Strategies to Maximize the Security Efforts into the Agile Software Development Life Cycle Without Increasing the Headcount

Anderson Maranhão Ventura Dadario
Information Security
Flare Security, São Paulo, Brazil
anderson@dadario.com.br

Abstract

The Scrum [1] agile methodology is on the rise [2] and the security analysts need catch up this new approach to software development in order to minimize the risks and protect the application and infrastructure. However find the injection points to apply security are not trivial since Scrum is more complex than Waterfall methodology [3] and involves four types of meetings, three roles and three artifacts. This study presents how to maximize the security resources allocation and how to take advantage of automation, Extreme Programming [4] engineering practices and delegation of security responsibilities with security champions without increasing the headcount.

Keywords: SDLC Security; Agile; Waterfall; Scrum; Extreme Programming

1. Introduction

The main interest of companies is to maximize business by matching all customer needs in order to create revenue or reduce costs [29]. This led companies to rethink processes that delay or prevent the creation of revenue or increase the costs, such as Waterfall [3] software development methodology that started to be replaced by Scrum [1]. However both were not designed with security in mind.

Scrum [1] is an incremental and iterative software development process that is becoming more popular [2] and is challenging the information security teams to efficiently build more secure software, address compliance requirements and reduce costs [5]. It is challenging because Waterfall [3], the previous widely used [6] software development methodology, was simpler, with fewer interactions and more bureaucratic. Scrum in the other hand is more complex, with a considerable number of interactions and less bureaucratic as possible.

2. Security on Scrum

The first attempt to efficiently apply security on Scrum tends to be the same used on Waterfall [7, 8], by mixing the security phases within the development phases. Utilizing this concept, a common approach to Scrum security is to allocate a security resource to be involved in all types of meeting: daily, planning, review and retrospective. However, as the planning meeting [9] can take up to eight hours and has the purpose to identify the work that need to be delivered in the end of the current sprint, the security resource interacts a very few part of his time, compromising his participation in other activities such as other teams planning meeting.

The second attempt is to use the time wisely and do not let the security resources locked up in these longer planning meetings by introducing a post-planning meeting to discuss only the selected stories and apply security to them, as Veracode experimented [10]. The problem with this attempt is the fact that it breaks the Scrum concept because after a planning meeting, the stories cannot be changed. This happens because the stories already were estimated, and the deliverable is already settled.

The most comprehensive alternative is to add security acceptance criteria in the stories before the planning meeting occurs. The meeting that satisfy this need is the Grooming [11] although it is not part of Scrum. Grooming meeting happens before the planning meeting and is the ongoing process of reviewing product backlog items and checking that they are appropriately prioritised and prepared in a way that makes them clear and executable for teams once they enter sprints via the sprint planning activity.

During the Grooming, the security resource can add security requirements to the acceptance criteria of the stories and create or update the threat modeling [12] of the application and document it. With the documentation, another security resource that was not aware of the application become able to assist in the acceptance criteria and perform security tests such as code review [13], design review [14] or penetration test [15]. Distributing the type of work within security resources avoid the single point of failure [16] originated because of the dependency of a single security resource allocated to all Scrum ceremonies.

Although the maximization of the security resources allocation efficiency is important to successfully inject security into the agile software development life cycle, other aspects propitiated by agile methodologies such as Extreme Programming engineering practices [17] that combines with Scrum should also be contemplated.

3. Extreme Programming Engineering Practices

There are twelve extreme programming engineering practices divided into four areas. As the twelve practices are focused on the development process, not all of them are directly related to information security. The most relevant practices regarding information security are part of the two areas: “Continuous processes” and

"Shared understanding".

In the "Continuous processes" area, there are three practices: "Continuous integration", "Design improvement" and "Small releases". These practices explores the concept of short but multiple iterations present on Scrum and absent on Waterfall. The meaning of continuous processes for the information security team is to split the one-shot risk analysis, testing and other activities performed only one time into activities that need to be performed every sprint, every commit on the source code repository, every release and so on. However, doing multiple times the same activity can be time consuming, so what could be automated needs to be prioritized. This automation can leverage the practice of "Continuous integration" [18] and "Small releases" by adding security tests in the build pipeline [19] such as Static Application Security Testing (SAST) [20] or Dynamic Application Security Testing (DAST) [21]. The vulnerabilities found by these tests can be integrated in a centralized risk management software that will also receive manual input of manual code reviews and manual vulnerability analysis. The manual analysis, depending on the criticism and sensibility of the data stored, processed or transferred, becomes necessary and preferable over an automatic analysis given the capacity to overcome the limitations of automatic analysis such as perform business logic testing and detect the context of the application according to the Open Web Application Security Project (OWASP) Application Security Verification Standard (ASVS) project [22].

The practice "Design improvement" states that as the agile methodology focus on delivering only what is needed, the designed architecture may not support it properly and should be rebuilded. During this design rebuilding, the security resource should participate to apply the security design principles [23] and contribute to architect a solution that will protect future changes from increasing the attack surface and mitigate, avoid or transfer future risks.

The "Shared understanding" area covers the practices "Coding standard", "Collective code ownership", "Simple design" and "System metaphor", but the last one is only about naming concepts. "Simple design" is already part of the security principle known as "Economy of mechanisms" [24].

The "Coding standard" is the reference that all developers must have in mind before programming. Given this importance, there is no better place to define the defensive coding techniques and other secure code practices such as the secure coding practices of OWASP [25].

"Collective code ownership" encourages the developers to work as team and look at the code as a team responsibility and not a responsibility of each developer.

This is important to share the duty to secure the software and do not simply look at a vulnerable code and left it as it is.

Even following these practices, there is still a challenge to handle multiple projects with few security resources. However there are some tasks that can be delegated to non-security resources that will support the security resources. Mozilla already define some [26] for their projects.

4. **Security Responsibilities Delegation**

As mentioned in the Microsoft Secure Development Lifecycle (SDL) process guidance [27] and on Mozilla security champions page [26], exists a role named 'security champion' that is assumed by a development team member and has the responsibilities of acting in the behalf of the security resources but with lower autonomy. The person that will assume this role needs to be properly trained in more depth than those that only need to attend the general security awareness training since activities such as explaining the security vulnerabilities for the team, spread the Rugged Software manifesto [28] among the team, identify and document risks and perform secure code review may become attributions of the security champion.

The election of the security champion may be *ad hoc* or through a structure process followed by specific training and custom certifications accredited by the security team. The specific training must contemplate the security champion responsibilities and tasks, and should be reinforced in a regular basis as any other security awareness training according to Microsoft SDL [27].

5. **Conclusions**

Neither the Waterfall and the Scrum methodology were designed with security in mind, so the security community needed to identify the injection points to apply security. The security applied to these methodologies was not mature in the beginning so the gaps as described in this study were identified. There is a need to understand both Scrum and Extreme Programming.

With the understanding of the development methodologies it was possible to identify that Grooming meeting are extremely useful to inject security earlier instead of Planning meetings.

The separation of duties between types of security resources helps to avoid the single point of failure originated by the dependency of a single security resource.

The application of the extreme programming engineer practices mostly related to information security helps to integrate security seamlessly to the software development life cycle, as the security champion role also do.

The security champion helps to increase the security team without adding a new security resource. This benefit helps to spread security awareness and delegate information security tasks.

6. **References**

[1] http://en.wikipedia.org/wiki/Scrum_(software_development) Retrieved 2014-07-16.
[2] http://www.google.com/trends/explore#q=Scrum&cmpt=q Retrieved 2014-07-16.
[3] http://en.wikipedia.org/wiki/Waterfall_model Retrieved 2014-07-16.
[4] http://en.wikipedia.org/wiki/Extreme_programming Retrieved 2014-07-16.
[5] http://www.microsoft.com/security/sdl/about/benefits.aspx Retrieved 2014-07-16.
[6] http://www.google.com/trends/explore#q=waterfall%20methodology Retrieved 2014-07-16.
[7] http://www.onpointcorp.com/uploads/137/doc/Security_in_the_SDLC.pdf Retrieved 2014-07-16.
[8] http://www.rsaconference.com/writable/presentations/file_upload/asec-107.pdf Retrieved 2014-07-16.
[9] http://en.wikipedia.org/wiki/Scrum_(software_development)#Sprint_planning_meeting Retrieved 2014-07-16.
[10] https://info.veracode.com/webinar-building-security-into-the-agile-sdlc.html Retrieved 2014-07-16.
[11] http://en.wikipedia.org/wiki/Scrum_(software_development)#Backlog_refinement_.28grooming.29 Retrieved 2014-07-16.
[12] http://en.wikipedia.org/wiki/Threat_model Retrieved 2014-07-16.
[13] http://en.wikipedia.org/wiki/Code_review Retrieved 2014-07-16.
[14] https://www.owasp.org/index.php/OWASP_Secure_Application_Design_Project Retrieved 2014-07-16.
[15] http://en.wikipedia.org/wiki/Penetration_test Retrieved 2014-07-16.
[16] http://en.wikipedia.org/wiki/Single_point_of_failure Retrieved 2014-07-16.
[17] http://en.wikipedia.org/wiki/Extreme_programming_practices Retrieved 2014-07-16.
[18] http://en.wikipedia.org/wiki/Continuous_integration Retrieved 2014-07-16.
[19] http://www.infoq.com/articles/orch-pipelines-jenkins Retrieved 2014-07-16.
[20] http://en.wikipedia.org/wiki/Static_program_analysis Retrieved 2014-07-16.
[21] http://blogs.gartner.com/it-glossary/dynamic-application-security-testing-dast/ Retrieved 2014-07-16.
[22] https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project Retrieved 2014-07-16.
[23] https://buildsecurityin.us-cert.gov/articles/knowledge/principles/design-principles Retrieved 2014-07-16.
[24] https://buildsecurityin.us-cert.gov/articles/knowledge/principles/economy-of-mechanism Retrieved 2014-07-16.
[25] https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide Retrieved 2014-07-16.
[26] https://wiki.mozilla.org/Security/Champions Retrieved 2014-07-16.
[27] http://www.microsoft.com/en-us/download/details.aspx?id=29884 Retrieved 2014-07-16.
[28] https://www.ruggedsoftware.org/ Retrieved 2014-07-16.
[29] http://en.wikipedia.org/wiki/Profit_motive Retrieved 2014-07-16.