Pattern based Web Security Testing

Paulo J. M. Araújo[1] and Ana C. R. Paiva[1,2]
[1]*Faculty of Engineering of the University of Porto, Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal*
[2]*INESC TEC, Rua Dr. Roberto Frias, s/n 4200-465 Porto, Portugal*

Keywords: Security Testing, Pattern based Testing, Pattern based Security Testing, Security Web Testing.

Abstract: This paper presents a Pattern Based Testing approach for testing security aspects of the applications under test (AUT). It describes the two security patterns which are the focus of this work ("Account Lockout" and "Authentication Enforcer") and the test strategies implemented to check if the applications are vulnerable or not regarding these patterns. The PBST (Pattern Based Security Testing) overall approach has two phases: exploration (to identify the web pages of the application under test) and testing (to execute the test strategies developed in order to detect vulnerabilities). An experiment is presented to validate the approach over five public web applications. The goal is to assess the behavior of the tool when varying the upper limit of pages to visit and assess its capacity to find real vulnerabilities. The results are promising. Indeed, it was possible to check that the vulnerabilities detected corresponded to real security problems.

1 INTRODUCTION

To have quality today, a software application should work flawlessly and be safe. However, exposure of applications to undesirable attacks is common and brings new challenges. Developers are concerned mainly to create applications that work properly and sometimes they neglect the security of the application. That is why testing the developed application is critical to ensure the quality and reliability of the product.

However, the test implementation requires time and money which are limited resources. In recent years, a number of testing tools based on different methodologies have emerged to try making the testing process faster and more systematic. One of these methodologies is testing based on models/patterns which have been increasingly accepted (Utting and Legeard, 2007).

In a previous project called PBGT (Pattern Based GUI Testing) [1] we developed several tools to test software applications through their Graphical User Interface (GUI) (Moreira et al., 2013), (Moreira et al., 2017), (Paiva and Vilela, 2017). The main goal of PBGT is the develop generic test strategies to test common recurrent behavior that can be applied over different applications after a configuration step. This is a black box testing approach with no access to the source code of the applications under test. The test cases are built from a model describing the testing goals and afterwards the testing process is automated.

The research work presented in this paper extends the concepts developed within the PBGT project for testing the security aspects of the web applications, i.e., it develops test strategies to test known security patterns.

A survey about security issues and security patterns served as the basis for the development of generic test strategies to test security patterns on different web applications. This work presents the test strategies defined for two security patterns: "Authentication Enforcer" and "Account Lockout".

The remainder of this document covers state of the art about security patterns in Section 2, presents the contributions of the paper in Section 3, the validation of PBST (Pattern Based Security Testing) through case study is in Section 4 and finally some conclusions in Section 5.

[1]https://www.fe.up.pt/˜apaiva/pbgtwiki/doku.php

2 SECURITY PATTERS

Integrating security requirements at different stages of the software development cycle is increasingly a necessity and is a concern of both programmers and stakeholders involved in the software development process. However, there are still differences in how security engineers and software engineers design the

Araújo, P. and Paiva, A.
Pattern based Web Security Testing.
DOI: 10.5220/0006606504720479
In *Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018)*, pages 472-479
ISBN: 978-989-758-283-7

security requirements for a particular application.

There have been several methodologies that seek to streamline and normalize the way to develop applications that require demanding quality requirements. One of the most recent methodologies and one that has an increasing number of adepts is the development based on patterns.

Christopher Alexander (Alexander et al., 1977) first defined patterns in architecture as a representation of the "current best guess as to what arrangement of the physical environment will work to solve the problem presented". Generalizing this concept, one can assert that a pattern is a recurring solution for a recurring problem.

When considering security aspects (Heyman et al., 2007a), security patterns can be defined as reusable solutions for security problems.

As Thomas et al. (Heyman et al., 2007b) said, it is more reliable to use the security features already known and tested than to invent ad-hoc solutions from scratch. From the article "Measuring the level of security introduced by security standards" (Fernández et al., 2010) we can also conclude that the use of standards brings significant improvements in the final security of the developed system.

Different authors present different ways of classifying security patterns according to their points of view, and a good classification of a pattern facilitates the correct selection later.

Security patterns, according to Eduardo et al. (Fernandez et al., 2008), can be grouped into the following categories: identification and authentication, access control and authorization, registration, encryption and intrusion detection.

J. Yoder and J. Barcalow (Yoder and Barcalow, 1998) present a collection of seven security patterns: single access point; check point; roles; session; full view with errors; limited view; secure access layer.

Darrel et al. (Kienzle et al., 2006) show a repository of 26 patterns and 3 mini patterns, that are organized in two groups: structural patterns, and procedural patterns.

Other authors present collections of patterns grouped/classified in different forms (Anand et al., 2014), (Heyman et al., 2007a), (Slavin et al., 2012).

As Ina Schieferdecker et al (Schieferdecker et al., 2012) state, security testing is intended to validate that in an application the following security properties are guaranteed: confidentiality (guarantees the secrecy of data), integrity (guarantees that data is not modified in an unauthorized way), authenticity (be sure about the identity of a person), availability (means that data/services are available) and non-repudiation (ensure that a transferred message has been sent and received by the parties claiming to have sent and received the message).

It is therefore necessary to test the application using white-box tests to discover code faults written by the developers and also to use black-box tests, that is, to try to act as an attacker by attempting to penetrate the system and observing how the Application behaves in the face of such attempts to intrude and/or tamper with the data.

The research work described in this paper is based on the work developed by the authors Munawar Haifz et al. (Adamczyk et al., 2007), (Hafiz et al., 2012) which present the patterns organized by the STRIDE threat model (developed by Microsoft). STRIDE means Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privileges.

- Spoofing is an attempt to gain access to any application or system using a fake identity in which a vulnerable system will give unauthorized access to sensitive data.
- Tampering consists of tampering with data during communication where data integrity is compromised.
- Repudiation arises when a user refuses to acknowledge that he or she is the participant of a particular transaction.
- Information Disclosure arises when confidential data is exposed or lost unintentionally.
- Denial of Service is to cause failures in system availability.
- Elevation of Privileges arises when a user is able to exploit a vulnerability and access data and resources for which the same user does not have access privileges.

For each of these aspects, the authors define a set of security patterns. As an example, for Spoofing we have: Account Lockout; Assertion Builder; Authentications Enforcer; Brokered Authentication; Credential Tokenizer; Intercepting Web Agent; Message Replay Detection, and Network Address Blacklist.

As already mentioned, many of the patterns are common among several approaches of different authors. In the web page that is maintained by one of the authors (SPC, 2017), one can obtain more details about each one of the patterns, the relationships among them, as well as the totality of the patterns proposed by them.

However, most of these patterns are frequently disregarded in software development.

In recent years there has been an increase in the number of security testing tools available, with tools focused on only one or a few vulnerabilities, and others that seek to cover as many known vulnerabilities as possible. Some examples are Acunetix (Acunetix, 2017), BeEF (BeEF, 2017), Burp Suite (portswigger, 2017), Iron Wasp (ironwasp, 2017), NetSparker (netsparker, 2017), SQLMap (sqlmap, 2017), Sqlninja (sqlninja, 2017), Vega (Vega, 2017), W3af (w3af, 2017), Wapiti (wapiti, 2017), ZAP (ZAP, 2017).

However, as far as we know, none of them is organized by security patterns. That is why developing a generic testing approach that is able to detect if the security patterns were taken into account during development and are well implemented may be useful.

3 PATTERN BASED SECURITY TESTING

The purpose of this work is to test security patterns following a black-box testing approach. The goal is to define and implement generic test strategies that are able to detect if security patterns were taken into account during the development and, ultimately, contribute for better software.

The security patterns that the PBST (Pattern Based Security Testing) tool developed in this work is able to test now are “Account Lockout” and “Authentication Enforcer” as described in (Adamczyk et al., 2007), (Hafiz et al., 2012).

Account Lockout is intended to protect accounts from automatic attacks based on the “guessing” of passwords. When implementing a limit number of attempts followed with wrong passwords for the same user prevents or at least hampers the task of hackers. To protect the systems against this attack one can lock the account after *N* incorrect attempts.

Authentication Enforcer is responsible for authenticating and verifying user identity. One way to prevent impostors from accessing the system is by creating a single point of access where all requests to enter the system are checked and apply an authentication protocol to verify the identity of the agent. On successful authentication, create a proof of identity of the agent (Hafiz et al., 2012).

In order to check if these security patterns were well implemented, a test strategy was defined and will be explained in more detail in subsection “Tester”.

The goal is to identify possible vulnerabilities that report that a security pattern is not well implemented. So, we started by grouping known vulnerabilities around security patterns and afterwards defined a set of test strategies to detect those vulnerabilities.

3.1 Tool

The PBST testing tool has four main components: Graphical User Interface, Explorer, Tester, Report generator.

3.1.1 Graphical User Interface

The Graphical User Interface (GUI) establishes the interaction with the testers / users. The main menu of the tool is shown in Figure 1.

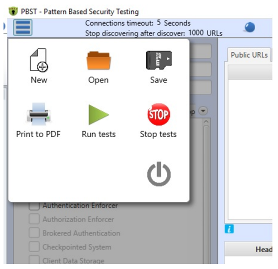

PBST - Pattern Based Security Testing
Connections timeout: 5 Seconds
Stop discovering after discover: 1000 URLs

Figure 1: Menu of the PBST tool.

3.1.2 Explorer

The PBST tool starts by exploring the web application under test trying to find all its web pages.

For each discovered page it is necessary to collect all the useful information available. Within this information there is, for example, the information contained in the meta, script, link and form tags. In addition to the tags, we can also read comments that may exist in the source code of the page in search of possible keywords forgotten by the programmers such as: query, password and user. Other information to collect is the headers sent in the server response and the cookies. The implementation needed to obtain the cookies is not implemented in the current state of the tool.

For each URL that is discovered by the explorer process one or two accesses are made to the server, first without authentication and, in case of failure, another one using the credentials provided to authenticate. This procedure besides allowing to obtain information of both public accessed pages also allows to obtain information of pages of restricted access and to catalog the pages like being of public access or of private access.

Several threads were used to make this tracking process as efficient as possible. In order to not overload the system, only with this application, the number of threads thrown take into account the number of processors, with as many threads as possible being created in the processors on the computer where the application is running.

The search is performed using BFS (Breadth-First Search) width lookup. Departing from the provided URL (seed / root), all URLs achievable from it are placed in a FIFO (First In, First Out) queue. For each new URL of the web application found, we search all URLs achievable from it and so forth. All the new URLs found are placed at the end of the queue (only if not yet explored). To know which URLs were already explored (ensure that the same URL is not queued twice), we maintain an auxiliary control list with all already explored URLs. The search ends when the queue is empty (when there are no more URLs to explore).

In addition, each analyzed URL is added to a list to be tested by the attack and analysis (tester) component. Two URLs are equal if they have the same base URL, i.e., if they differ only in the arguments they are considered the same.

The upper limit of URLs explored is defined by the tester.

3.1.3 Tester

After exploration, each page selected for testing is tested according to the security patterns selected by the tester. After the test is finished, information about the success or failure of the test / attack is saved.

Although the application tests security patterns, in reality what is done is to check whether or not the application has a particular vulnerability related to a security pattern. These vulnerability tests are grouped around security patterns in order to ensure that a particular security pattern is present and whether or not vulnerabilities related to that pattern are present. If no vulnerabilities are detected in any of the tests selected to test the security pattern in question then it is possible to say that it is not vulnerable (at least within the vulnerabilities known at the moment and tested).

For the ”Account Lockout” standard only a test is necessary. The goal is to try invalid login (username and password) several times to check if the access is blocked. This test strategy is detailed in Section “Experiment”.

To validate the correct implementation of the “Authentication Enforcer” security pattern, several tests are necessary but only four [2] are developed related to the following vulnerabilities: Clickjacking; Transport Credentials; SQL Injection; Default Credentials. We explain these vulnerabilities in more detail in the sequel.

- Clickjacking – This is when an attacker uses multiple transparent or opaque layers to fool a user. So the attacker is to ”hijack” clicks to your page and forward them to another page, probably belonging to another application, domain or both.
- Transport credentials – Testing the transport of credentials means verifying that the user’s authentication data is transferred through an encrypted channel to avoid being intercepted by malicious users. The analysis simply focuses on trying to understand whether the data travels unencrypted from the web browser to the server, or if the Web application takes appropriate security measures using a protocol such as HTTPS.
- SQL injection – It consists of the insertion or “injection” of a partial or complete SQL query through data entry or transmitted from the client (browser) to the web application.
- Default credentials – Nowadays, it is common for software applications to use popular open source or commercial software that can be installed on servers with minimal configuration or customization by the server administrator. In addition, many hardware devices (for example, network routers and database servers) offer web-based configuration or administrative interfaces. Often, this reused software/applications, once installed, is not configured correctly and the default credentials provided for initial authentication and configuration are never changed These default credentials are well known to penetration testers and, unfortunately, also by malicious attackers who can use them to gain access to various types of applications.

[2] www.owasp.org

To identify the presence of these vulnerabilities, we developed generic test strategies for each of the them. When one of these vulnerabilities is detected by the tests, it is possible to say that the application under test has an incorrect implementation of the “Authentication Enforcer” pattern.

Clickjacking

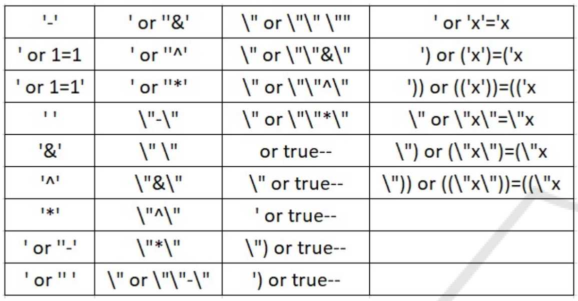
To test the clickjacking vulnerability, the headers of the AUT are scanned to see if the “x-frame-options” option is present. If this header is not present the application is potentially vulnerable to clickjacking unless other mechanisms are implemented that prevent it from being able to place the application inside a frame of another application (malicious application).

Transport Credentials

In order for an application not to be vulnerable while sending the access credentials to the server, they must be sent using the POST method and the transport channel must be secure, so the *HTTPS* protocol must be used. In this way, to validate whether or not an application has this vulnerability, an analysis is made of the authentication form in order to check which protocol is used to send the credentials to the server.

SQL Injection and Default Credentials

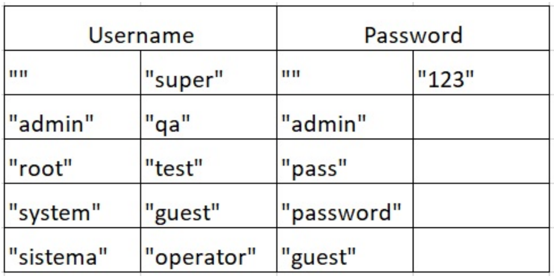
The security tests of SQL Injection are tests that attempt to log in via SQL sent, as access attempts, in the username and password. For this test, we used the list of possible strings to input as username and password shown in Figure 2.



'-'	' or "&'	\" or \"\" \""	' or 'x'='x
' or 1=1	' or "^'	\" or \"\"&\"	') or ('x')=('x
' or 1=1'	' or "*'	\" or \"\"^\"	')) or (('x'))=(('x
' '	\"-\"	\" or \"\"*\"	\" or \"x\"=\"x
'&'	\" \"	or true--	\") or (\"x\")=(\"x
'^'	\"&\"	\" or true--	\")) or ((\"x\"))=((\"x
'*'	\"^\"	' or true--	
' or "-'	\"*\"	\") or true--	
' or " '	\" or \"\"-\"	') or true--	

Figure 2: Strings used in SQL Injection for username and password fields.

The security tests of Default Credentials attempt to log in via default words sent as access attempts in the username and password. Default words are used, for instance, in test environments and, several times, forgotten going to production which allows improper accesses to the software in production. To test if it is possible to log in with default credentials the list of strings shown in Figure 3 was used (the quotation marks are not part of the string):



Username		Password	
""	"super"	""	"123"
"admin"	"qa"	"admin"	
"root"	"test"	"pass"	
"system"	"guest"	"password"	
"sistema"	"operator"	"guest"	

Figure 3: Default values used in username and password fields.

The tool tests all the combinations of username and password shown in Figure 3 and also repetition of the same word, e.g., adminadmin, which are also common as default words.

Account Lockout

To test the "Account Lockout" pattern, it is only necessary to perform a test. This test consists of trying to enter a wrong login data three times (the same login with different wrong passwords). After three invalid attempts, a correct access with the correct credentials is done. If the access is successful then we already know that the application does not block after three invalid attempts. In this case, the test is repeated, but this time with five invalid attempts followed by the sixth attempt with the correct credentials. In this case, if the sixth access is successful then we can consider that the application has vulnerabilities in the pattern "Account Lockout", because the security pattern advises to block the application after three or five invalid attempts.

3.1.4 Report Generator

Figure 4 shows a report generated after a test performed to a website with only six pages. The report shows the URL of the AUT, the number of pages classified as public or private, the number of URLs selected for testing and tested, part of the list of security patterns "Not tested", the result of the test for the web pages tested and the details about the vulnerabilities detected for each of the tested patterns. In this Figure 4, it can be seen that for the TdinApp both tested patterns are identified as vulnerable and regarding the "Account Lockout" pattern, all vulnerabilities tested for this pattern were detected.

Finally, it is also possible to see that, right now, all security patterns except "Account Lockout" and "Authentication Enforcer" are not yet implemented and because of that "Not tested".

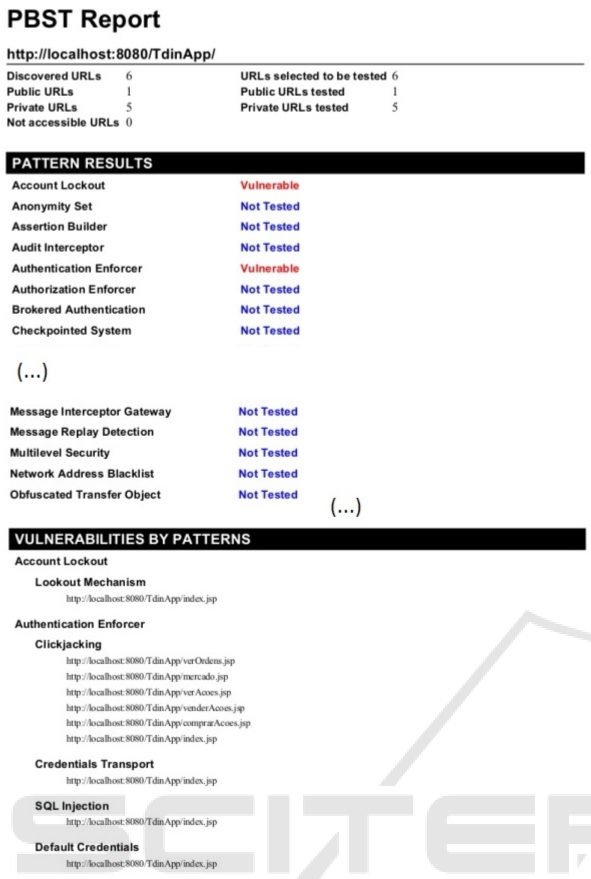
4 CASE STUDY

The goal of this case study is to validate the PBST testing approach, namely:

- Assess how the PBST tool behaves when varying the upper limit of web pages to explore;
- Assess if the PBST tool is able to detect vulnerabilities and if the ones detected are actually real vulnerabilities (true positives).

Current Restrictions of the PBST Tool

At this point in time, the application only tests web pages containing forms, without using JavaScript, and pages where the authentication mechanism is done only by the use of username and password. Because



PBST Report

http://localhost:8080/TdinApp/

Discovered URLs	6	URLs selected to be tested	6
Public URLs	1	Public URLs tested	1
Private URLs	5	Private URLs tested	5
Not accessible URLs	0		

PATTERN RESULTS

Account Lockout	Vulnerable
Anonymity Set	Not Tested
Assertion Builder	Not Tested
Audit Interceptor	Not Tested
Authentication Enforcer	Vulnerable
Authorization Enforcer	Not Tested
Brokered Authentication	Not Tested
Checkpointed System	Not Tested

(...)

Message Interceptor Gateway	Not Tested
Message Replay Detection	Not Tested
Multilevel Security	Not Tested
Network Address Blacklist	Not Tested
Obfuscated Transfer Object	Not Tested

(...)

VULNERABILITIES BY PATTERNS

Account Lockout
- Lookout Mechanism
  - http://localhost:8080/TdinApp/index.jsp

Authentication Enforcer
- Clickjacking
  - http://localhost:8080/TdinApp/verOrdens.jsp
  - http://localhost:8080/TdinApp/mercado.jsp
  - http://localhost:8080/TdinApp/verAcoes.jsp
  - http://localhost:8080/TdinApp/venderAcoes.jsp
  - http://localhost:8080/TdinApp/comprarAcoes.jsp
  - http://localhost:8080/TdinApp/index.jsp
- Credentials Transport
  - http://localhost:8080/TdinApp/index.jsp
- SQL Injection
  - http://localhost:8080/TdinApp/index.jsp
- Default Credentials
  - http://localhost:8080/TdinApp/index.jsp

Figure 4: Report generated.

of these limitations, the explorer phase selects only the pages that meet these requirements for testing.

4.1 Subjects

In order to validate the testing approach developed, we selected five web applications public available:

- SIGARRA – is the information system of the University of Porto and was selected because it is a large application that presents the authentication form on many of its web pages.
- jigsawplanet – this application was selected after an Internet search for free applications and it was referenced on a page that advertises various applications.
- TdinApp – this application was created in a project of the discipline of Distribution and Integration Technologies. Since we had access to the source code, this application was useful to check if the results obtained by the PBST tool were altered if a vulnerability was removed. This application was only available on localhost.
- acunetix – this application was selected because it was created with the purpose of being used to validate test tools or for the training of security testing specialists (e.g., testasp). This application contain several security vulnerabilities. There is a blog where you can find this and other test pages.
- testfire – this application was selected because the same reasons as acunetix above.

4.2 Experiment

The experiment was divided in three phases:

- the first phase aimed to test the explorer component;
- the second phase aimed to test the explorer and the testing of the two patterns: "Authentication Enforcer" and "Account Lockout";
- the third phase aimed to test the overall process.

4.2.1 First Phase

The tests in this phase are intended to verify how the application behaves when varying the number of pages to test (the upper limit of pages to visit is a parameter defined by the tester). We used three of the subjects. During this experiment, we extracted metrics such as time and quantities of pages selected for testing (according to the actual restrictions of the tool already mentioned). The results are in Figure 5.

Upper limit of pages to visit	100	500	1000	5000	10000
sigarra	20	61	116	781	4948
testasp	18	44	78	353	763
jigsawplanet	21	52	98	444	885

Figure 5: Time taken in seconds.

Figure 6 shows the number of pages that were selected for testing within the search limit.

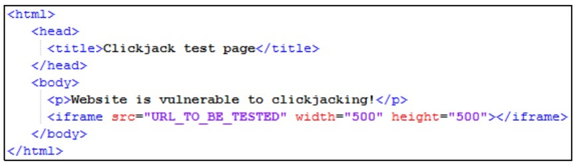
Upper limit of pages to visit	100	500	1000	5000	10000
sigarra	25	65	91	378	452
testasp	28	139	274	1366	2958
jigsawplanet	31	84	152	702	1021

Figure 6: Pages selected for test.

4.2.2 Second Phase

During this phase we tested the "Authentication Enforcer" and the following vulnerabilities: "clickjacking", "credentials transport", "SQL Injection" and "default credentials".

To verify that the application actually detects the "clickjacking" vulnerability, a script has been created



```
<html>
   <head>
     <title>Clickjack test page</title>
   </head>
   <body>
     <p>Website is vulnerable to clickjacking!</p>
     <iframe src="URL_TO_BE_TESTED" width="500" height="500"></iframe>
   </body>
</html>
```

Figure 7: Script to confirm clickjacking vulnerability.

to run manually and to make sure the result of the test was correct (Figure 7).

Within the group of pages marked, by the PBST application, as containing the “clickjacking” vulnerability, we selected some to verify if with the auxiliary script it was possible to place the application within a frame. All the pages checked by this process confirm that the result of the test was correct. Indeed, the pages marked as vulnerable were in fact vulnerable. We also checked if the result of the test was correct for the pages marked has not vulnerable. Indeed, for these pages it was not possible to place them inside a frame so, the result of the test was correct.

When the PBST detects a vulnerability regarding SQL injection it provides information about the string used that allowed the detection of the vulnerability. In case of the “testfire” such string was ’ or’ ”&’. In the case of the “Default Credentials” vulnerability, the application also provides the strings used for the username and for the password. In the case of “testfire” such strings were *admin* and *admin*.

Considering the whole set of applications tested, the only one that did not present any of the vulnerabilities of the pattern “Authentication Enforcer” was the application “jigsawplanet”. For all the others some vulnerabilities were detected. In particular, the application testfire had all the vulnerabilities associated with this test can be seen in Figure 4.

4.2.3 Third Phase

At this phase the exploration and testing of the two patterns were tested. This phase is necessary and complements the previous one because the lockout test can only be performed after all the rest is finished.

Regarding the application “jigsawplanet”, it is secure for “Authentication Enforcer” but vulnerable for “Account Lockout”.

In order to check if the “Account Lockout” pattern was well detected, we manipulated the application “TdinApp”. We obtained results according to the manipulation which confirms the correctness of the testing strategy.

5 CONCLUSIONS

This paper presented a new approach to test security aspects of web applications organized by security patterns. The patterns implemented right now are the “Authentication Enforcer’ (which checks vulnerabilities related to clickjacking, transport credentials, SQL Injection and Default Credentials) and “Account Lockout” (which checks if the application blocks after three or five invalid login attempts).

Grouping vulnerability testing into security patterns can be an asset to a test application in that it is thus possible to assert that at the time the test was performed and for known vulnerabilities at that time, the application is either “safe” / “well implemented” or “Not safe” / “Not well implemented” regarding a certain security pattern. This structure may help the programmers to identify which aspects they have to improve in order to fix the problems detected. Another advantage is that it may be a requirement for a particular application to have certain security patterns, but at the same time there may be others that are irrelevant, so that having the tests grouped by patterns allows an easily selection of the group of vulnerabilities that are necessary to test.

This paper explains how the PBST tool works and presents a case study to validate the overall approach. From the experiments performed, it is possible to state that PBST is able to identify vulnerabilities successfully. Of course, the application still needs a lot of development to extend the number of security patterns tested and be a really useful tool.

During the validation phase we tested the PBST tool over five applications and the results were promising. We detected vulnerabilities and confirmed that the vulnerabilities detected corresponded to real problems.

Excluding Sigarra application and according to the experiments, it is possible to say that the time spent with the exploration increases linearly with the upper limit of pages to visit. Of course, this time is also influenced by other factors such as the processing capacity of the computer, the Internet connection and the speed of response of the Server where the application is hosted.

As future work we aim to extend the set of security patterns to test because, at this point in time, some vulnerabilities may be undetected.

The exploration needs also improvements since we have restrictions regarding the web pages we are able to test right now which need to be overcome.

Right now the exploration stops when the tester presses a button for that or when the exploration reaches the upper limit of pages to visit. In the future it may the helpful to have time limit.

Finally, we aim to test the overall approach over more web applications.

REFERENCES

Acunetix (2017). Advanced penetration testing tools included. http://www.acunetix.com/vulnerability-scanner/penetration-testing/. [Accessed on 12/7/2017].

Adamczyk, P., Hafiz, M., and Johnson, R. E. (2007). Organizing security patterns. *IEEE Software*, 24:52–60.

Alexander, C. W., Ishikawa, S., Silverstein, M., and Jacobson, M. (1977). *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, New York, USA, 1 edition.

Anand, P., Ryoo, J., and Kazman, R. (2014). Vulnerability-based security pattern categorization in search of missing patterns. In *Proceedings of the 2014 Ninth International Conference on Availability, Reliability and Security*, ARES '14, pages 476–483, Washington, DC, USA. IEEE Computer Society.

BeEF (2017). The browser exploitation framework project. //beefproject.com/. [Accessed on 27/7/2017].

Fernandez, E. B., Washizaki, H., Yoshioka, N., Kubo, A., and Fukazawa, Y. (2008). *Classifying Security Patterns*, pages 342–347. Springer Berlin Heidelberg, Berlin, Heidelberg.

Fernández, E. B., Yoshioka, N., Washizaki, H., and Van-Hilst, M. (2010). Measuring the level of security introduced by security patterns. In *ARES 2010, Fifth International Conference on Availability, Reliability and Security, 15-18 February 2010, Krakow, Poland*, pages 565–568.

Hafiz, M., Adamczyk, P., and Johnson, R. E. (2012). Growing a pattern language (for security). In *Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2012, pages 139–158, New York, NY, USA. ACM.

Heyman, T., Yskout, K., Scandariato, R., and Joosen, W. (2007a). An analysis of the security patterns landscape. In *Third International Workshop on Software Engineering for Secure Systems, SESS 2007, Minneapolis, MN, USA, May 20-26, 2007*, page 3.

Heyman, T., Yskout, K., Scandariato, R., and Joosen, W. (2007b). An analysis of the security patterns landscape. In *Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, SESS '07, pages 3–, Washington, DC, USA. IEEE Computer Society.

ironwasp (2017). Iron web application advanced security testing platform. //ironwasp.org/. [Accessed on 27/7/2017].

Kienzle, D. M., Elder, M. C., D, P., D, P., Tyree, D., and Edwards-hewitt, J. (2006). Security patterns repository, version 1.0. http://www.scrypt.net/celer/securitypatterns/repository.pdf. [Accessed on: 12/7/2017].

Moreira, R., C.R. Paiva, A., and Memon, A. (2013). A pattern-based approach for gui modeling and testing. In *2013 IEEE 24th International Symposium on Software Reliability Engineering, ISSRE 2013*, pages 288–297.

Moreira, R. M. L. M., Paiva, A. C. R., Nabuco, M., and Memon, A. (2017). Pattern-based GUI testing: Bridging the gap between design and quality assurance. *Softw. Test., Verif. Reliab.*, 27(3).

netsparker (2017). Netsparker web application security scanner. //www.netsparker.com/. [Accessed on 27/7/2017].

Paiva, A. C. R. and Vilela, L. (2017). Multidimensional test coverage analysis: PARADIGM-COV tool. *Cluster Computing*, 20(1):633–649.

portswigger (2017). Automated crawl and scan. //portswigger.net/burp/. [Accessed on 27/7/2017].

Schieferdecker, I., Grossmann, J., and Schneider, M. A. (2012). Model-based security testing. In *Proceedings 7th Workshop on Model-Based Testing, MBT 2012, Tallinn, Estonia, 25 March 2012.*, pages 1–12.

Slavin, R., Shen, H., and Niu, J. (2012). *Characterizations and boundaries of security requirements patterns*, pages 48–53.

SPC (2017). Security Pattern Catalog. http://munawarhafiz.com/securitypatterncatalog/index.php. [Accessed on: 12/7/2017].

sqlmap (2017). Automatic sql injection and database takeover tool. //sqlmap.org/. [Accessed on 27/7/2017].

sqlninja (2017). A sql server injection and takeover tool. //sqlninja.sourceforge.net/. [Accessed on 27/7/2017].

Utting, M. and Legeard, B. (2007). *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Vega (2017). vega vulnerability scanner. //subgraph.com/vega. [Accessed on 27/7/2017].

w3af (2017). w3af. //w3af.org/. [Accessed on 27/7/2017].

wapiti (2017). Wapiti – the web-application vulnerability scanner. //wapiti.sourceforge.net/. [Accessed on 27/7/2017].

Yoder, J. and Barcalow, J. (1998). Architectural patterns for enabling application security.

ZAP (2017). The owasp zed attack proxy (zap). //www.zaproxy.org/. [Accessed on 27/7/2017].