applications do not protect its functions properly. If the function level protection is managed via configuration, the system must be configured properly. If the function level protection is managed by code, developers must include code checks properly. Finding such flaws is easy, the hardest part is identifying which URL's or functions exist.

2.8. Cross-Site Request Forgery (CSRF)

Occur when web applications allow attackers to predict details of a particular action, allowing the creation of malicious web pages which generate forged requests that are indistinguishable from legitimate ones. Attackers can forge HTTP requests and persuades a victim to submit them via image tags, e-mail, XSS and more. These kind of attacks will succeeds if the user is authenticated. Finding CSRF flaws is easy via penetration tests or code analysis.

2.9. Using Components with Known Vulnerabilities

Occur when the third part components, like frameworks and libraries, are not up to date. Some knowing vulnerable components can be identified and exploited. Most developers do not care if the third part components are up to date. Actually, many of them do not know all the components they are using or their versions. The dependencies between components make things even worse.

2.10. Unvalidated Redirects and Forwards

Occur when applications redirect users to other pages, or even internal forwards, and the target page is specified in a parameter that was not validated. This allow attackers to choose the destination pages, linking to malicious targets. The victims are likely to click on this links, since they are in a trusted site.

2.11. Other issues

There are many other issues that could affect the security of web applications. There are also essential knowledge for web application developers who aims to develop security web applications. Information about how to effectively design, develop, test, and find vulnerabilities in web applications can be found in (OWASP, 2015f, 2015m, 2015g; CWE/SANS, 2015) and (SAFECODE, 2015a).

3. Web Application Security Standards and Practices

Web security standards and practices specifies coding standards and basic security practices that must be followed when developing web applications. OWASP has a Developer Guide (OWASP, 2015l) that intend to be a first principles book about how to implement secure software. The Application Security Verification Standard Project (OWASP, 2015a) is other important OWASP project, aiming to normalize the range in the coverage and level of rigor when performing web application security verification.

The complete security for a web application is very difficult, but there are some strategies that can be used by developers and users to keep their applications secure. In this section we will focus on recommended techniques to prevent the most critical web application security risks. These techniques are recommended by (OWASP, 2015c; CWE, 2015; CVE, 2015) and (CIS, 2015).

3.1. Injection Preventions

Injection preventions requires keeping untrusted data separated from commands and queries. An analises of the source code is an accurate way to see if the application uses the interpreters properly. Code analysis tools can help to find the use of interpreters. Dynamic scanners may verify if some exploitable injection flaws exist, but they can not always reach interpreters and have difficulty detecting whether an attack was successful. Some injection preventions are:

- Ensure that all external commands called by the application are statically created;
- All input must be considered malicious. A validation strategy like a white list of acceptable inputs must be implemented, rejecting all different input specifications. The OWASP Enterprise Security API (OWASP, 2015j) has an extensible library of white list input validation routines;
- When performing an input validation, all relevant properties should be considered, including length, type of input, syntax, acceptable values, consistency and the accordance with the application rules;
- Define permission levels to the system, preventing unauthorized users from accessing privileged resources;
- Use mechanisms to automate data and code separation. These mechanisms are able to provide encoding and validation for inputs automatically rather than relying on the developer to provide this capability;

- Process SQL queries using ready statements, parameterized queries or stored procedures. Do not build parameters dynamically and run sequences of queries within these characteristics;
- Run code/scripts using the lowest privileges that are possible to perform the necessary tasks;
- Common users should have the least possible privileges, always following the principle of least privilege;
- Safety checks are normally applied on the client side, but it is necessary that these checks are also performed on the server side. Attackers can bypass the client side checks by modifying values after the validation or changing the client to remove the checks. So, it is important a redundant check implemented on the server side;
- The implementation of an application firewall can help detect some attacks against this vulnerability. However, an application firewall can not cover all possible input vectors.

3.2. Broken Authentication and Session Management Preventions

The primary recommendation is to make available to developers a single set of strong authentication and session management controls, such as the OWASP Application Security Verification Standard (OWASP, 2015e) (sections Authentication and Session Management), and have a simple interface for developers, a good example is the OWASP Enterprise Security API (OWASP, 2015j). Some common preventions are:

- Make sure that the sessions are invalidated when the user logs off;
- Check if the sessions have timeout, invalidating them after their timeout;
- All pages that require authentication must have a log off link;
- Session tokens should be long and random enough to be resistant to possible attacks;
- Only session IDs generated by the application should be recognized as valid by the application;
- Make sure that all components that the application depends are defined in terms of the security functions they provide.

3.3. Cross-Site Scripting (XSS) Preventions

XSS preventions requires separation of untrusted data from active browser content. The techniques to preventing XSS attacks are very similar to the ones used to prevent the injection attacks. However, other strategies also should be followed:

- Escape all untrusted data based on the HTML context (body, attribute, JavaScript, CSS and URL) that the data will be placed into. The OWASP XSS Prevention Cheat Sheet (OWASP, 2015n) shows details about data escaping techniques;
- The use of libraries and frameworks that make output management easier. Some libraries that can be used for this purpose are the OWASP Enterprise Security API (OWASP, 2015j), Microsoft Anti-Cross Site Scripting Library (MICROSOFT, 2015) and Apache Wicket (APACHE, 2015);
- Specify an output encoding, like ISO-8859-1 or UTF-8. Specifying an output encoding prevents the choice of a different one. When the encodings are inconsistent, the application can handle a sequence of bytes as special, even they not being;

- To prevent XSS attacks against the session cookies, set them with *HttpOnly*. The *HttpOnly* option does not allow the cookie to be accessed by client side scripting. So, if the user accesses a link that exploits this flaw, the browser will not reveal the cookie to a third part;
- When the acceptable set of objects is known, such as the file names or URLs, it is recommended to make a list with these objects, rejecting any other unknown entry;
- For rich content, it is recommended the use of an auto-sanitization library, like the OWASP AntiSamy Project (OWASP, 2015d);
- Also it is recommended a Content Security Policy (CSP) (OWASP, 2015b) to prevent XSS flaws across the entire application.

3.4. Insecure Direct Object References Preventions

The best way to prevent this kind of vulnerability is avoiding direct object references, some approaches must be followed to protect each accessible object:

- Use object indirect references to users and sessions, preventing directly access to unauthorized resources. The OWASP Enterprise Security API (OWASP, 2015j) contains access reference maps which developers can use to eliminate direct object references;
- The use of direct references by untrusted users must include an access control to verify if they are authorized to access the requested objects;
- The use of cryptography to make harder for an attacker to guess the legitimate values;
- The association of a digital signature, enabling the server to catch objects access violation;
- Error messages should contain only useful details. Very detailed information can be used to refine the original attack and increase the chances of access;
- Also run the code using the least privilege required for each task and ensure that the security checks performed on the client side are also performed on the server side.

3.5. Security Misconfiguration Preventions

Developers and administrators must work together to ensure that all application levels are configured properly. Automated scanners can be useful to detect security failures, configuration errors, use of default accounts, unnecessary services etc. It should be highlighted some practices that can leave the application more secure:

- All components must be kept up to date, including the Operating System, Web/App Servers, Database Management Systems, Applications and External Libraries;
- Unused resources should be disabled or uninstalled (ports, services, pages, accounts, privileges etc);
- Default accounts with default passwords should be disabled;
- Error messages should not contain excessive information or track the application stacks;

- Configuration management tests can be used to perform an analysis of infrastructure and topology architecture. They can reveal a lot about an application, such as information about the source code, HTTP methods allowed, administrative functionalities, authentication methods, and infrastructure configurations;
- SSL/TLS validations. SSL and TLS are protocols that provide, through encryption, secure channels for confidentiality and authentication of the information transmitted. It is important to check if the encryption algorithm is up to date and the correct use of it;
- A repeatable and automated process that makes fast and easy to deploy the environments (development, test and production) that is properly locked down. All the environments should be configured identically (with different passwords). This is important to minimize the effort required to setup a new secure environment;
- A well defined process for deploying all new software updates and patches in a timely manner to each deployed environment;
- A security policy that requires scans and audits periodically to detect future misconfigurations or missing patches.

3.6. Sensitive Data Exposure Preventions

The first thing to determine is which data is sensitive enough to require extra protection. Some examples are passwords, credit card numbers, health records and personal information, like address, phone numbers, document numbers etc. It is extremely important that all sensitive data, stopped or in transit, are always encrypted. This kind of data can never be stored os transmitted as clear text. Other prevention mechanisms are:

- Do not store sensitive data unnecessarily. Discards them as soon as possible avoiding that they can be stolen;
- Always make sure to use keys and strong cryptography algorithms. Proper encryption key management is very important. Consider modules validated by the Cryptographic Module Validation Program (CMVP) (NIST, 2015a). Modules validated by them are accepted by Federal Agencies of U.S and Canada for the protection of sensitive information;
- Passwords should be stored with encryption algorithms such as *bcrypt*, *PBKDF2* or *scrypt*, which are specially designed algorithms for storing passwords;
- Turn off auto complete feature in forms that contain sensitive data and also disable *caching* for pages that contain sensitive data;
- More detailed information can be found at the OWASP Application Security Verification Standard (OWASP, 2015e) (sections Cryptography, Data Protection and Communications Security).

3.7. Missing Function Level Access Control Preventions

To prevent this kind of vulnerabilities, the application must implement a consistent authorization module that is called in all business functions. Such protection is provided by one or more external components. The enforcement mechanisms should:

- Deny all access by default;
- Require explicit grants to specific roles;
- If a called function is involved in a workflow, make sure the conditions are valid to allow access;

- Generally, web applications do not display links to unauthorized functions, but this kind of presentation layer access control does not provide protection enough. Proper checks also need to be implemented in the controller and/or business layer.

3.8. Cross-Site Request Forgery (CSRF) Preventions

To check if an application is vulnerable to CSRF it is important to verify if its links and forms have unpredictable CSRF tokens. Without these tokens, attackers can forge malicious requests. CSRF preventions require the inclusion of an unpredictable token in each HTTP request, such tokens should be unique per user session. An alternative of defense is to require the real intention to send a request through authentication or proving that is a real user through a CAPTCHA.

The OWASP (OWASP, 2015c) has created the OWASP *CSRFTester* Project (OWASP, 2015i), it aims to give developers the ability to test their applications regarding to CSRF flaws. Other OWASP project to prevent CSRF risks is the OWASP *CSRFGuard* (Java EE, .NET and PHP) (OWASP, 2015h), a library that implements a variation of the synchronizer token pattern to reduce the risk of CRSF attacks.

In addition to using tools such as OWASP *CSRFTester* and OWASP *CSRFGuard*, other measures must be taken to prevent CSRF attacks:

- The use of some library for proper management session, once it helps preventing this type of attack;
- Make sure the application does not have XSS flaws, since most defenses against CSRF can be circumvented using scripts controlled by the attacker;
- Identify dangerous operations. When the users perform this type of operations a separate confirmation should be sent to ensure that the user really want to perform the operation.

3.9. Components with Known Vulnerabilities Preventions

In theory, this vulnerability should be detected easily, but unfortunately the scanners softwares do not always specify the exact versions of the components. What makes more difficult to found this vulnerabilities is the fact that there is no a central where these vulnerabilities are reported and detailed. Nowadays, sites like Common Vulnerabilities and Exposures (CVE) (CVE, 2015) and National Vulnerability Database (NVD) (NIST, 2015b) are making easier to search for these information, their databases contain detailed information about some software vulnerabilities.

If one of the components used in the application has a vulnerability, the application must be examined carefully to verify if the vulnerable part of this component is used in the code and if a failure could result in an impact to the user.

The OWASP Good Component Practices Project (OWASP, 2015k) aims to help with the best practices when using components within applications. The project covers the use of components during the whole development cycle, from the moment that the component is selected to its implementation and maintenance within the project.

The best protection mechanism is do not use components created by third parties. But, unfortunately, this is not always possible. From the moment that third party components are being used, some extra cares must be taken:

- Identify all the components used and their versions, including all dependencies;
- Upgrades to the components newest versions is critical;
- Being informed about the safety of these components in collective databases and security discussions forums;
- Establish security policies that monitor the use of third-party components, such as requiring certain software development practices, safety tests, acceptable licenses and constant updates;
- Consider, when necessary, the inclusion of security features around the components to disable unused functions and improve safety features considered weaks.

3.10. Unvalidated Redirects and Forwards Preventions

Redirects and forwards within the application should be avoided whenever possible. If the use can not be avoided, same cares must be taken:

- Redirects and forwards must not contain parameters that allow the user to determine the target;
- If the target parameter can not be avoided, ensure that the value provided is valid and authorized for the application. It is recommended to require target parameters as mapping values instead of URLs, where the local server can translate this mappings to the targets URLs;
- Use input validation strategies, as a white list of acceptable inputs, rejecting any entry that does not conform to the specifications;
- Using an intermediate page that provides the user a clear warning that he/she is leaving the current site. Impose a long period before the user be redirected to another page, or require the user to click on a redirect link;
- When the set of URLs is limited and known, create a mapping to a set of fixed values and reject all other inputs;
- Knowing all entries that are potentially unreliable: parameters or arguments, cookies, any data that is read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail addresses, external databases and systems that provide data to the application;
- The use of an application firewall can be useful to detect attacks against this weakness. This can be beneficial in cases where the application code can not be fixed;
- The use of libraries like the OWASP Enterprise Security API (OWASP, 2015j).

4. Open Source Tools for Finding Vulnerabilities

The goal when developing web applications is to be able to deploy them working correctly even when under an attack. To achieve this goal it is necessary: (1) identify and understand the common vulnerabilities, (2) design and implement these applications trying to avoid the common vulnerabilities, (3) understand the security implications regarding to the client side technologies (like JavaScript), (4) detect security vulnerabilities using appropriate tools and (5) deploy and configure them, and their environments, properly.

All these topics where discussed before, except the use of appropriate tools for detecting security vulnerabilities. There are many web security tools available, such as web testing proxies, vulnerability scanners and web application firewalls. Web proxies and vulnerability scanners are mostly used to evaluate the security of an application, while web application firewalls are used to block malicious input. There are commercial and open source tools in all these three categories.

The use of web proxies and vulnerability scanners are essential to develop and deploy secure web application. Web proxies are used for security assessment of a web application. They run on the same machine as the web browser and offer the ability to edit web inputs, such as form parameters, cookies and HTTP headers. Vulnerability scanners allow to test a large amount of web pages in a short period of time. They scan a web application submitting a series of testing strings to each discovered input and the responses are analyzed against common vulnerabilities. While scanners can detect simple cases of some web vulnerabilities, they can not detect most access control problems or application logic flaws.

In 2010, Shay Chen has created the Web Application Vulnerability Scanner Evaluation Project (WAVSEP) (WAVSEP, 2015), a vulnerable web application designed to help assessing the features, quality and accuracy of web application vulnerability scanners. Some benchmarks performed using this platform were performed (SECTOOLMARKET, 2015), these benchmarks has evaluated a number of characteristics of web vulnerability scanners, such as the detection accuracy (for different vulnerabilities), the features support (e.g. authentication and usability) and adaptability and coverage (e.g the input vector support). Based on these benchmarks we have used the following three open sources web vulnerability scanners to scan a simple web application previously developed: (1) Skipfish (SKIPFISH, 2015), (2) Zed Attack Proxy (ZAP) (OWASP, 2015o) and (3) Iron Web Application Advanced Security Testing Platform (IronWASP) (IRONWASP, 2015).

The Skipfish tool is sponsored by Google and was developed by Michal Zalewski, Niels Heinen and Sebastian Roschke. It is an active web application security reconnaissance tool. Its key features are: high speed (highly optimized HTTP handling, achieving 2000 requests per second), ease to use (supporting a variety of web frameworks and mixed technology sites) and cutting-edge security logic (high quality, low false positives and differential security checks).

The ZAP tool was developed by a global team of volunteers and sponsored, directly or indirectly, by several organizations, such as OWASP, Mozilla, Google, Microsoft etc. It is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with large experience in security as well as for developers and testers who are new to penetration testing. Among its several technical features, the tool stands out for being open source, cross platform, easy to install and use, fully internationalized (already translated into over 20 languages), community based with an active development by an international team of volunteers. It also was elected in 2013 (by users) as the best security tool.

The IronWASP tool was developed by Lavakumar Kuppan. It is an open source system for web application vulnerability testing. It is designed to be customizable, where users can create their own custom security scanners. Though an advanced use of Python or Ruby scripting, experts would be able to make full use of the platform, several other features are simple enough to be used by developers and testers who are new in security. Its main features are: an easy to use interface (no security expertise required), a powerful and effective scanning engine, supports recording Login sequence, reporting in both

HTML and RTF formats, checks for over 25 different kinds of well known web vulnerabilities, false positives/negatives detection support, a built-in scripting engine that supports Python and Ruby, extensible via plugins or modules in Python, Ruby, C# or VB.NET and a growing number of modules built by researchers in the security community.

For our tests we have used a previously developed web application. In fact, this web application was a simple CRUD (create, read, update and delete functions) developed without the base knowledge of the web application security best practices. Our activities sequence were:

- An scanning over the original application with each tool;
- An individual analises over the scanning results;
- Based on the vulnerabilities found, we have corrected the original application and its environment;
- Finally, we have scanned the changed application to verify if all vulnerabilities were treated.

5. Scanning results

We have analyzed the scanning results according to the following aspects:

- The amount of time spend to application scan;
- The amount and witch vulnerabilities was found;
- The interpretability of the results.

The scanning results regarding to the original application version are shown in Table 1.

Table 1. Execution time and errors handled by each tool.

	IronWasp	ZAP	Skipfish
Execution time (minutes)	37	30	45
Injection	11	-	1
XSS	19	3	-
XSRF	-	-	3
Cookies	1	1	-
Incorrect or missing charset	-	-	18
Error 404	-	-	4
Header errors	-	26	-
Total	32	30	26

Despite finding the largest variety of errors, the Skipfish tool did not specify clearly what kind of vulnerability they can lead. Incorrect or missing charsets can lead to XSS/XSRF flaws, but this was not noticed by the tool. On the other hand, it was the only tool that has reported XSRF flaws. It took the biggest execution time, substantially larger than the other tools. It has a complicated layout, making difficult to understand its error reports.

The ZAP tool did not find a significant variety of errors. The main kind of errors found (header errors) does not mean so much by themselves. It is known that these kind of errors also can lead to XSS/XSRF flaws, which is not noticed by the tool. Despite not presenting many results, they were easy to analyze, showing details of the pages and blocks of code where the errors were found.

With the IronWASP tool, the scanning time was intermediate, as well as the number and variety of errors. Furthermore, the tool has detected the greatest number of high risk errors, making clear what vulnerability they were. The tool has a simple layout and a simplified analysis of results, showing the blocks of code where the errors were found. Based on our tests, this tool has presented the best relation between efficiency and simplicity of use.

6. Conclusions

Security in web applications is a very complex area, it covers from the user and its browser to the extensiveness and vulnerability of the Internet. The lack of security in web applications can not be justified as a consequence of the ease of access and heterogeneity of the Internet. As showed in this paper, there are several organizations and communities working to help developers and companies with lectures, documentation, resources and tools, bringing all the support and mechanisms for maintaining the web applications security. The large number of vulnerabilities found in web applications is a result of the lack of knowledge of basic safety concepts. Either developers and companies do not care to write secure code, until they have a serious problem regarding to security.

It is essential design and develop web applications focused on security. This mean, web applications that perform validations entries, error handling, limited access, use of security APIs, correct use of the technology, among other methods described in this study. Finally, it is important to use a set of tools to test the security of applications. As can be seen in our tests and also in (SECTOOLMARKET, 2015), the use of more than one tool can provide greater safety benefits such as a greater number of identified errors and a greater diversity of them, helping us to find possible security flaws in our web applications.

References

APACHE. *Apache Wicket*. 2015. ⟨https://wicket.apache.org⟩.

CIS. *Center for Internet Security*. 2015. ⟨https://www.cisecurity.org⟩.

CVE. *Common Vulnerabilities and Exposures*. 2015. ⟨http://cve.mitre.org/about/index.html⟩.

CWE. *Common Weakness Enumeration*. 2015. ⟨http://cwe.mitre.org⟩.

CWE/SANS. *CWE/SANS TOP 25 Most Dangerous Software Errors*. 2015. ⟨http://www.sans.org/top25-software-errors⟩.

DOUPÉ, A. et al. *Enemy of the State: A State-aware Black-box Web Vulnerability Scanner*. [S.l.]: Proceedings of the 21st USENIX Conference on Security Symposium, 2012.

IRONWASP. *Iron Web application Advanced Security testing Platform*. 2015. ⟨https://ironwasp.org⟩.

ISO/IEC. *Information technology security techniques management of information and communications technology security part 1: concepts and models for information and communications technology security management*. [S.l.]: SO/IEC, 2004.

ISO/IEC. *Code of practice for information security management*. [S.l.]: ISO/IEC, 2005.

KALS, S. et al. *SecuBat: A Web Vulnerability Scanner*. [S.l.]: Proceedings of the 15th International Conference on World Wide Web, 2006.

MELLADO, D. et al. *A systematic review of security requirements engineering*. [S.l.]: Elsevier / Computer Standards & Interfaces, 2010.

MICROSOFT. *Microsoft Anti-Cross Site Scripting Library*. 2015. ⟨http://msdn.microsoft.com/pt-br/security/aa973814.aspx⟩.

MITRE. *MITRE Corporation*. 2016. ⟨http://www.mitre.org⟩.

NIST. *Cryptographic Module Validation Program*. 2015. ⟨http://csrc.nist.gov/groups/STM/cmvp/index.html⟩.

NIST. *National Vulnerability Database*. 2015. ⟨https://nvd.nist.gov/home.cfm⟩.

O'NEILL, M. *The internet of things: do more devices mean more risks?* [S.l.]: Elsevier / Computer Fraud & Security, 2014.

OWASP. *OWASP Top 10 Application Security Risks*. 2013. ⟨https://www.owasp.org/index.php/Top10⟩.

OWASP. *Application Security Verification Standard Project*. 2015. ⟨https://www.owasp.org/index.php/OWASP_Application_Security_Verification_Standard_Project⟩.

OWASP. *Content Security Policy*. 2015. ⟨https://www.owasp.org/index.php/Content_Security_Policy⟩.

OWASP. *Open Web Application Security Project*. 2015. ⟨https://www.owasp.org⟩.

OWASP. *OWASP AntiSamy Project*. 2015. ⟨https://www.owasp.org/index.php/AntiSamy⟩.

OWASP. *OWASP Application Security Verification Standard Project*. 2015. ⟨https://www.owasp.org/index.php/ASVS⟩.

OWASP. *OWASP Cheat Sheet Series*. 2015. ⟨https://www.owasp.org/index.php/Cheat_Sheets⟩.

OWASP. *OWASP Code Review Project*. 2015. ⟨https://www.owasp.org/index.php/OWASP_Code_Review_Project⟩.

OWASP. *OWASP CSRFGuard*. 2015. ⟨https://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project⟩.

OWASP. *OWASP CSRFTester Project*. 2015. ⟨https://www.owasp.org/index.php/Category:OWASP_CSRFTester_Project⟩.

OWASP. *OWASP Enterprise Security API*. 2015. ⟨https://www.owasp.org/index.php/Categoryx:OWASP_Enterprise_Security_API⟩.

OWASP. *OWASP Good Component Practices Project*. 2015. ⟨https://www.owasp.org/index.php/OWASP_Good_Component_Practices_Project⟩.

OWASP. *OWASP Guide Project*. 2015. ⟨https://www.owasp.org/index.php/OWASP_Guide_Project⟩.

OWASP. *OWASP Testing Project*. 2015. ⟨https://www.owasp.org/index.php/OWASP_Testing_Project⟩.

OWASP. *XSS (Cross Site Scripting) Prevention Cheat Sheet*. 2015. ⟨https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet⟩.

OWASP. *Zed Attack Proxy Project*. 2015. ⟨https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project⟩.

SAFECODE. *SAFECode Publications*. 2015. ⟨http://www.safecode.org/publications⟩.

SAFECODE. *Software Assurance Forum for Excellence in Code*. 2015. ⟨http://www.safecode.org⟩.

SANS. *SysAdmin, Audit, Network, Security Institute*. 2015. ⟨http://www.sans.org/⟩.

SECTOOLMARKET. *A Dynamic Security Benchmark Presentation Platform*. 2015. ⟨http://www.sectoolmarket.com⟩.

SHAHRIAR, H. *Security Vulnerabilities and Mitigation Techniques of Web Applications*. [S.l.]: Proceedings of the 6th International Conference on Security of Information and Networks, 2013.

SHEMA, M. *Web Application Security For Dummies*. [S.l.]: John Wiley & Sons Ltd, 2011.

SKIPFISH. *Skipfish - Web Application Security Scanner*. 2015. ⟨https://code.google.com/p/skipfish⟩.

SOLMS, R. V.; NIEKERK, J. V. *From information security to cyber security*. [S.l.]: Elsevier / Computer Fraud & Security, 2013.

THOMÉ, J.; GORLA, A.; ZELLER, A. *Search-based Security Testing of Web Applications*. [S.l.]: Proceedings of the 7th International Workshop on Search-Based Software Testing, 2014.

WALDEN, J. *Integrating Web Application Security into the IT Curriculum*. [S.l.]: Integrating Web Application Security into the IT Curriculum, 2008.

WASC. *Web Application Security Consortium*. 2015. ⟨http://www.webappsec.org⟩.

WAVSEP. *The Web Application Vulnerability Scanner Evaluation Project*. 2015. ⟨https://code.google.com/p/wavsep⟩.

WHITEHATSECURITY. *2014 Website Security Statistics Report*. [S.l.]: Website Security Statistics Report, 2014.