Architectural Risk Analysis in Agile Development of Cloud Software

Martin Gilje Jaatun*†
*Software Engineering, Safety and Security
SINTEF Digital
Trondheim, Norway
†Department of Electrical Engineering and Computer Science
University of Stavanger
Stavanger, Norway
Email: martin.g.jaatun@{sintef.no,uis.no}

***Abstract*—Software in the cloud is predominantly developed using agile methodologies, where practices such as continuous deployment and DevOps contribute to increased speed and quick turnarounds. This increased speed does however require additional focus on software security in order to avoid security bugs and architectural flaws from crippling a cloud business. Architectural Risk Analysis has been touted as one of the most powerful software security activities, but in some agile development projects there is no distinct architecture activity, and often no dedicated architects. In this paper we will examine the challenges of performing Architectural Risk Analysis in agile development projects.**

Keywords: Architectural Analysis, Threat Modeling, Cloud Software, Software Security, Secure Software Engineering

I. Introduction

Software in the cloud is predominantly developed using agile methodologies, where practices such as continuous deployment and DevOps contribute to increased speed and quick turnarounds [1]. This increased speed does however require additional focus on software security in order to avoid security bugs and architectural flaws from crippling a cloud business.

Software security can, at the risk of over-simplification, be reduced to two facets: Avoiding security bugs (coding errors) and avoiding security flaws (design errors). Architectural Risk Analysis (often referred to as simply Architectural Analysis) was identified by McGraw [2] as the second most effective software security activity among the Cigital Touchpoints, and the single most effective activity for identifying security design flaws.

Unfortunately, with the current popularity with agile development methodologies for developing cloud software, the focus on architecture appears to be diminished in many development organizations. However, whether or not the developers are paying attention to it, the architecture is still *there* [3], and architectural flaws are just as devastating. The challenge is thus to ensure that the architectural risk analysis can happen without breaking the agile paradigm.

The remainder of the paper is structured as follows: In Section II we present relevant background. In Section III we present a preliminary case study. We discuss further in Section IV, and conclude in Section V.

II. Background

Abrahamsson et al. [4] argue that not all design decisions are architectural, and sometimes they are made on day one. Some of these decisions are relevant for security, and when they are made without taking security into account, the results can be detrimental to security.

A. The Microsoft Security Development Lifecycle

Around the turn of the century, Microsoft Windows users were exposed to a series of different malware; names such as Iloveyou, Code Red and Nimda are sure to evoke fond memories among professionals with a few years under their belts. This eventually grew to a liability of such proportions that Bill Gates i 2002 wrote a memo to all employees, where he announced the introduction of the Microsoft Trustworthy Computing Initiative. This led to a two-month stop in development of new features, and more than 9000 developers were sent for security training (The underlying problems could of course not be solved immediately, and in 2002 and 2003 the community made the acquaintance of Beast, Slammer and Blaster, that caused equally serious problems).

Then, in 2004 Michael Howard and Steve Lipner presented the Microsoft Security Development Lifecycle, tailored after a traditional waterfall development lifecycle. After a few iterations, Microsoft SDL has settled in the form described in Table I [5].

As originally defined, Microsoft SDL is not compatible with an agile development methodology, so a few years ago Microsoft created a more agile variant dubbed "SDL Agile" [6]. The change primarily lies in categorizing the various SDL activities in one of three groups:

- Performed in every sprint (7,8,9,10,15,16)

TABLE I
ORIGINAL MICROSOFT SECURITY DEVELOPMENT LIFECYCLE

1. Training	1. Core security training
2. Requirements	2. Establish security requirements 3. Create quality gates/bug bars 4. Perform Security and Privacy risk assessments
3. Design	5. Establish design requirements 6. Perform attack surface analysis/reduction 7. Use threat modeling
4. Implementation	8. Use approved tools 9. Deprecate unsafe functions 10. Perform Static Analysis
5. Verification	11. Perform dynamic analysis 12. Perform fuzz testing 13. Conduct attack surface review
6. Release	14. Create an incident response plan 15. Conduct final security review 16. Certify release and archive
7. Response	17. Execute incident response plan

- Performed once per project (1,2,5,14)
- Bucket activities; a selection is performed in every sprint (3,4,6,11,12,13)

Microsoft provides lists of activities that are even more detailed than in Table I [6]. SDL Agile is illustrated in Fig. 1, where the bucket activities are depicted as "pie wedges".

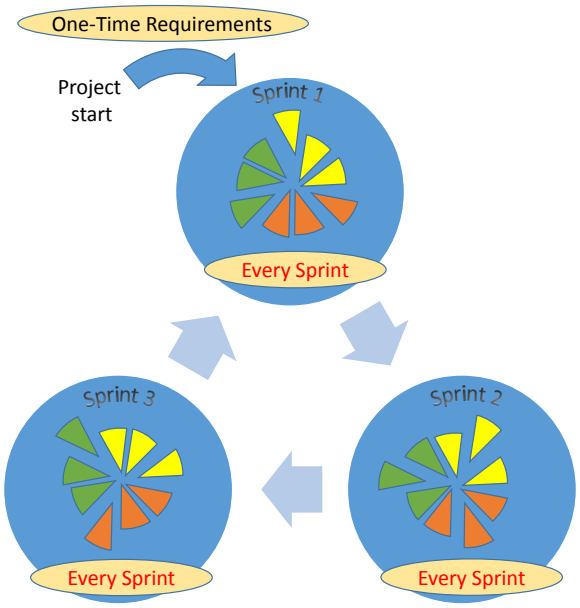

Fig. 1. MS SDL for Agile

B. Touchpoints

Gary McGraw [7] introduced an artifact-oriented secure software development lifecycle (SSDL) known as The Cigital[1] Touchpoints (see Fig. 2). In order of effectiveness, McGraw lists the touchpoints as:

1) Code review
2) Architectural risk analysis
3) Penetration testing
4) Risk-based security tests
5) Abuse cases
6) Security requirements
7) Security operations

As mentioned, the second most effective touchpoint is thus architectural risk analysis.

C. Architectural Risk Analysis

Architectural risk analysis in the context of software security has been described by McGraw [7] as follows:

- Start with a forest-level view of the architecture
- Determine the system's attack resistance (by using checklists, e.g., STRIDE – see below)
- Perform ambiguity analysis
- Identify vulnerabilities in underlying frameworks

The observant reader will see that this is pretty much the same as what is known as threat modeling as defined by (among others) Howard & Lipner [5] and Swiderski & Snyder [8], although it was defined by Cigital (now: Synopsys) as a distinct process [7].

According to McGraw[7], architectural risk analysis can be done at different stages of the development life cycle, but it certainly makes sense to do this at least in the design phase. Kruchten advocates that a certain minimum of *architecting* should be performed also for agile development projects, and worries that "the last responsible moment" for making architecture decisions may be misjudged [3],

[1]The software security consultancy Cigital, co-founded by Gary McGraw, was acquired by Synopsys in 2016

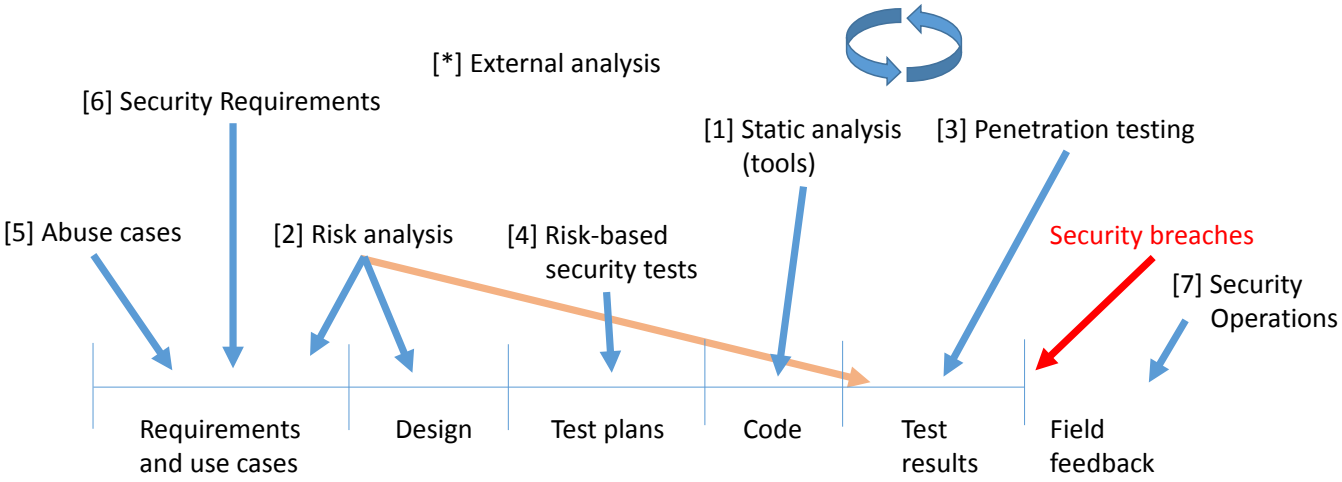

Fig. 2. The Cigital Touchpoints

leading to unfortunate "organic" architecture that later needs to be changed at considerable cost.

The "forest-level" view of the architecture could be based on anything; UML if you are of that persuasion, or any other form of boxes and arrows. We have found [9] that data flow diagrams work well for this purpose (see below). Often, the very act of drawing this representation on (e.g.) a whiteboard can lead to important insights.

The forest-level view is then used as a basis for determining the system's attack resistance. It is often useful to use some sort of checklist for this purpose, STRIDE is a popular choice, but also other (possibly more domain-specific) options could work.

Ambiguity analysis consists in two (or more) security experts studying the available architecture documentation individually, and then meeting to each present their individual take on it. Whenever there is a disagreement, this is a sign of ambiguity in the specification, and more effort is needed to determine any possible security implications of this ambiguity.

Finally, any underlying frameworks or libraries must be examined for known vulnerabilities, whether the latest versions of libraries have been used, and whether patches have been applied.

D. Other Forms of Architectural Analysis

What is the relation between architectural risk analysis and security architecture? Is it similar to the comparison between security features and secure features? Ryoo et al. [10] point to the Architecture Tradeoff Analysis Method (ATAM) [11], and suggest three approaches to architectural analysis based on vulnerabilities, patterns, and tactics, respectively. However, Ryoo's proposal seems rooted in classical waterfall-style development, and thus not particularly agile.

Use of security patterns is also advocated by Halkidis et al. [12], who present a method for mathematically calculating the security risk of a specific architecture provided that security patterns have been used in the construction of the architecture. Whereas security patterns doubtlessly would have been a good thing, in our experience they are not used by the vast majority of software development organizations, and this method is thus also not applicable to our use case.

E. STRIDE - a Mnemonic for Attacks

The STRIDE mnemonic was introduced by Microsoft [8] as an aid to threat modeling. The letters spell out the following threats:

- Spoofing
- Tampering
- Repudiation
- Information disclosure
- Denial of Service
- Elevation of privilege

STRIDE has been criticized as a poor choice for performing threat modeling, but even so it is a simple starting point that is easy for most developers to grasp, and even if it cannot be considered exhaustive, it is useful to create a baseline for analysis. To quote Adam Shostack, "STRIDE is a good framework, [but a] bad taxonomy" [13].

F. Tool Support

Microsoft has made the Microsoft Threat Modeling Tool (TMT) freely available[2], focusing on drawing Data Flow Diagrams and applying STRIDE to the result. Although TMT strictly speaking is not intended for Architectural Analysis as defined by McGraw [7], it works well for drawing Data Flow Diagrams that fulfill the objective of having a "Forest-level view".

[2]https://www.microsoft.com/en-us/download/details.aspx?id=49168

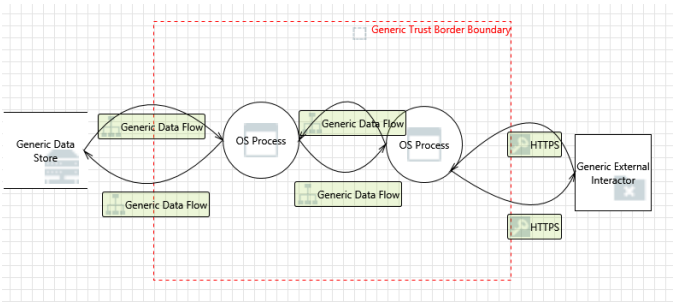

Fig. 3. Forest-level view with TMT

TMT also has other automatic features to aid threat analysis, but the few industry actors we have identified that use TMT have made little or no use of these features, stating that "it is too complicated". This implies that most people seem to be using TMT as a drawing tool only.

It is not surprising that the Building Security In Maturity Model (BSIMM) framework [14] has Architecture Analysis (AA) as a distinct practice (see Table II). The latest edition of the BSIMM report [15] lists the following activities in the Architecture Analysis practice:

- LEVEL 1
  - Perform security feature review.
  - Perform design review for high-risk applications.
  - Have the Software Security Group (SSG) lead design review efforts.
  - Use a risk questionnaire to rank applications.
- LEVEL 2
  - Define and use AA process.
  - Standardize architectural descriptions (including data flow).
- LEVEL 3
  - Have software architects lead design review efforts.
  - Drive analysis results into standard architecture patterns.
  - Make the SSG available as an AA resource or mentor.

III. Beginnings of a Case Study

We are working with a number of software development organizations who use agile development practices, and we find that they generally do not perform any systematic architectural risk analysis activities as part of their software development lifecycle (SDLC). This is also borne out by a previous study of software security maturity among public organizations [16].

However, in agile development, there may be no identified design phase, and even worse: there may not even be a formally defined architecture! Among the Architectural Analysis activities specified in the BSIMM [15], we would like to highlight the following activities:

AA1.1 Perform security feature review
AA1.2 Perform design review for high-risk applications
AA2.1 Define and use an Architectural Analysis process
AA2.3 Standardize architectural descriptions (including data flow)
AA3.2 Drive analysis results into standard architecture patterns

We find that it is usually easy to argue for AA1.1 and AA1.2, as security features "obviously" need to be carefully scrutinized, and the same goes for high-risk applications. As mentioned above, this does not mean that this is codified beyond "taking an extra look at it". The latest BSIMM report [15] seems to confirm that Architecture Analysis is an esoteric art, as only 12.5% of the surveyed organizations confirm that they have defined an Architectural Analysis process for security that they actually use. This implies that even when architectural analysis is performed, it is usually ad-hoc and unstructured.

A major challenge is how to document the outcome of the threat modeling. One of the companies we work with admitted that although they found using the TMT to draw the data flow diagram practical as a prelude to running through the STRIDE categories, they made no attempt to store the created diagrams for later use. Issues that were identified during the process were added to the backlog (e.g., Jira), but only actionable issues, which carry the risk of leaving out issues that somehow are worrying, but need to be revisited at some later time.

However, even when the outcomes are not documented beyond concrete issues that immediately end up in the backlog, performing an architectural risk analysis process will still have intrinsic value to the developers involved, as it will contribute to general security awareness. Much like agile development in general, this will work well as long as there is a relatively stable team of competent developers [17], but might leave security gaps in teams with inexperienced members and/or high turnover.

Other companies state that they do add annotated data flow diagrams to their documentation systems (e.g., Confluence), but we have been unable to find evidence that these are actually used in subsequent analysis efforts.

IV. Discussion

Among the various steps involved in architecture analysis, ambiguity analysis seems to be particularly difficult to accomplish without security experts. It is therefore tempting to highlight this as an activity that is likely to be outsourced to a consultancy firm, or dropped altogether. Another question is whether this activity is possible without architectural documentation, or other documentation. This might mean that it would be necessary for the development team to spend extra time explaining the architecture to the external security expert(s), who in turn would have to document the architecture based on the presentation by the team.

One option might be to split the ambiguity analysis between the security expert (when there is only one) and

TABLE II
The BSIMM Software Security Framework

Governance	Intelligence	SSDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

the team. The expert would then do one independent analysis, and present it to the team to see if they agree. However, there have been no documented cases of this mode of operation, and it is uncertain whether this would be sufficient to catch ambiguities, since the developers in this case do not perform a detailed analysis by themselves.

We have found [18] that threat modeling activities become difficult when development teams are distributed across different geographical locations. However, when it comes to ambiguity analysis, the geographical separation may actually work to your advantage, since it creates a natural division in two (or more) groups. It could be argued that with distributed teams, ambiguity analysis becomes even more important, since it otherwise might be more difficult to catch misconceptions. There is little or no difference between the attack resistance analysis and plain threat modeling as advocated by Howard and Lipner [5], and thus ambiguity analysis seems to be the main added value of "Architectural Risk Analysis" compared to "Threat Modeling".

Gary McGraw [7] stated that one of the three pillars of Software Security is "Knowledge". In previous work [19] we have found that Software Security knowledge is correlated with years of experience. This highlights the fact that up to this day, developers in general have not received instruction in Software Security as part of their education. This clearly needs to change, and one of the specific skills that need to be taught is threat modeling or architectural risk analysis.

V. Conclusion and Further Work

Architectural analysis is challenging in an agile development process, and more work is needed to identify and tailor an architecture analysis process that can align with agile development. One particular challenge is how to integrate architectural risk analysis in DevOps/Continuous Integration/Continuous Deployment pipelines, since threat modeling and related activities are not easily automated. We are currently recruiting more development organizations to contribute to this goal, and through studies of what works (and doesn't work) in practice, we hope to move the state-of-the-art one step in the right direction.

Acknowledgment

The research in this paper has been supported by the Norwegian Research Council through the project SoS-Agile, grant number 247678.

References

[1] M. G. Jaatun, D. S. Cruzes, and J. Luna, "DevOps for better software security in the cloud," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, ser. ARES '17. New York, NY, USA: ACM, 2017, pp. 69:1–69:6. [Online]. Available: http://jaatun.no/papers/2017/secdevops-author.pdf

[2] G. McGraw, "Software security," *Security & Privacy, IEEE*, vol. 2, no. 2, pp. 80–83, Mar 2004.

[3] H. Erdogmus, "Architecture meets agility," *IEEE Software*, vol. 26, no. 5, pp. 2–4, Sept 2009.

[4] P. Abrahamsson, M. A. Babar, and P. Kruchten, "Agility and architecture: Can they coexist?" *IEEE Software*, vol. 27, no. 2, pp. 16–22, March 2010.

[5] M. Howard and S. Lipner, *The Security Development Lifecycle*. Microsoft Press, 2006. [Online]. Available: https://aka.ms/SDL/PDF

[6] Microsoft, "Security development lifecycle for agile development, version 1.0," Tech. Rep., June 30 2009. [Online]. Available: https://www.blackhat.com/presentations/bh-dc-10/Sullivan_Bryan/BlackHat-DC-2010-Sullivan-SDL-Agile-wp.pdf

[7] G. McGraw, *Software Security: Building Security In*. Addison-Wesley, 2006.

[8] F. Swiderski and W. Snyder, *Threat Modeling*. Microsoft Press, 2004.

[9] M. G. Jaatun, K. Bernsmed, D. S. Cruzes, and I. A. Tøndel, "Threat modeling in agile software development," in *Exploring Security in Software Architecture and Design*, M. Felderer and R. Scandariato, Eds. IGI Global, 2019.

[10] J. Ryoo, R. Kazman, and P. Anand, "Architectural analysis for security," *IEEE Security Privacy*, vol. 13, no. 6, pp. 52–59, Nov 2015.

[11] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems (Cat. No. 98EX193)*. IEEE, 1998, pp. 68–78.

[12] S. T. Halkidis, N. Tsantalis, A. Chatzigeorgiou, and G. Stephanides, "Architectural risk analysis of software systems based on security patterns," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 3, pp. 129–142, July 2008.

[13] A. Shostack, "Elevation of privilege: Drawing developers into threat modeling," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*, 2014.

[14] L. Williams, G. McGraw, and S. Migues, "Engineering security vulnerability prevention, detection, and response," *IEEE Software*, vol. 35, no. 5, pp. 76–80, 2018.

[15] G. McGraw, S. Migues, and J. West, "Building Security In Maturity Model (BSIMM 9)," 2018, https://www.bsimm.com/content/dam/bsimm/reports/bsimm9.pdf.

[16] M. G. Jaatun, D. S. Cruzes, K. Bernsmed, I. A. Tøndel, and L. Røstad, "Software security maturity in public organisations," in *Information Security*, ser. Lecture Notes in Computer Science, J. Lopez and C. J. Mitchell, Eds. Springer International Publishing, 2015, vol. 9290, pp. 120–138. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-23318-5_7

[17] T. Dybå and T. Dingsøyr, "What do we know about agile software development?" *IEEE software*, vol. 26, no. 5, pp. 6–9, 2009.

[18] D. S. Cruzes, M. G. Jaatun, K. Bernsmed, and I. A. Tøndel, "Challenges and experiences with applying microsoft threat modeling in agile development projects," in *Proc. 25th Australasian Software Engineering Conference (ASWEC)*, Adelaide, Australia, Nov. 2018. [Online]. Available: http://jaatun.no/papers/2018/Threat_modeling-aswec-2018_final.pdf

[19] T. D. Oyetoyan, M. G. Jaatun, and D. S. Cruzes, "A lightweight measurement of software security skills, usage and training needs in agile teams," *International Journal of Secure Software Engineering*, vol. 8, no. 1, pp. 1–27, 2017.