Security Testing in Agile Web Application Development - A Case Study Using the EAST Methodology

Gencer Erdogan[1], Per Håkon Meland[2], and Derek Mathieson[1]

[1] CERN - The European Organization for Nuclear Research
CH-1211 Genève 23, Switzerland
{Gencer.Erdogan,Derek.Mathieson}@cern.ch
[2] SINTEF ICT, System development and security
NO-7465 Trondheim, Norway
Per.H.Meland@sintef.no

Abstract. There is a need for improved security testing methodologies specialized for Web applications and their agile development environment. The number of web application vulnerabilities is drastically increasing, while security testing tends to be given a low priority. In this paper, we analyze and compare Agile Security Testing with two other common methodologies for Web application security testing, and then present an extension of this methodology. We present a case study showing how our Extended Agile Security Testing (EAST) performs compared to a more ad hoc approach used within an organization. Our working hypothesis is that the detection of vulnerabilities in Web applications will be significantly more efficient when using a structured security testing methodology specialized for Web applications, compared to existing ad hoc ways of performing security tests. Our results show a clear indication that our hypothesis is on the right track.

Keywords: Security testing, Web applications, Scrum.

1 Introduction

Using agile methodologies for Web application development is a growing trend [1] and there is evidence that confirms that this is a good fit [2]. Web applications are routinely exposed to malicious attacks, and in order to mitigate the security risks, several ways of integrating security into agile development methodologies [3,4,5,6,7,8] have been suggested. Most security methodologies are built on traditional development methodologies, which are qualitatively and quantitatively different from agile development methodologies [5]. Furthermore, the traditional security methodologies are often sequential rather than iterative. This often leads to a "big design up front" in order to assess the security of a system [3], which is considered as an anti-pattern in the Agile Manifesto [9].

The number of web application vulnerabilities is drastically increasing. In their Global Internet Security Threat Report, Symantec reports that they detected 499,811 new malicious code threats during the second half of 2007, which

A. Sillitti et al. (Eds.): XP 2010, LNBIP 48, pp. 14–27, 2010.

is a 571% increase over the second half of 2006 [14]. These numbers indicate that security tends to be overlooked [15]. Particularly, the provision of sufficient security testing of Web applications is often neglected because of their short time-to-market, and the difficulty of proving a significant payoff for the effort [10,11,12].

In this paper, we analyze and compare Agile Security Testing (as defined by Tappenden et al. [17]) with two other common methodologies for Web application security testing. We then present an extension of this methodology, named EAST, which has been specialized for Web applications development in combination with Scrum. Furthermore, we present a case study showing how EAST performs compared to a more ad hoc approach used within an organization, before we discuss our results and conclude the paper.

2 Agile Security Testing for Web Based Applications

The use of agile development methodologies is a growing trend for Web application development. This has further lead to the idea of agile security engineering, which adopts the same philosophy as agile software engineering in order to mitigate security risks in software [17]. It is a highly iterative process for delivering the security solution and translating security objectives (requirements) into automated security test cases. In addition, it promotes the idea of creating security test cases before the system exists, i.e., test driven development (TDD).

2.1 Agile Security Testing

The Agile Security Testing methodology, suggested by Tappenden et al. [17], consists of three main steps. **Step 1**, the modeling of security requirements, is executed by creating abuser stories [18,19] and/or misuse cases [20,21] in order to elicit security requirements. These are then used as reference points when testing for security in order to verify or falsify a given security requirement. **Step 2**, a highly testable architecture, is achieved by adding a test layer on top of each of the three layers that Web applications typically consist of, i.e., presentation layer, business service layer and data service layer. The resulting architecture is very well suited to agile development methodologies because of its many test layers. Additionally, it is useful for security testing because the architecture makes it possible to employ various security testing techniques within any number of the test layers. **Step 3**, running automated security tests, which is necessary in order to fully benefit from Agile Security Testing.

2.2 Penetration Testing

Penetration Testing is the most commonly applied security testing methodology, but it is also the most commonly misapplied security testing methodology [22]. It is misapplied firstly, by being carried out at the end of the development life cycle and secondly, by being performed in a “time boxed” manner where a small and predefined portion of time and resources is given to the effort. In order

to prevent the misapplication of penetration testing, Thompson [23] suggests a structured penetration testing methodology. Although this methodology is more formal than Agile Security Testing, it is applicable to Web application development and consists of five main steps. **Step 1** is to create a threat model in order to get a detailed, written description of the risks that threatens the application. The goal is to get an overview of the various conditions (vulnerabilities) that have to be present in order to realize a given threat. **Step 2** is to build a test plan. The test plan acts as a road map for the total security testing effort. It is created to get a high-level overview of the security test cases, an overview of how exploratory testing (i.e., simultaneous learning, test design, and test execution) will be conducted, and to get an overview of the components that will be tested. **Step 3** is to execute the security tests. These are divided into four main groups; dependency testing, user interface testing, design testing and implementation testing. **Step 4** is to create a report of the findings from the security testing process. The report must at least cover reproduction steps, severity and exploit scenarios. **Step 5** is to execute a postmortem evaluation. A postmortem evaluation is basically a meeting held by the security test team where the focus should be on why vulnerabilities (bugs or flaws) were missed during development, and how to improve the process to prevent or isolate such security issues in the future.

2.3 The Open Web Application Security Project (OWASP) Testing Framework

The OWASP Testing Framework is not developed for a specific development process, but is rather a comprehensive generic development model that contains the necessary activities needed for systematic security testing of Web applications. This framework consists of five main phases [24] where each phase has its associated activities. **Phase 1** (before development begins): Review policies and standards, and develop measurement and metrics criteria (ensure traceability). **Phase 2** (during definition and design): Review security requirements, review design and architecture, create and review UML models, and create and review threat models. **Phase 3** (during development): Code walkthroughs and code reviews. **Phase 4** (during deployment): Application penetration testing and configuration management testing. **Phase 5** (maintenance and operations): Conduct operational reviews, conduct periodic checks and ensure change verification.

2.4 Comparison of Methods

The Penetration Testing Approach and the OWASP Testing Framework are applicable in Web application development, but they are not very suitable in an agile setting without customization. They were not developed with the Agile Manifesto in mind. E.g., step 2 and step 4 in the Penetration Testing Approach are heavily dependent on documentation. This also applies for phase 2 in the OWASP Testing Framework. Furthermore, as mentioned in Section 2.3, the OWASP Testing Framework is created for a general software life cycle, which

makes it possible to pick and choose the necessary phases. However, the activities are sometimes closely coupled. E.g., in phase 3; to only carry out a code walkthrough without carrying out a code review afterwards would not be of any particular benefit.

The Agile Security Testing methodology has a rather low complexity compared to the other two methodologies. There are three main steps that needs to be carried out compared to five and thirteen steps in the Penetration Testing Approach and the OWASP Testing Framework respectively. These three main steps require little intervention by security experts (but does not eliminate the need completely). All methodologies work with threats and/or security requirements, which can reduce the scope of the testing and helps to decrease the knowledge gap between security experts and software developers. None of the methodologies have any step or activity for mitigating false positives, which is one of the great real-life challenges. In Agile Security Testing we also miss steps for postmortem evaluations and describing security decisions, which are found in at least one of the others.

3 Extending Agile Security Testing and Integrating It into Scrum

As shown in the previous section, there are gaps between Agile Security Testing and methodologies more specific to Web application development (but which are not typically agile). We have therefore defined our own variant named *Extended Agile Security Testing* (EAST) (see Figure 1), which seeks to close these gaps by including three new complementary activities:

Penetration testing and mitigating false positives: False positives (non-existent bugs that are reported as detected by a testing tool) and false negatives (i.e., existing bugs that are not detected by a testing tool) are known to be a problem in automated software security testing. A high rate of false positives creates high workload and makes it difficult to find and fix the actual bugs in the software. We therefore integrate a false positive mitigation process with the penetration testing process. The penetration testing process is regarded as a part of step 3 in Agile Security Testing. The false positive mitigation process is to be carried out in the following way:

1. The penetration testing tool is used to perform a penetration test.
2. After a penetration test, the result is reviewed and the false positives are marked so that they will not be registered as bugs next time the tool performs a penetration test. Each marked false positive vulnerability is associated with its respective Web site.

However, this approach is dependent on either: (1) the penetration testing tool having the ability to mark and remember specific false positives or, (2) the penetration testing tool having the ability to import a false positive repository (e.g., false positive database, XML file, etc.).

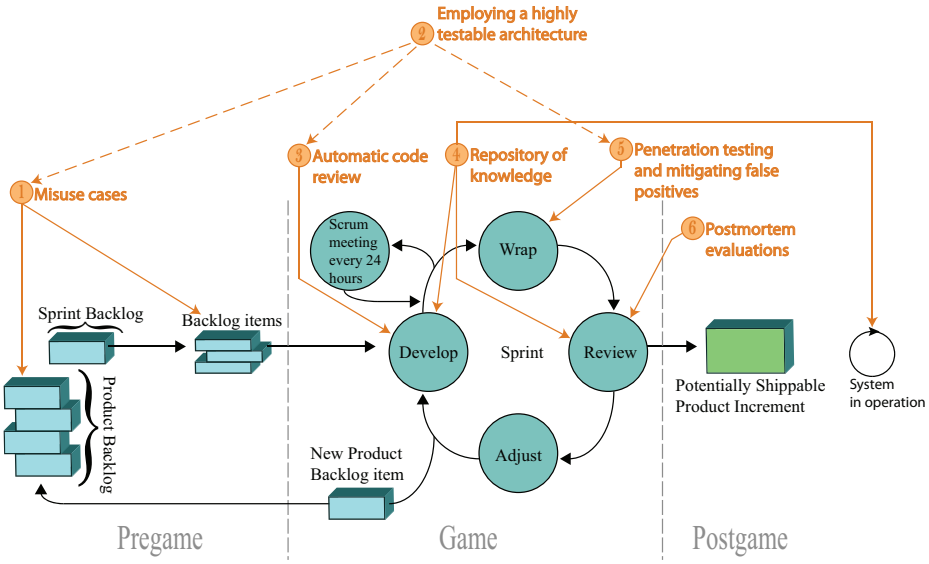

Fig. 1. The EAST steps integrated in the appropriate phases of Scrum

Postmortem evaluations: In order to continuously harden the security testing process, a postmortem phase needs to be in place at the end of the security testing process. It is realized by executing the following steps:

1. Provide answers to why certain vulnerabilities were missed during development.
2. Improve the issued development process in order to mitigate or isolate the underlying vulnerabilities.
3. Create, or find, or improve a security testing activity in order to detect the underlying vulnerabilities.

Repository of knowledge: According to Rus et al. [26], a software organization's main asset is its intellectual capital. One obvious problem in this respect is that intellectual capital lies within the minds of the employees. As experienced people leave the organization, so does the knowledge with them. In order to keep the knowledge alive within the organization, knowledge management must be present in the organization. Knowledge management is beyond the scope of this paper, but by adding a phase in Agile Security Testing that triggers the security testing participants (e.g., developers, QA people, decision makers etc.) to document and archive why certain security decision were made, security specific knowledge can be kept in repositories, and thereby kept alive within the organization. This is realized by executing it during the development and the review phase of the development life cycle. Additionally, it is important to maintain the repository for the underlying system while it is in operation.

As shown in Figure 1, EAST has been integrated into Scrum. Scrum was selected mainly because of its general popularity, according to a survey carried out by Davidson [27], the most widely used agile development methodology is Scrum, followed by eXtreme Programming. Scrum was also the prevalent methodology in our case study organization. The following points describe *how* and *why* the six EAST steps are integrated in Scrum as indicated in Figure 1:

EAST Step 1 (Misuse cases):
- *How*: The pregame phase in Scrum consists of planning and system architecture/high level design, which is carried out by creating product backlog items that are further refined into sprint backlog items. The creation of misuse cases are therefore executed in two steps in the pregame phase, in which step 1 is optional and step 2 is mandatory:
  1. During the creation of product backlog items, high level misuse cases (i.e., misuse cases that contain high level specifications of the system) are created for each product backlog item. Since the level of system specifications in a product backlog item is at a high level (not refined), the resulting misuse cases will also contain high level specifications. This step can be skipped if the product backlog item does not contain enough details in order to create misuse cases (e.g., missing details about the system architecture and design).
  2. When a sprint backlog item is refined into several backlog items (indicated in the pregame phase in Figure 1), the system specifications are well defined and set up for development. The same transformation is to be applied on the misuse cases created in step 1 in order to create detailed misuse cases. If no misuse cases were created in step 1, they must be created containing detailed specifications of the system in this step. Finally, security requirements are to be derived using the resulting misuse cases.
- *Why*: First, misuse cases let developers to think like an attacker (malicious user) and thereby enables them to get an overview of potential threats and vulnerabilities the evolving system may possess. Second, by using the misuse cases as a starting point the developers can create security requirements. The security requirements are then used to verify whether the system fulfils the required level of security (during the security testing process). Third, the creative process of creating misuse cases let developers gain security specific knowledge. Last but not least, discovering vulnerabilities and creating countermeasures during definition, high level design, and low level design are the three most cost efficient ways of mitigating vulnerabilities [28].

EAST Step 2 (Employing a highly testable architecture):
- *How*: Unit security testing is achieved by performing automatic code review during the development phase (EAST step 3). System security testing is achieved by performing penetration testing after the creation of an executable version of the part or parts of the system during the wrap phase (EAST step 5). Security acceptance testing is achieved by using the security requirements (EAST step 1) as reference points to verify or falsify the required

level of security. Furthermore, EAST step 3 can be regarded as development testing, EAST step 5 can be regarded as system testing, and EAST step 1 can be regarded as basis for acceptance testing.

- *Why*: A highly testable architecture is both useful for agile development methodologies and security testing. The highly testable architecture introduces test layers on top of the Web application layers as explained in Section 2. It is therefore an architecture that suits agile development methodologies very well. Furthermore, it makes it possible to apply various security testing techniques within any number of the test layers (automatic code reviewing, penetration testing, etc.).

EAST Step 3 (Automatic code review): – *How*: The development phase in Scrum consists of the following activities: Domain analysis, design, development, implementation, testing and documentation. Since testing is one of the activities, automatic code review is to be carried out in this phase while source code is being developed. I.e., for each unit (e.g., a class) the developer finishes, he or she must perform automatic code analysis on that particular unit. This is to be carried out using a static analysis tool.

- *Why*: By integrating automatic code review in the development phase, developers are able to correct the existing bugs at an early stage. Furthermore, this process continuously hardens the source code against security bugs. However, in order for this activity to be of maximum benefit, the developers need to have experience using the underlying static analysis tool and have security specific knowledge [41].

EAST Step 4 (Repository of knowledge): – *How*: Every security decision that has been made during the development phase and the review phase must be documented. More specifically, this has to be done during the documentation activity in the development phase and while reviewing potential risks in the review phase. The goal is to justify and document why certain security specific decisions were made. Additionally, the repository needs to be updated whenever a vulnerability is discovered while the system is in operation (illustrated by the arrow going from EAST step 4 to the "System in operation" loop in Figure 1). The level of detail on the justification may vary, but it must at least contain the following points:

  - **Application:** The name of the application that the security decision applies for.
  - **Decision ID:** An ID for the given security decision.
  - **What:** A short explanation of what the security decision is.
  - **Where:** The name of the affected part(s) of the application(s) due to the security decision (class(es), module(s), etc.).
  - **How:** A short explanation of how the security decision is realized.
  - **Why:** A short explanation of why the given security decision was made.

- *Why*: By documenting security specific decisions that has been made during development it is possible to keep the decisions in a repository,

and thereby possible to keep security specific knowledge alive within the organization. E.g., for training purposes and for tracing earlier security specific decisions in order to understand why certain things are done the way they are. At first glance, this step may be regarded as a contradiction to one of the key thoughts in agile development, which is to document as little as possible [9]. However, there is a knowledge gap between security experts and software developers [29]. Additionally, there is a risk of losing years of knowledge when people quit their position. We have therefore added this step in the EAST methodology in order to mitigate the knowledge gap and to mitigate the loss of security specific knowledge within the organization. Furthermore, the documentation of such security specific decisions are not comprehensive, but rather a brief summary and justification of the underlying security decision.

EAST Step 5 (Penetration testing and mitigating false positives)
- *How*: After an executable part or parts of a system is created in the wrap phase, a penetration test using a Web Vulnerability Scanner has to be carried out on the executable part(s). The penetration testing results are then to be analyzed and the false positives are to be marked. The false positives are to be marked as explained earlier in this section.
- *Why*: By performing penetration tests in the wrap phase makes it possible to discover vulnerabilities in the application during a sprint (continuously). This creates a base for the review phase in which, among other things, risks are discussed, countermeasures are created and EAST Step 6 (Postmortem evaluations) is carried out. Furthermore, by continuously marking false positives, the testing tools' knowledge base is improved to better understand the application under test, and consequently report fewer false positives in subsequent iterations.

EAST Step 6 (Postmortem evaluations):
- *How*: During the review phase there has to be a postmortem evaluation meeting session. The postmortem evaluation is to be carried out after the wrap phase and EAST Step 5. Furthermore, it is to be carried out as explained earlier in this section.
- *Why*: This step enables the security testing participants to reflect over the vulnerabilities, the development process and the security testing process. This is important in order to continuously improve the security testing process.

4 Case Study

Our case study was carried out in the Administrative Information Services (AIS) group at CERN - The European Organization for Nuclear Research. The AIS group has the responsibility for all administrative applications and corporate data at CERN, and has currently six main applications in operation for users inside and outside CERN. One of the six applications is the Electronic Document Handling (EDH), which has approximately 14 000 users worldwide and

is CERN's largest administrative Web application. The services that are provided by EDH are called EDH documents, which are basically Web forms. The security tests in our case study were applied to EDH, but since EDH is very large and complex, only two of the most frequently used EDH Web forms were considered in our experiment; Internal Purchase Request (DAI) and Material Request (MAG). Three developers volunteered to participate in the case study, with *Acunetix WVS* [39] and *PMD* [38] as the main testing tools, and *SeaMonster* [40] for security modelling. The case study consisted of two parts:

The first part was to execute the security testing in four iterations; two iterations using the security testing methodology normally applied by the AIS group (which is an ad hoc way of performing security tests), and two iterations using the EAST methodology. Furthermore, the testing iterations were executed in the following chronological order: *The AIS group's methodology* → *EAST* → *the AIS group's methodology* → *EAST*. The security testing methodology that is applied by the AIS group is carried out in two main steps:

1. A penetration testing is carried out in the postgame phase. Web Vulnerability Scanners are not used. Instead, guidelines are used by the testing participants to manually perform penetration tests.
2. The participants creates a report of the findings after the penetration testing. The report is used as a basis to create countermeasures for the vulnerabilities. Then, the countermeasures are added in the product backlog. Finally, the vulnerability is mitigated for the next product increment by performing the Scrum phases on the particular backlog item containing the vulnerability countermeasures.

Additionally, the penetration tests are often carried out in an ad hoc fashion, i.e., they are not always carried out before each and every "Potentially Shippable Product Increment" step (see Figure 1).

The second part was to evaluate EAST based on the results from the security testing iterations.

Automated security tests can last for hours given the limitless supply of Web application vulnerabilities [30,31,32], and the vast array of automated vulnerability tests a Web Vulnerability Scanner can perform. We therefore used the OWASP top 10 vulnerabilities [33] as a starting point and decided to test for the following vulnerabilities:

V.01 Reflected Cross Site Scripting
V.02 Stored Cross Site Scripting
V.03 SQL Injection
V.04 Malicious File Execution
V.05 Insecure Direct Object Reference
V.06 Cross Site Request Forgery (CSRF)
V.07 Information Leakage and Improper Error Handling
V.08 Broken Authentication and Session Management
V.09 Failure to Restrict URL Access

The top two vulnerability classes defined by OWASP are Cross Site Scripting (XSS) and Injection Flaws. Each of these vulnerability classes consists of four and eight vulnerability "types" respectively. E.g., Cross Site Scripting consists of: Reflected XSS, stored XSS, DOM XSS and Cross Site Flashing. V.01 was tested because Reflected XSS is known to be the most frequent type of XSS attack [34]. V.02 was tested because Stored XSS is known to be the most dangerous type of XSS attacks [35]. V.03 was tested because SQL injection is one of the most frequently applied attack type [36]. Insecure Cryptographic Storage and Insecure Communications from the OWASP top 10 list were not tested. The former is not possible to test using automated security scanning tools, and the latter was not tested because EDH uses Hypertext Transfer Protocol Secure (HTTPS) during all communication it has with a client.

5 Results and Discussion

There are three factors that were used as basis to compare the efficiency of the security testing methodology applied by the AIS group and the EAST methodology, respectively:

1. The amount of time spent on the security testing process.
2. The amount of vulnerabilities found during the security testing process.
3. The ability to mitigate false positives during the security testing process.

Figure 2 shows an overview the test results obtained from each test iteration. By studying the results, we discovered the following:

– By comparing the average time spent on testing both DAI and MAG using the security testing methodology applied by the AIS group, versus the

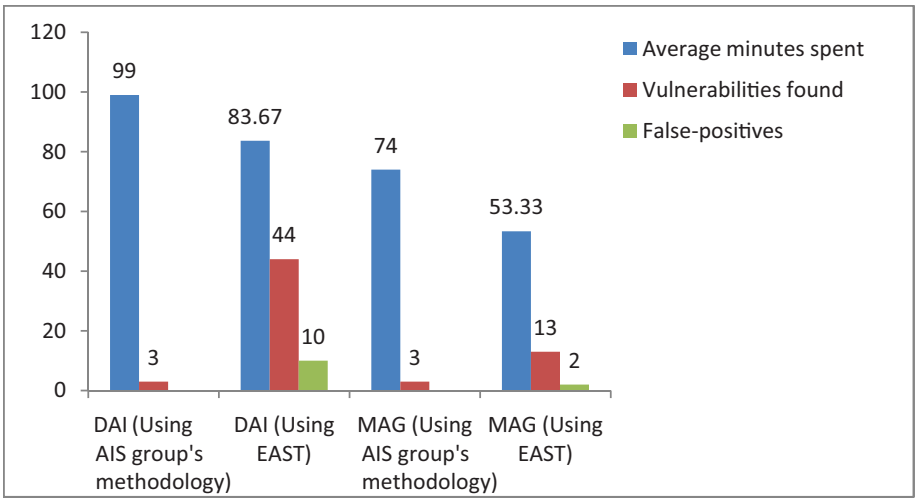

Fig. 2. Comparison of the test results

average time spent on testing both DAI and MAG using the EAST methodology, we discovered that the EAST methodology is in average 21% more effective.
- The number of vulnerabilities found by testing DAI and MAG using the security testing methodology applied by the AIS group is 3, while the number of vulnerabilities found by testing DAI and MAG using the EAST methodology is 56. In this respect, the EAST methodology is approximately 95% more effective. However, the three vulnerabilities that were found by using the security testing methodology applied by the AIS group were not found when using the EAST methodology. This indicates that human intervention is necessary in circumstances where tools have limited capabilities.
- By testing DAI and MAG using the EAST methodology, there were found 12 false positives. The false positives were marked via Acunetix WVS and thereby added to a false positive repository. Next time DAI and MAG were scanned, the marked false positives were regarded as false positives and not as vulnerabilities. In turn, this mitigates false positives during the security testing process. The security testing methodology applied by the AIS group does not have an activity in place that mitigates false positives. Hence, only the EAST methodology mitigates false positives during the security testing process. However, this approach to false positive mitigation is dependent on either: (1) the penetration testing tool (that is being used in the EAST methodology) must have the ability to mark and remember specific false positives or, (2) the penetration testing tool must have the ability to import a false positive repository (e.g., false positive database, XML file).

The following points describe the threats to the generalizability of the test results:

Tools: The usage of one specific Web Vulnerability Scanner is obviously an important factor that affects the results of the security tests. Other Web Vulnerability Scanners could produce different results regarding time spent and vulnerabilities found. Furthermore, it was decided that *PMD* would be used as the automatic code scanner tool. This was due to the participants' previous experience in using PMD. A better starting point in this respect would be to find a tool that produces the minimum amount of false positives. This risk is mitigated by leaning towards the findings done by Baca et al. [41], i.e., the best security testers are those who have both security experience and experience in using the static analyzer tool. Finally, the participants used an unfamiliar tool to model misuse case diagrams. The lack of experience in using this tool would inevitably affect the test results regarding time spent.

Security specific knowledge: The majority of the participants had little security specific knowledge, and this would typically lead to the detection of less vulnerabilities.

Security testing experience: All of the participants had little experience in security testing. This can also be used as a basis to question the validity of the test results. On the other hand, this is another indication showing that structured security testing is still not widely applied in organizations.

This risk can only be mitigated by continuously applying structured security testing using, e.g., the EAST methodology.

Testing thoroughness: The thoroughness of the participants' test execution was not measured. This risk is mitigated (or believed to be mitigated) by the fact that the participants actively volunteered.

Psychological effects, and the number of participants: As explained in Section 4, the test iterations were carried out sequentially. This will have a learning effect on the participants, i.e., for each iteration, the participants would gain more security specific knowledge and more experience in performing the tests. Obviously, this would be to the benefit of the participants, which would further produce better results. In this context, better results means spending less time to complete a test iteration and/or to find more vulnerabilities. Although the participants knew that the purpose of the testing process was to observe the testing methodologies, they may have been more effective while performing the security tests and thereby improving the results, as a response to the fact that they were the ones that produced the results that were taken into consideration. Thus, the participants may have considered themselves as a part of the "object" being observed and thereby improved the test results. This form of reactivity is referred to as the Hawthorne effect. Figure 2 shows that the average time spent on the test iterations decreases for each test iteration. This indicates that there has indeed been a learning effect on the participants during the testing process. Furthermore, the number of participants is another factor that could be used to question the validity of the test results. However, this was a side effect of the limited time and resources at our disposal.

6 Conclusion and Further Work

There is a need for security testing methodologies specialized for Web applications and their agile development environment. The EAST methodology is mainly based on Agile Security Testing, but is more tailored for Web development in combination with Scrum. A case study evaluation performed in an organization showed that compared to the current more ad hoc way of doing security testing, the EAST methodology is approximately 21% more effective in average regarding time spent, approximately 95% more effective regarding the amount of vulnerabilities found, and has the ability to mitigate false positives.

Future evaluations of the EAST methodology should be focused on mitigating the threats to the validity of the results presented in Section 5. They should also address the question of how efficient EAST is compared to extensive security testing methodologies applied on Web applications. These extensive security testing methodologies would naturally require more time and thereby be less efficient regarding time spent. However, what would be interesting to discover, is whether the EAST methodology lacks activities that are vital for the overall security testing process, compared to the traditional security testing methodologies.

References

1. Jazayeri, M.: Some trends in Web application development. In: International Conference on Software Engineering, pp. 199–213. IEEE Computer Society, Washington (2007)
2. McDonald, A., Welland, R.: Agile web engineering (AWE) process. Technical report, Department of Computer Science, University of Glasgow, UK (December 2001)
3. Kongsli, V.: Towards agile security in web applications. In: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (2006)
4. Ge, X., Paige, R.F., Polack, F.A.C., Chivers, H., Brooke, P.J.: Agile development of secure web applications. In: Proceedings of the 6th international conference on Web engineering. ACM, New York (2006)
5. Chivers, H., Paige, R.F., Ge, X.: Agile security using an incremental security architecture. In: Baumeister, H., Marchesi, M., Holcombe, M. (eds.) XP 2005. LNCS, vol. 3556, pp. 57–65. Springer, Heidelberg (2005)
6. Siponen, M., Baskerville, R., Kuivalainen, T.: Integrating security into agile development methods. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, vol. 7, p. 185a (2005)
7. Wayrynen, J., Bodén, M., Bostrom, G.: Security Engineering and eXtreme Programming: An Impossible Marriage? In: Zannier, C., Erdogmus, H., Lindstrom, L. (eds.) XP/Agile Universe 2004. LNCS, vol. 3134, pp. 117–128. Springer, Heidelberg (2004)
8. Beznosov, K.: Extreme Security Engineering: On Employing XP Practices to Achieve "Good Enough Security" without Defining It. In: First ACM Workshop on Business Driven Security Engineering (BizSec), Fairfax, VA (2003)
9. Agile Manifesto, `http://agilemanifesto.org/` (Last date accessed 2009-12-10)
10. Hieatt, E., Mee, R.: Going Faster: Testing The Web Application. IEEE Software 19, 60–65 (2002)
11. Di Lucca, G.A., Fasolino, A.R., Faralli, F., De Carlini, U.: Testing Web applications. In: Proceedings of International Conference on Software Maintenance, pp. 310–319 (2002)
12. Di Lucca, G.A., Fasolino, A.R.: Testing Web-based applications: The state of the art and future trends. Information and Software Technology 48, 1172–1186 (2006)
13. Erdogan, G.: Security Testing of Web Based Applications, M.Sc. in Computer Science, Norwegian University of Science and Technology (2009)
14. Turner, D., Fossi, M., Johnson, E., Mack, T., Blackbird, J., Entwisle, S., Low, M.K., McKinney, D., Wueest, C.: Symantec Internet Security Threat Report: Trends for July-December 2007. Technical report, Symantec Corporation, Vol. XIII (2008)
15. Thompson, H.H.: Why Security Testing Is Hard. IEEE Security & Privacy 1, 83–86 (2003)
16. Zhu, H., Horgan, J.R., Cheung, S.C., Li, J.J.: The first international workshop on automation of software test. In: Proceedings of the 28th international conference on Software engineering. ACM, New York (2006)
17. Tappenden, A., Beatty, P., Miller, J., Geras, A., Smith, M.: Agile security testing of Web-based systems via HTTP Unit. In: Proceedings of Agile Conference, pp. 29–38 (2005)
18. Peeters, J.: Agile Security Requirements Engineering. In: Symposium on Requirements Engineering for Information Security (2005)

19. McGraw, G.: Software Security: Building Security. Addison-Wesley, Reading (2006)
20. Sindre, G., Opdahl, A.L.: Eliciting security requirements with misuse cases. Requirements Engineering 10, 34–44 (2005)
21. Røstad, L.: An extended misuse case notation: Including vulnerabilities and the insider threat. In: The Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality (2006)
22. Arkin, B., Stender, S., McGraw, G.: Software penetration testing. IEEE Security & Privacy 3, 84–87 (2005)
23. Thompson, H.H.: Application penetration testing. IEEE Security & Privacy 3, 66–69 (2005)
24. The Open Web Application Security Project. OWASP Testing Guide V3.0, http://www.owasp.org/index.php/Category:OWASP_Testing_Project (Last date accessed 2009-11-13)
25. Ananta Security. Web Vulnerability Scanners Evaluation, http://www.darknet.org.uk/content/files/WebVulnScanners.pdf (Last date accessed 2009-11-13)
26. Rus, I., Lindvall, M.: Knowledge management in software engineering. IEEE Software 19, 26–38 (2002)
27. Davidson, M.: Survey: Agile interest high, but waterfall still used by many. Agile Trends Survey (2008), http://searchsoftwarequality.techtarget.com/news/article/0,289142,sid92_gci1318992,00.html (Last date accessed 2009-11-26)
28. Wysopal, C., Nelson, L., Dustin, E., Nelson, L., Zovi, D.D.: The Art of Software Security Testing. Addison-Wesley, Reading (2006)
29. Erdogan, G., Baadshaug, E.T.: Extending SeaMonster to support vulnerability inspection modeling. Technical report, NTNU, Department of computer and information science (2008)
30. BugTraq mailing list, http://www.securityfocus.com/archive/1 (Last date accessed 2009-11-13)
31. Common Vulnerabilities and Exposures, http://cve.mitre.org/ (Last date accessed 2009-11-13)
32. Computer Emergency Readiness Team (CERT), http://www.cert.org/ (Last date accessed 2009-11-13)
33. OWASP Top 10 vulnerabilities, http://www.owasp.org/index.php/Top_10_2007 (Last date accessed 2009-11-13)
34. Hope, P., Walther, B.: Web Security Testing Cookbook. O'Reilly, Sebastopol (2008)
35. The Open Web Application Security Project. OWASP Testing Guide V3.0, http://www.owasp.org/index.php/Category:OWASP_Testing_Project (Last date accessed 2009-12-02)
36. Andrews, M.: Guest Editor's Introduction: The State of Web Security. IEEE Security and Privacy 4, 14–15 (2006)
37. Koomen, T., van der Aalst, L., Broekman, B., Vroon, M.: TMap Next: For Result-driven Testing. UTN Publishers (2006)
38. PMD - Java source code scanner (Static Analysis Tool), http://pmd.sourceforge.net/ (Last date accessed 2009-11-14)
39. Acunetix Web Vulnerability Scanner, http://www.acunetix.com/ (Last date accessed 2009-11-14)
40. SeaMonster V3.0, http://sourceforge.net/projects/seamonster/ (Last date accessed 2009-11-14)
41. Baca, D., Petersen, K., Carlsson, B., Lundberg, L.: Static Code Analysis to Detect Software Security Vulnerabilities - Does Experience Matter? In: IEEE International Conference on Availability, Reliability and Security, pp. 804–810 (2009)