Product Security Management in Agile Product Management

Antti Vähä-Sipilä*

Nokia
antti.vaha-sipila@nokia.com, avs@iki.fi

Abstract. This paper provides a model for product security risk management and security requirements elicitation in an agile product management framework, using the concepts of Scrum and an epics-based agile requirements model. The paper documents some real-life experiences of rolling out such a risk management model. The model addresses security threat analysis and risk acceptance, and is agnostic to the actual security engineering practices employed in the Scrum teams, and is scalable over large and small enterprises. **This paper has been submitted to a conference and is embargoed until further notice. Please do not copy or forward this paper.**

1 Background

1.1 Risk management in software security

Security risks are a subcategory of business risks. Often, security-related business risks are expressed in terms of lost sales, clean-up costs, incident handling costs and other items that signal loss of revenue. In rare cases, security could actually be seen as a business enabler, and in those cases, the risks could even be articulated in terms of effect on the marketability of the product. McGraw [7] cites risk management as the first pillar of software security.

Security-related business risks should, therefore, be fed into the overall business risk management processes and practices of a commercial enterprise. When using agile process models, risk management should also be agile, in order not to destroy the lean properties of the agile models.

First, business people do not always (and in many cases, should not need to) understand the intricasies of technology. The technical impact of a design-related security risk may be obvious to the engineer, but it may be hard to explain in terms of lost revenue, and the probability of that risk actually being realised is usually subject to much handwaving and guesswork.

Second, security risks should be discussed on all layers of software engineering – from requirements, through design, down to implementation. Thinking about security risks on the high level yields different kinds of results than code-level risk assessment. Both are, however, needed.

* This paper builds on ideas gleaned from many discussions with various people over the time. I wish to give special thanks to Martin von Weissenberg, Vasco Duarte, Heikki Mäki and Lauri Paatero, who I have been able to use as sounding boards.

1.2 Target of this paper

This paper provides a model for product security risk management and security requirements elicitation that can be used in a software development process that builds on Scrum [15] as its core, an agile requirements model (for example, [6] and [11]) based on high-level business case descriptions called *epics*, and system integration in the form of agile release trains [5, ch. 18]. Parts of it can, however, be used in a software development organisation using any of these management frameworks – indeed, the first version of this model was only applied to the Scrum core.

Additionally, a target of this model is to work from the 'ideologically pure' agile principles. I have yet to meet an organisation that would apply agile methods exactly as specified in the literature. There are always impurities when applying agility into real-life organisations. However, it is much easier to adapt a model into an applied agile setup if the source is defined in terms of 'ideologically pure' agile models.

Finally, one of the targets of this model was not to create any new classes of artifacts or process activities that would not already be present in the existing models. This is because tools used by companies are geared towards standard agile frameworks, and we are not in a position to dictate requirements to those tools. Also, activities tend to get optimised away. If security is the only driver for an activity, there is a strong tendency to see that as extra work, and an incentive to optimise it away. Hence, this model aims to reuse agile activities that have already been defined elsewhere and therefore have other reasons to exist.

This paper does not address the actual engineering aspects of secure software development (for example, secure design, secure coding, and security testing). There is a considerable amount of literature available on these issues, and there is no reason why those practices could not be used within a Scrum team for their development work. However, as we see later, introducing the agile practices of test-driven development (TDD) and continuous integration (CI) fit this model very well.

The responsibility for successful agile security risk management falls mostly on Scrum teams and Product Owners. The Scrum team is the focal point of technical threat analysis, whereas the product management needs to understand business-level threat analysis and is responsible for providing aggregate risk information to the customer.

1.3 Previous work on this area

Software security work in an agile setting has been mostly approached from the engineering point of view. Microsoft, who have produced a significant amount of material on large-scale software security processes, have a model for security engineering in Scrum sprints in [19] and [9]. As an another example, [17] discussed security engineering practices in agile development methods.

From the security management point of view, [12] argues that most principles that make a process agile are actually security best practices as well. [6] sets out

some thoughts on managing non-functional requirements in an agile requirements model, which is one of the underlying agile practices this paper builds on. [20] suggests using stakeholder stories to capture security requirements, a strategy which is in line with this paper, as well as advocating the use of test-driven development.

2 Model description

2.1 The risk management cycle

This model introduces a security risk management cycle on two layers in the agile process model.

The *security risk management cycle* is the lifecycle of a security-related business risk. It has the following steps:

1. Finding threats through security threat analysis and privacy impact assessment.
2. Determining how the risks resulting from these threats will be controlled, or whether they will be controlled at all.
3. Reviewing and approving of the residual risk, if any.

A key activity in product security management is security threat discovery. Academically, the difference between a threat and a risk is that a threat can exist without it being probable at all. If, on the other hand, one can identify a vulnerability, a plausible exploitation vector, or a real-life probability for the threat to materialise, we clearly have a risk. Risks have a probability and an impact, which, for our purposes, is understood as a negative business (that is, monetary) impact.

Threat analysis, therefore, aims to unearth threats that might turn out to be security-related business risks.

A risk is controlled by introducing new requirements, which can be functional or non-functional. Functional requirements are typically product features – in this case, often security features – and non-functional requirements are typically qualities and quality assurance needs, such as testing needs, or how things should be done. Risk can also be terminated by actually removing potentially vulnerable functionality. This would have an effect of removing certain requirements altogether.

For each threat that could plausibly give rise to a security-related business risk, a decision has to be made: control the risk or accept the risk. Residual risk approval is a business decision, where the potential cost of realisation of uncontrolled risk is compared to the cost of controlling the risk. There can be no option to ignore a risk (as that would equal to taking an unknown business risk).

The model is described through mapping this cycle on three agile methods:

1. Scrum sprints [15] and backlog grooming, also known as backlog maintenance or refinement ([3], [11], [4])

2. Feature decomposition as described in the agile requirements model ([6] and [11])
3. The agile release train [5, ch. 18]

2.2 Real-life experiences of applying the model

The development of this model started with the Scrum level activities. In the original model, threat analysis was suggested to be done during Sprint planning and as an extra sprint activity, but many teams were quick to suggest that a more natural place for it would be in backlog grooming phase.

The initial pilot of this model only applied the practices in Scrum sprints. Teams were asked to nominate a security person, and a qualitative survey was conducted by interviewing these security persons of seven Scrum teams, plus two senior software developers of one of these teams (total of nine individuals). The teams had had a possibility to employ these practices for a time ranging from three to six months before the survey. Threat analysis was done largely in a freeform, 'brainstorming' fashion, using data flow analysis to structure the discussion.

The following findings were mentioned in some form or another by the majority of these teams:

- Conducting a threat analysis on Scrum level was felt as an useful activity that yielded tangible results. The lightweight nature was quoted as a benefit of the model. (6 teams)
- Threat analysis should be triggered by the introduction of new requirements (as opposed to an activity that would take place in every sprint, or be triggered by a milestone or gate, or not to have specific triggers at all). The analysis should support requirements elicitation and should be started on higher level requirements (as opposed to only analysing backlog items). (6 teams)
- People who do threat analysis should be supplied with practical support material which should be customised to the specific needs, and which would preferably be templated. (5 teams)
- Nominating a single person as a security person works. The foreseen benefits included scalability as the number of teams grows and having someone specifically to keep this area in mind. (4 teams)
- Security professionals should be made available as internal consultants. Access to these security experts should be as easy as possible. (4 teams)

These findings have been taken into account in the description of the new model. Further observations on taking the model into use are described in section 3.

2.3 Risk management in Scrum sprints and backlog grooming

Threat analysis and selection of controls in Scrum sprints should be triggered by new or changed functionality introduced by the product backlog items. Threat

analysis can happen implicitly as a part of design, or preferably as an explicit activity during backlog grooming. Reviewing the risks and approving the residual risk should always happen during the Sprint review (see figure 1).

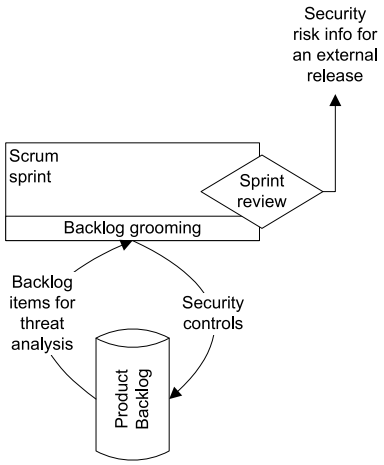

Fig. 1. Elements of security risk management in a Scrum sprint. Threat analysis happens in backlog grooming and looks at the whole Product Backlog (that is, not just the Sprint Backlog). Threats are mitigated by adding new Product Backlog items. The Sprint Review determines whether the sprint is *done* from the security perspective, and provides residual risk information to the customer (via the Product Owner).

Threat analysis. Backlog grooming, also known as backlog maintenance or refinement, is an ongoing activity where Product Backlog items are redefined and massaged into better and more digestable form. Unlike Sprint planning, backlog grooming does not concentrate on a specific sprint. This makes it possible to plan security controls over several sprints.

Backlog grooming can take a considerable chunk of the Scrum team's time: for example, [14, p. 114] and [3] propose using ten per cent of Scrum team's time for grooming. That should leave more than ample time to do proper threat analysis.

On the Scrum team level, threat analysis itself is usually a technical security engineering exercise, and the vulnerabilities or exploitation vectors may be very detailed. Typically, the controls which the Scrum team decides to implement are functional requirements, meaning security features or design decisions. Such controls can be readily expressed as backlog items, and those should be pushed to the Product Backlog. The Product Backlog is constantly being prioritised by the Product Owner, which results in the newly discovered controls being prioritised by the business owner, thus effectively escalating risk management decisions quickly.

However, sometimes the team may feel that they should create a higher-level requirement or push a backlog item to some other team's Product Backlog. The requirements management processes should allow this to happen, as this is an example of the Scrum team actually breaking out of its box and taking a larger, end-to-end view, which is always commendable.

Non-functional controls. Some controls arising from the threat analysis can of course also be non-functional. Non-functional requirements are typically system qualities (such as robustness, performance, or usability) or quality assurance targets. Usually, non-functional requirements are long-lived and need to be checked against regression, and unlike with security features or functionality, you cannot say that you're finished implementing them. This makes them tricky to implement as backlog items, which are intended to be consumed and ticked off the list.

In the literature, it has been proposed that non-functional requirements would be expressed as constraints (for example, [11] and [6]). Constraints are statements of the system which are either true or untrue. Security, being a subset of quality, often has things which cannot be easily or meaningfully expressed in functional terms. Cohn [1] proposes treating constraints as a sort of metadata ("constraint cards") to backlog items, where they would act as reminders of "how" to implement a backlog item. However, many constraints are completely cross-cutting, meaning that they have an effect on the complete system and all of its features, so effectively they would easily end up as metadata for the whole project.

As backlog items are intended to get consumed and done with, polluting backlogs with static constraints, or burdening individual backlog items with metadata, are not ideal solutions. [1, ch. 7 and 16] proposes transforming constraints into acceptance test cases, which indeed is the end state where we want them to be (see also figure 2). [20] also takes this view specifically for security controls, which should create negative 'assurance' tests.

In order to do this, this model calls for a backlog item which is about acceptance test case creation. In security, these test cases are often negative tests. Therefore, the steps to handle a non-functional security requirement could be:

- Decompose the non-functional requirement into measurable constraints. For example, a requirement to be robust against malformed inputs can be expressed as a constraint to pass 100.000 semi-random fuzz test cases.
- Add a backlog item for each constraint that calls for the addition of this test case in the (hopefully automated) test environment. In this example, the backlog item would read "create a fuzz test case set and add that to automated module test framework".

Whether or not the tests are created out of functional or non-functional requirements, test-driven development (TDD) offers a very useful agile tool to ensure that these constraints are being met both now and in the future. The security test cases can be automatically run in continuous integration.

Residual risk approval. When a product increment (backlog item, sprint, or a product release) fulfills agreed quality criteria, it is *done*. [13] Whether or not Sprint Backlog items are *done* is determined in the Sprint review.

Backlog items that are *done* should be potentially shippable, meaning that they should be mature enough from quality standpoint to be sold, taken into use, or put into operation. [10] Of course, the system might not be actually commercially viable or realistically usable given the feature set implemented this far, but whatever has been implemented should be of sufficient quality. From security perspective, an increment is *done* if the residual security-related business risk of the product is acceptable.

The Scrum team typically has a set of guidelines to follow when determining whether something is *done*. These guidelines are called the *definition of done* (for example, [18]). The definition of done can be utilised to create some of the paper trail required for security assurance, if necessary. At a minimum, in the sprint review, the team should ask itself whether "given the threats we have found, and controls we have implemented this far, can we say that we are *done* from the security perspective".

2.4 Risk management in the agile requirements model

In the agile requirements model ([6] and [11]), system requirements are presented on various abstraction levels: *epics*, *features*, *user stories*, and *backlog items*. Higher-level abstractions are *decomposed* into smaller pieces, until they become backlog items, and get consumed by Scrum sprints.

Threat analysis should be viewed as a security requirements elicitation activity, and should therefore made an integral part of this decomposition process. Residual risk approval also happens during this decomposition process as a product requirements management activity.

As with the threat analysis on the Scrum sprint level, this model does not mandate any specific threat analysis methodology. If you are using use case based descriptions, an useful tool for eliciting security requirements would be *misuse cases* [16]. The activity should slant towards requirements elicitation, for example as follows (see also figure 2):

- On the epics level, threats could be formulated as the users' security or privacy needs grounded in user psychology and user perceived loss of value. A Privacy Impact Assessment (PIA) could be done already at this level. Mitigation of potential risk could be expressed, for example, as feature-level misuse cases.
- On the feature level, threat analysis could be slightly more technical in nature. Depending on the threat, control targets could create positive technical requirements, non-functional requirements, or story-level misuse cases.
- On the story level, exact implementation choices could be spelled out, and controls should be expressed mainly in terms of non-functional requirements or positive technical controls.

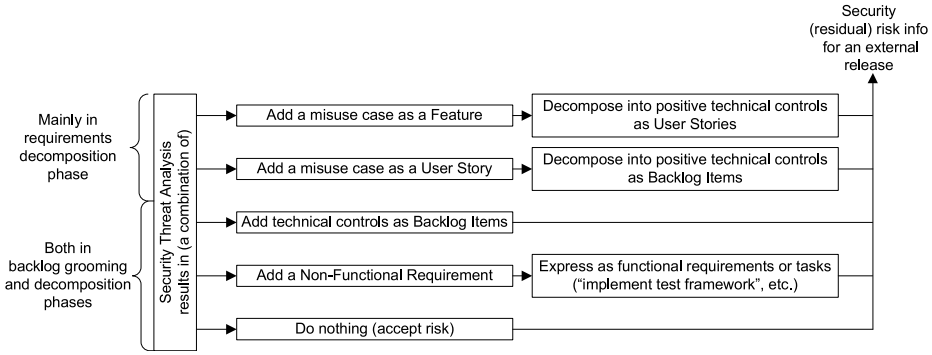

Fig. 2. The results of security threat analysis should end up as requirements or tasks, or as an explicit risk acceptance decision. Requirements expressed as misuse cases or non-functional requirements need to be expressed as positive functional requirements or tasks.

As soon as architectural choices get made, analysis should utilise those as input, for example for creating a non-functional requirement for robustness testing of interfaces.

Typically, the people involved in requirements decomposition should be involved in the product-level business decisions and they should have a complete picture of the product's market positioning. The participants' profile probably also steers the nature of this threat analysis to the direction of being more freeform, although [17] proposes a formal model for a threat analysis on this level.

2.5 Risk management in the agile release train

When several Scrum teams deliver components of a larger system, the resulting integration schedule is sometimes called the *release train*. Components are integrated after each sprint, preferably in a synchronised schedule. At set times, an internal or external release is produced. [5, ch. 18] These internal or external releases can also be used as 'milestones' or 'gates', and in an enterprise setting, should act as checkpoints for security risk status.

With regard to the risk management cycle, this stage is only about risk review and acceptance. The checkpoint should check whether risk analysis has in fact been done at any preceding stage, but now it would be too late.

A vehicle for collecting the risk data could be a *release sprint* (also called a *hardening sprint*, even if it does not necessarily have anything to do with hardening in the security sense). A release sprint is a fixed-length sprint for ensuring product quality, and (for example, in [5, ch. 18]) it precedes each internal and external release. It is not a bug fixing sprint, so the team cannot just push fixes forward on that sprint. [2] During this sprint, information about the identified threats, controlled risks and accepted risks should be collected, and provided to

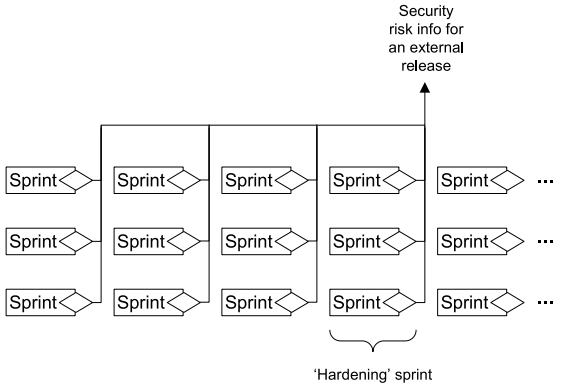

Fig. 3. Security risk information, meaning the knowledge of residual (uncontrolled) risk, should be collected over all Scrum teams. In a hypothetical fully synchronised Agile Release Train, risk information would be made available through the Product Owners for every external release.

whatever body is responsible for approving the product release as a whole (see figure 3).

At this stage, the risk information leaves the agile software development process and is documented in a governance document. What this actually means is specific to enterprise and their compliance requirements. Some companies might need a formal write-off from a high-ranking manager, some others might just archive the risk data to be able to show (in an audit, or in the court of law) that they have followed due diligence and their risk acceptance decisions have been explicit.

3 Model roll-out considerations

3.1 Ensuring team competences

Security engineering practices on Scrum team level enhance software security even without a risk management framework in place. Also, the bulk of the technical threat analysis is performed on the Scrum team level. Adding high-level risk management checkpoints without technical capacity for identifying security issues easily leads to rubber-stamping checklists.

This is why implementation of this risk management framework should start from the Scrum team. From the risk management perspective, the Scrum teams should adopt threat analysis as a standard practice in backlog grooming, and reviewing that sprint's security decisions in the sprint review.

In order to facilitate this, the team needs a trigger. This can be a single person from the team who would act as a security person,[1] responsible for triggering the security work.

These security persons do not need to be security professionals. They should not be single-handedly responsible for actually doing threat analysis, and they should not definitely be the person who gets the blame for any vulnerabilities. Instead, these people should act as the 'security conscience' for the team.

They should, at a minimum, be active at least twice during a sprint. In backlog grooming, they should remind the team to conduct a threat analysis for the most important backlog items. In sprint review, they should ask the question 'From security perspective, can we say that the Sprint Backlog items are *done*?'.

However, nothing prevents such a person from becoming a security professional, possibly taking on further roles such as being the contact point in security incident cases affecting the team's code. Also, these individuals should volunteer for the role, with the idea in mind that someone who is truly interested in security can be relied on acting as the 'security conscience'.

This role should be visible in that person's job description and incentive plans, and for people interested in security, this allows them to grow in this area even if they would be in an otherwise generic, non-security-specific software project.

3.2 Product Owner as the focal point

The Product Owner is in the key position when creating secure products, as they work both with backlog grooming and decomposition, and therefore have a possibility to attend all threat analysis activities. Having said this, the Product Owner should not do threat analysis alone. The teams' security persons should view Product Owners as their allies, with teams' security people 'pushing' their teams to do threat analysis, and Product Owners exerting 'pull'.

If the Product Owner position is diluted, for example, by removing their ability to truly make business risk management decisions, they cannot work effectively using this model. There are different ways in which this dilution could happen – either the Product Owner is not empowered enough, for example, he has no say over team's resource investments, or the Product Owner role could be split between different persons with unclear final say. Useful traits for Product Owners and potential pitfalls are listed, for example, in [11].

3.3 Facilitating threat analyses and creating a baseline

Just simply requiring threat analyses or privacy impact assessments to be conducted will in most cases not produce very high-quality results. Security experts need to support the first (few) threat analyses.

[1] The Building Software In Maturity Model (BSIMM) [8] terms these persons collectively as the *satellite*, persons who 'show an above-average level of security interest or skill', and advocates cultivating their social networks.

The first analysis is usually a retrospective threat analysis with a target to create a baseline. This first threat analysis would not care which backlog items are the highest priority just now – the emphasis would be on reviewing the codebase and feature set that has been implemented up to now. The future threat analyses on new backlog items and requirements would then be incremental on top of this baseline.

In addition to giving a baseline and context for future analyses, this extended threat analysis also works well as a training opportunity. The target is to, over time, empower the Scrum team and make it truly autonomous.

3.4 Suitability for other product qualities, and caveat emptor

It is very likely that this management model is a good fit for other non-functional product qualities, such as performance and robustness.

However, using this model for managing safety (for example, mechanical or electric safety of products) should be extremely carefully thought out, because those qualities are more strictly regulated and the risks also extend to risks to health and environment. In addition, certain industries such as aerospace and healthcare may have many security-related issues that actually have safety aspects. Even though this model does help in finding security issues that have a safety dimension, follow-up of those threats may need to be more formal than presented here.

References

1. Mike Cohn. *User Stories Applied: For Agile Software Development.* Addison-Wesley Professional, 2004.
2. Mike Cohn. Correct Use of a Release Sprint. http://blog.mountaingoatsoftware.com/correct-use-of-a-release-sprint, June 2007. Retrieved 2009-09-29.
3. Mike Cohn. *Succeeding with Agile: Software Development Using Scrum.* Addison-Wesley Professional, a 'Rough Cuts' 2009-11-13 draft edition, 2009.
4. Dan R. Greening. Backlog Grooming: Scrum Pattern. http://knol.google.com/k/dan-r-greening/backlog-grooming/, August 2008. Retrieved 2009-09-10.
5. Dean Leffingwell. *Scaling Software Agility: Best Practices for Large Enterprises.* Addison-Wesley Professional, 2007.
6. Dean Leffingwell and Juha-Markus Aalto. A Lean and Scalable Requirements Model for Large Enterprises. http://scalingsoftwareagility.files.wordpress.com/2007/03/a-lean-and-scalable-requirements-information-model-for-agile-enterprises-pdf.pdf, 2009. Retrieved 2009-09-29.
7. Gary McGraw. *Software Security: Building Security In.* Addison-Wesley Professional, 2006.
8. Gary McGraw, Brian Chess, and Sammy Migues. Building Security In Maturity Model. http://www.bsi-mm.com/, 2009. Retrieved 2009-09-29.

9. Microsoft. Microsoft Security Development Lifecycle. http://www.microsoft.com/sdl, 2009. Version 4.1a. Retrieved 2009-11-16.
10. Dhaval Panchal. What is Definition of Done (DoD)? http://www.scrumalliance.org/articles/105-what-is-definition-of-done-dod, September 2008. Retrieved 2009-09-15.
11. Roman Pichler. *Agile Product Management with Scrum: Creating Products that Customers Love*. Addison-Wesley Professional, a 'Rough Cuts' 2009-09-01 draft edition, 2010.
12. Reijo Savola. Agile Information Security Management in Software R&D. https://www.owasp.org/images/c/cd/RWSUG5_Agile_Security_Management.pdf, 2008. Retrieved 2009-09-29.
13. Ken Schwaber. *Agile Project Management with Scrum*. Microsoft Press, 2004.
14. Ken Schwaber. *The Enterprise and Scrum*. Microsoft Press, 2007.
15. Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice-Hall, 2001.
16. Guttorm Sindre and Andreas L. Opdahl. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, 2005.
17. Mikko Siponen, Richard Baskerville, and Tapio Kuivalainen. Integrating security into agile development methods. In *Proceedings of the 38th Hawaii International Conference on System Sciences*, 2005.
18. Chris Sterling. Building a Definition of Done. http://chrissterling.gettingagile.com/2007/10/05/building-a-definition-of-done/, October 2007. Retrieved 2009-09-15.
19. Bryan Sullivan. Streamline security practices for agile development. *MSDN Magazine*, November 2008.
20. Dave Wichers. Breaking the Waterfall Mindset of the Security Industry. http://www.owasp.org/images/b/b8/AppSecEU08-Agile_and_Secure.ppt, 2008. Retrieved 2009-09-29.