Cloud Software Finland

HANDBOOK OF

The Secure Agile Software Development Life Cycle

This work was supported by TEKES as part of the
Cloud Software Program of DIGILE (Finnish Strategic
Centre for Science, Technology and Innovation in the
field of ICT and digital business).

Handbook of the Secure Agile Software Development Life Cycle

Publisher: University of Oulu, 2014
Editors: Pekka Pietikäinen, Juha Röning
Authors: Jouko Ahola, Christian Frühwirth, Marko Helenius, Lea Kutvonen, Juho Myllylahti,
Timo Nyberg, Ari Pietikäinen, Pekka Pietikäinen, Juha Röning, Sini Ruohomaa, Camillo Särs,
Tuuli Siiskonen, Antti Vähä-Sipilä, Ville Ylimannela

ISBN number: 978-952-62-0341-6

Layout: Paikallinen Mainostoimisto

Juvenes Print Oulu, 2014

Contents

Foreword

> "The Cloud Software program (2010-2013) aims to significantly improve the competitive position of Finnish software intensive industry in global markets. According to the 2009 survey most significant factors of competitiveness are:operational efficiency, user experience, web software, open systems, security engineering and sustainable development. Cloud software ties these factors together as software increasingly moves to the web. Cloud Software program especially aims to pioneer in building new cloud business models, lean software enterprise model and open cloud software infrastructure."
> **– Janne Järvinen, Focus Area Director**

Software quality problems, wide impact vulnerabilities, phishing, botnets and criminal enterprise have proven that software and system security is not just an add-on despite past focus of the security industry.

Cloud computing introduces a whole ecosystem of clients, services and infrastructure, where trust boundaries are moved even further into components, where physical location or even ownership is unknown. Add-on security therefore becomes more futile than it ever was. There is no place where these add-on components would reside.

Security, trust, dependability and privacy are issues that have to be considered over the whole life-cycle of the system and software development from gathering requirements to deploying the system in practice. Doing this does not only make us safer and secure but improves overall system quality and development efficiency.

In the past few years, several initiatives have surfaced to address security in the software development lifecycle. These include prescriptive models from companies, such as Microsoft Security Development Lifecycle (SDL), descriptive activity surveys such as the Building Security In Maturity Model (BSIMM), and even standards, such as the ISO/IEC 27034. Building a mature software security initiative may be expensive. Smaller software vendors, specifically small and medium enterprises, may not afford to have dedicated resources for their own security initiatives. However, they still need to compete against the larger players.

Many of recent security initiatives have been relatively open and can be leveraged to help the Finnish Industry and to initiate new business. Finland has pioneered research in Security Metrics, Vulnerability, Managing Complexity, Security as a Quality Aspect and Software Robustness areas. This research can therefore be applied directly to be a part of new, improved SDLs.

There is a desire to improve software and system development life-cycle efficiency so those efforts can drive security and security can support them. Secure Development Lifecycles in Cloud Services require a change of mindset from individual devices or pieces of software, to complex systems, such as Cloud Services, consisting of numerous software components, as well as infrastructure, all of which are all developed with varying development life-cycles, and are procured from a variety of sources (e.g., subcontractors and open source for software and, e.g., Amazon EC2 and private clouds for infrastructure). These are then integrated and verified (internally, or using external auditors), and finally deployed.

Ecosystems should be recognized and supported since the secure software development lifecycle is not isolated to the conventional vendors but affects post deployment end-users, 3rd party developers and e.g. carrier partners.

Chapter contents

This book brings together experiences and results from security research done in liaison by the authors during the Cloud Software Program, each bringing in their own viewpoint. The chapters are standalone articles, which can be read in any order. With this book we hope to communicate forward the explicit and tacit knowledge we have accumulated with the hopes that readers of the book (such as you) might gain something from our experiences and in result make the kingdom of clouds a bit safer and trustworthier place for the common good of us all.

Chapter 2: Generic Security User Stories
Tuuli Siiskonen, Camillo Särs and Antti Vähä-Sipilä and Ari Pietikäinen introduce Generic Security User stories, which are a tool for introducing security requirements into software projects, where the project organisation may not have access to the services for a full-time security professional.

Chapter 3: Security in Agile Product Management
Antti Vähä-Sipilä discusses how software security engineering practices can be mapped to various agile product management concepts. The basic tenet is that work needs to be made visible, and as the main vehicle for that is the product backlog, the article concentrates on how software security work can be put on the backlog.

Chapter 4: Security activities in scrum control points
Ville Ylimannela and Marko Helenius describe how to map important control points in Scrum and decide what security activities should be performed in the control points. The security activities discussed are: security templates, threat assessment, feature flagging and residual risk approval. The criteria used to judge the activities are: documentation, customer interaction, speed of execution, simplicity and security.

Chapter 5: Risk management
Ville Ylimannela and Marko Helenius outline a method for identifying and managing security risks in an agile software project utilizing risk boards.

Chapter 6: First Steps to Consider Privacy
Ari Pietikäinen and Jouko Ahola discuss privacy issues and personal database protection, analysing risks related to them,as well as the main initial steps any enterprise shall take to manage privacy issuea.

Chapter 7: Security Metrics
Reijo Savola, Christian Frühwirth, Ari Pietikäinen and Timo Nyberg provide a a simple security, privacy and trust metrics development approach and a method for aligning security metrics with security objectives

Chapter 8: Fuzzing
Juho Myllylahti and Pekka Pietikäinen write about fuzzing. Fuzzing is an effective and practical form of fault-based testing which is especially popular in security and robustness testing. Fuzzing tools widely used by security researches and other parties who try to uncover vulnerabilities in widely used real world software. The chapter presents the general idea,
terminology and tries to give insight on how to combine fuzzing and agile development practices.

Chapter 9: Dynamic Trust Management
Sini Ruohomaa and Lea Kutvonen take a look at dynamic trust management, which provides for enhanced security during operational time when a composition of software services run together forming a system that spans over organizational boundaries, in a shared cloud or between clouds.

Appendix 1: Generic Security User Story templates
The full set of Generic Security User Stories, introduced in Chapter 2.

Generic Security User Stories

Concepts section:
Tuuli Siiskonen, F-Secure
Camillo Särs, F-Secure
Antti Vähä-Sipilä, F-Secure

Experiences section:
Ari Pietikäinen, Ericsson

Executive summary

Smaller organisations and software development groups may not have access to dedicated services of a security professional. If the organisation or a group is using a product backlog to drive their software development work, a set of ready-made backlog items (requirements and task descriptions) may be helpful. Adding these ready-made backlog items on the product backlog is a bit like building a checklist into the requirements.

We present a set of backlog items in the form of "user stories". A "user story" is a requirement with a specific format, offering both the value proposition and the next steps. The aim is that the backlog items should fit in with minimal editing.

Concepts

User Stories

A "user story" is a format for capturing what a user needs to do, and also describes the value that the user would get from a specific functionality, often used in agile product management. Here, we use the term "user story" to denote the format in which the needs are written down, not the size of the work or feature that implementing it involves. Organisations that have comprehensive agile product management pipelines often use different names for different levels, or sizes, of stories: large ones may be known as epics and smaller ones features, user stories, and tasks. A good description of this sort of story-based structure can be found both in the paper by Leffingwell and Aalto, as well as the book by Leffingwell, both of which can be found in the references of this article.

A user story, as a format, is suitable for each of these levels. Many of these user stories would be what many companies would call features.

The stories in this list are subdivided into several parts:

Part of a User Story	An example
The stakeholder and the value proposition have been documented for each story so that product owners can reach out to these individuals to get their views on the relative importance of that user story. In many of the stories, the stakeholder title reflects a company who creates hosted "cloud" services. You may need to adjust the actors so that they correspond to your organisation's roles. Typically the stakeholder and value proposition are described as "as <someone>, I want <something> so that <value proposition>".	"As a user I want dancing hamsters on the screen so that I can become a better person."
A longer explanation of the story opens up the rationale and background of the story, and also explains some of the terminology. This explanation should be accessible to any software generalist.	"Dancing furry animals are a well-known method for stress relief, and stress is one of the leading blockers for persons on their journey to enlightenment. We provide an allergen-free access to furry dancing animals."
Each user story has a set of **acceptance criteria** that need to be met for the user story to be deemed to be complete, or "done". These acceptance criteria are always measurable, and often binary: you can say "yes" or "no" for each of them. Typically the criteria are either tests that would be written, or establishing that a document exists. In any case, after the acceptance criteria have been satisfied, the user story can be closed. Any tests that have been written continue to be run as a set of module or system tests, and should they later fail, it shows up as a bug. If you have quality criteria that apply to each and every task, those should probably be driven through more common "definition of done".	"Test that the product displays at least three dancing hamsters concurrently with an fps of greater than 25."
There is a set of refinement (a.k.a. "grooming") questions for every user story. These are questions which the product owner or the developers are encouraged to ask from themselves in order to discover the actual work that needs to be done for the user story. These questions can be discussed when splitting the user stories into smaller tasks, for example, in sprint planning, or a separate "backlog refinement" session. If you feel unsure as to where to start, the refinement questions provide way to approach the user story in a productive and practical way. Having refinement questions written down are not always necessary for all audiences, but are helpful when the stories are taken from an external source such as this list.	"How do we ethically source the hamster dance video? (Contact PETA and the American Humane Association for guidance.)"

Why Security User Stories?

User stories are the fuel of an agile team. The stories are fed to the team through an ordered (prioritised) list called the product backlog. The team implements the stories, and once done, the software is expected to provide the value specified by the user story. Security requirements are notoriously hard for customers to require explicitly; security, as an aspect of quality, is often assumed and an implicit requirement. The fact that it indeed is a requirement is often noticed when something goes wrong, and it becomes evident that security should have been required. This is why we have created a set of ready-made user stories that can be used to cover the security requirements, even if the customer lacks the expertise to specifically require them.

User stories also drive work and resource allocation. We believe that all work should be driven by a business requirement, and in agile, this means that a user story should exist. There cannot be some sort of invisible time allocation that would magically produce test cases and documentation outside the user stories. Creating test cases and writing documentation is work. If the team is expected to do this work, it needs to be driven through the backlog. At the same time, the investment in security becomes visible and measurable, and thus manageable.

Also, by scheduling all work through a backlog, product owner have a truthful picture about the project's status and the work that is yet to be done.

There are also other published lists of security user stories, for example, SAFECode's 2012 paper (see references).

Using Generic Security User Stories

Except for very special cases, it is very probable that consistently implementing these user stories will satisfy even the pickiest customers. These user stories reflect the security needs that are commonly found in network-facing services, cloud software, and their clients. They reflect (but do not directly copy) the existing security standards such as the ISF Standard of Good Practice, ISO/IEC 27002, and secure software development lifecycle models such as BSIMM.

Not all user stories apply to all kinds of software. For example, a client on a mobile phone probably does not do audit logging. There is a table which helps you to select the correct user stories that apply in your case. The stories here have a bias towards a cloud-based software system. Embedded software and hardware security might require stories that we do not cover here.

The user stories would preferably be used by the product owners to populate their product backlog when starting a new project, but can, of course, be added to an existing backlog as well.
It is possible that the number of stories feels large. However, the requirements defined here are usually considered to be valid. If your product backlog size will double after addition of these stories, it may be useful to pause and think whether the backlog was complete in the first place with regard to all other quality aspects. Are you expecting the development teams to implement hidden or unstated requirements? Is all the quality-related work, test case creation, and documentation work made visible?

Most acceptance criteria in the security user stories list ask for test cases to be created. They do not exist only for checking the implementation of the user story, but also for the sake of building an argument later for any third party. In this role, a test case is better than documentation. Documentation is just a snapshot in time – a test case evaluates the current status of security every time it is run. For most of the test cases, the intention is that the test would be run in test automation. There are a handful of tests, mainly exploratory security testing, which would be difficult to automate, but the bulk of tests should be run again and again.

Larger security themes

Although all stories listed here are standalone, they can be split into larger categories, or themes. Many of the stories also have a connection to some other quality aspect. This table shows these themes and relationships.

Theme	User Story
Access control and user management	User data model Account lifecycle Access control policy Access control implementation Administration access
Software security activities	Architectural threat analysis Interface robustness Third party security assessment Cryptographic key management
Compliance and preparing for incidents	Availability Business Continuity External security requirements Backups
Notifying users	Accessible Privacy Policy (PP) Notify of T&C and PP changes
Logging	Audit log of security activities Audit log of personal data access Log event separation Protection of log events Standardisation of log events
Managing the software asset	Patching and upgrading Change Management Component inventory and origin
Detection capabilities	Monitoring Enabling intrusion detection
Content management	Malware protection Content takedown Legal interception and access
Operational environment	Platform hardening Network architecture Separate development and production

Story selection matrix

Not every story is applicable to every product. As an example, if the product has no server-side component, many requirements concerning logging may vanish. On the other hand, if no user data is being handled, it is a fairly safe bet that it doesn't need to be protected, either.

In order to help to quickly narrow down the set of user stories that apply to a specific project, the following selection matrix (divided in two parts purely for layout reasons) can be used. By answering each question in the first column, you can find out which stories should be pulled on the Product Backlog, as indicated by the corresponding column.

	User data model	Account Lifecycle	Access control policy	Access control impl.	Arch. threat analysis	Interface robustness	3rd party sec assessment	Availability	Business continuity	External sec requirements	Accessible privacy policy	Notify T&C and PP changes	Audit log of sec activities	Audit log of pers data access	Log event separation
Do you process information about a user?	●			●	●					●	●	●		●	
Do you have data that belongs to or is created by a user?	●			●	●					●	●	●		●	
Do you have a concept of a "user account"?	●	●	●	●									●	●	
Do you offer a "service"?								●	●			●			
Is your system a "product" that has a "customer"?							●			●					
Do you have different user accounts for different reasons?	●		●	●									●		
Do you interface with any other system (locally or remotely)?					●	●									
Do you transfer information over a network?					●	●		●		●					
Is a part of your system "hosted" or "in a cloud"?			●	●	●			●	●	●			●	●	●

	Protection of log events	Standardisation of log events	Patching and upgrading	Change management	Component inventory	Monitoring	Enabling intrusion detection	Malware protection	Content takedown	Legal interception & access	Backups	Crypto key management	Platform hardening	Network architecture	Admin access	Separate dev & prod env
Do you process information about a user?	●						●			●	●	●	●			●
Do you have data that belongs to or is created by a user?	●							●	●	●	●	●	●			●
Do you have a concept of a "user account"?																
Do you offer a "service"?	●		●	●	●	●										
Is your system a "product" that has a "customer"?			●	●	●											
Do you have different user accounts for different reasons?															●	
Do you interface with any other system (locally or remotely)?												●	●			
Do you transfer information over a network?										●		●		●		
Is a part of your system "hosted" or "in a cloud"?	●	●	●	●	●	●	●				●		●	●	●	●

The Generic Security User Stories

Due to its length, the complete list of Generic Security User Stories can be found in an appendix of this book.

Experiences of Using Security User Stories

Many ICT companies have in the past relied on using a relatively static set of generic security baseline requirement as a basis for ensuring that their solutions are capable of providing appropriate security mechanisms to prevent security incidents from taking place. There are some often quoted sources for such requirements, e.g., RFC 3871 and ANSI standard T1.276-2008 (see the references).

When transforming a development organisation from using traditional waterfall methodologies towards an Agile way of working, it can be difficult to ensure that the baseline security requirements will get properly addressed in Agile requirement flow because the individual requirements have no User Story context thus not making their way into the backlog.

Our practical experiences of augmenting the generic security baseline requirements with a selection of Security Epics and Security User Stories have been both positive and illuminating. The main positives:

- Motivation for the requirements is easier to communicate through a User Story

- Several distinct, but associated requirements can be collected into an Epic containing a number of User Stories, bringing clarity to the associations

- The Security User Stories are much easier to attach to the backlog than isolated generic requirements as the requirements now have a specified context

- As the Security User Stories now are contained in the backlog, they get handled in the same manner as any other user stories in each sprint, e.g. they get prioritized, they get considered in "Definition of Done"

- Verification of effectiveness of security controls becomes more accurate

- Security objectives also become clearer as the focus shifts from "a product needing to contain some specific security functionality" to "a product getting protected by the use of the security functionality"

- Security awareness across the development organization(s) increases as development teams get exposed to Security User Stories

As our company still operates amidst waterfall and Agile transformation, we intend to keep both the baseline security requirements and the new agile version of it up to date.

References

SAFECode: Practical Security Stories and Security Tasks for Agile Development Environments, July 2012. http://www.safecode.org/publications/SAFECode_Agile_Dev_Security0712.pdf

Dean Leffingwell and Juha-Markus Aalto: A Lean and Scalable Requirements Information Model for the Agile Enterprise, 2009. http://scalingsoftwareagility.files.wordpress.com/2007/03/a-lean-and-scalable-requirements-information-model-for-agile-enterprises-pdf.pdf

Dean Leffingwell: Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley, 2011.

IETF RFC 3871: Operational Security Requirements for Large Internet Service Provider (ISP) IP Network Infrastructure. http://tools.ietf.org/html/rfc3871

ANSI standard T1.276-2008, (ATIS-0300276.2008[pre-pub]): Operations, Administration, Maintenance, and Provisioning Security Requirements for the Public Telecommunications Network; A Baseline of Security Requirements for the Management Plane. https://www.atis.org/docstore/product.aspx?id=24660

Security in Agile Product Management

Antti Vähä-Sipilä, F-Secure

Executive summary

Agile development practices can be used to create security-sensitive software, and software security practices can be driven through an agile process. This chapter discusses how software security engineering practices can be mapped to various agile product management concepts. The basic tenet is that work needs to be made visible, and as the main vehicle for that is the product backlog, the article concentrates on how software security work can be put on the backlog.

Introduction

This article is a product of a long evolution, tracing its roots to a workshop organised by the Finnish Information Security Association and Agile Finland. Since then, the message has been reiterated a few times, and this article reflects the current view that the author has.

This is by no means unique work. Security in an agile setting has been discussed by others. For example, see the section on "Existing frameworks for risk and security management in agile software development" in the paper on risk management in this book.

A short list of prior work in this area is in order; you can find the pointers to these from the references section. Microsoft's SDL-Agile, a mapping of Microsoft's Security Development Lifecycle to an agile development methodology, suggests dividing security activities into distinct classes that would be executed with a varying cadence. Rohit Sethi's InfoQ article concentrates on security qualities that are not user stories or attacker stories, arguably something that is the hard part - and his article should be required reading in addition to this one. SAFECode's paper on security-related stories has a particularly good section on residual risk.

The viewpoint of this article is decidedly in the domain of product management. The argument goes that unless an activity or a requirement is visible and gets time allocated to it, it is unlikely to happen. Also, development methods that are often dubbed "agile" such as test-driven development and extreme programming, can be used in non-agile product management setups as well, and hence their inherent security properties do not necessarily have anything unique to agile.

Concepts

What Are Software Security Activities?

Finding inspiration for software security activities is easy. Figuring out exactly which of these to drive is trickier. Various sources are brimming with activities that either someone does, wants to do, or tells you to do. Some studies, such as BSIMM by Cigital, a consultancy, are descriptive in nature: BSIMM's 2013 release lists 112 distinct software security activities, a result of extensive surveying, that someone somewhere in the industry performs. Whether or not each of them is applicable to your situation is up to you to analyse. Some other sources, such as OpenSAMM by OWASP, a web security community, and the 'Fundamental Practices' paper by SAFECode, an industry forum, are more prescriptive and tell what you ought to be doing. Unless you are already familiar with these resources, the reader is encouraged to go and have a quick look – it will clarify many aspects of what will follow.

In practice, you will find very similar things in all of these. The differences emerge in the way you slice and dice the activities when trying to actually do them.

Each company has its unique set of business, technology and cultural constraints, so it is not possible to give a detailed one-size-fits-all guideline. However, most companies would do well by making sure at least the following practice areas are reflected in their software development activities:

>> Security risk analysis: An activity that attempts to determine who might want to attack your software, why, how, what would be the impact, and how this risk could be mitigated. This is often referred to as "threat modelling" or "architectural risk analysis". As a result, you should expect to see new and changed use cases, requirements, constraints, and test cases.

>> Setting and enforcing a set of development-time constraints and guidelines: These activities can vary from mandating the use of a specific framework to using static analysis to identify bugs in code. These require understanding how your technology choices (e.g., choice of language or framework) contribute to potential vulnerabilities.

>> Security testing: Positive testing for security controls and negative testing that probes the "misuse cases" of software. This can be exploratory or automated, examples being freeform manual penetration testing and fuzz testing (a type of fault injection test). This should get its input from the risk analysis.

>> Vulnerability management: A set of activities that ensure that your company hears about vulnerabilities in the code you deliver or use. This addresses things like upgrading and patching vulnerable components.

There are other practice areas (e.g., massaging your top management to buy into all this) that should be addressed as well, but here we only discuss the activities that strictly fit inside the core software development process and its immediate assurance activities.

Typically, companies start with security testing and reactive vulnerability management, and then make their way towards more proactive security if they feel the business need for it. Unfortunately, many companies don't see this need until they're hit. Some reasons for this can be that customers do not very often require evidence of software security, and bad things are statistically speaking rare and not normally distributed, and information about them is not publicly shared. However, there are some developments that may actually cause customers to ask for it: In Finland, the VAHTI application security guideline of 2013 will be the de facto requirement for public sector procurement, and globally, ISO 27034 is likely to have a large effect. Microsoft already declared conformance of their SDL to 27034-1 in May 2013.

Driving security in agile product management

Risk and investment

Each software security activity is an investment, and hence it needs to have a business reason to happen. A good way to find this business rationale is to base all security activities on risk analysis. If risk can be quantified in monetary terms, it instantly gives an upper limit for the security activities' acceptable cost.

The difficult part is that risk has two factors: impact and probability. And the probability is very hard to pin down. Still, even a token attempt to quantify the risk helps in determining what type and level of mitigation makes sense and what obviously doesn't. On high level, the risk-based software security development activities start with modelling the attack and performing architectural, or design-level, risk analysis. This tells you what sort of attacks are plausible, what your attack surface looks like, and essentially creates a backlog of security features (controls) and various tasks (such as testing) that are assumed to mitigate the perceived risk.

This also makes it possible to assess the security of the resulting product or system: For each identified risk, some evidence of the mitigation and control activities should be later available. This forms the basis for mature software security assurance.

Classes of Software Security Activities

For the purposes of this essay, software security activities are divided to three groups:

1. Functional features. These are the "security features" or controls, typically things like authentication systems, logging, using a security protocol to protect confidentiality of transferred data, and privacy-related user stories such as prompting for user consent. What makes them "functional" is the fact that they can be seen as new feature development, and there can usually be a positive test case that verifies whether the functionality works "to specification". Of course, security features can also have security bugs – but finding those belong to the next two categories of activities.

2. Engineering tasks. These are security-related tasks that need to be done. A task has a beginning and an end. Some tasks are one-shot and some are recurring; however, even the recurring tasks are "done" at some point. An ongoing concern that never ends is not a task for our purposes here. Security-related engineering tasks include, for example, doing risk analysis and performing an exploratory test run.

3. Ways of working. These are security-related concerns that may include work, but are "never-ending". Some concerns might not be actual work, but more related to how things ought to be done. Examples include secure coding guidelines, actively keeping your top-N security bugs list in the developers' minds, and keeping up to date with open source components.

Work Made Visible

There is one vehicle for getting stuff done in an agile process, and that is the Product Backlog. This is the list of work that needs to get done. Whether or not the items on the Product Backlog (for which I will use the term Backlog Items) get consumed in Scrum sprints, or on a kanban board, or through some other agile method is mostly irrelevant for our discussion here.

The important thing to understand is that unless a work item is on the Product Backlog, it might not exist at all. A well-oiled and mature team could still produce it, but as it is not strictly required, you cannot really tell in advance whether it will get done. Also, unlike a Product Backlog item that gets archived as being complete, something done automatically on the side might not leave evidence behind. If you are looking for a security assessment or a higher level of security assurance, activities without evidence are very hard to prove.

Let's take an example from Scrum. Scrum teams are expected to deliver a "shippable" increment after every Scrum sprint. This means that the additional functionality implemented during the sprint has to be on a high enough quality level to ship – it doesn't mean it has to be feature complete. This is known as "Done" and the criteria that the team applies to know whether this is true is often known as the "Definition of Done".

It would sound realistic to add security aspects to the Definition of Done, right?

Unfortunately, this may only be successful if the development team is one of the rare ultra-mature performers who can stand up to their rights when threatened with a loss of incentive bonuses. In the real world, work that is dictated by the team's internal quality criteria often gets omitted and skipped, pushed to future (often into a "hardening sprint"), and to an outside observer, it just looks like the team is taking far too much time to deliver functionality. When you take a few steps back from the development team, a plush Definition of Done looks like overhead. And overhead is a prime candidate for "optimisation". Therefore, one of the key targets for driving software security in an agile product creation process is to always aim for the Product Backlog. The Product Backlog can, fortunately, accommodate for various different kinds of requirements. In addition to the bread-and-butter functional features, Product Backlogs can also carry something called Acceptance Criteria. These can be thought of as Backlog Item specific mini-Definitions of Done. Acceptance Criteria may be extremely useful for security engineering, as we see in the next section.

In addition, not everything on the Product Backlog needs to be a functional feature. Product Backlog can be used to drive any kind of tasks that have a beginning and an end. Implementing a specific test case for test automation, for example, can be a Backlog Item.

With these alternatives in mind, let's see how the different kinds of software security activities can be driven on the Product Backlog.

Putting Software Security Activities on the Product Backlog

Earlier, we classified software security engineering activities into three classes: functional features, engineering tasks, and ways of working. Each of these can be driven on the Product Backlog.

"Functional features" are the easiest: Just create a user story to implement the feature, and add it on the backlog, and drive the work as usual. Sometimes, however, it is difficult to say exactly what to do. It is very typical, for example, that architectural risk analysis (threat modelling) turns up a risk, but it is not immediately clear how it should be tackled. Instead of a user story, an option is to create an "attacker story" instead (Sindre and Opdahl, see references, called this a "misuse case"). Whereas user stories specify what users should be able to do in order to provide some value, attacker stories specify things that should not happen. Attacker stories act as placeholders on the Product Backlog, and the idea is that developers will transform them into actual implementable security controls, engineering tasks, or ways of working later on. This can happen in whatever way user stories are usually processed – whether it is in Sprint Planning, Backlog Refinement (a.k.a. Grooming), or within the 5 to 10 per cent designer time allocation that many advocate for backlog maintenance work. Adding an attacker story on the Product Backlog makes the issue visible, and ensures that it is not forgotten in some obscure corporate risk register, an Excel sheet, or a to-do list on a sticky note.

"Engineering tasks that are one-off" are also easy: The Product Backlog is a good way to drive them as well as functional features. However, tasks on the Product Backlog are not intended to live forever. They get prioritised, consumed, and done. Resurrecting a recurring task again and again on the backlog is against the spirit of the backlog.

"Recurring tasks" can be tackled in two ways. A natural question is to ask whether something that recurs could actually be automated. There are some software security activities that lend themselves quite well for automation – as an example, fuzz testing, static analysis, and basic vulnerability scanning. In these lucky situations, you could create Product Backlog item that is a task to automate something. Once the automation task has been done, the actual engineering task is automatically run in whatever build or test automation or deployment system you have. It then recurs without having to appear on the product backlog.

Of course, many security engineering activities are difficult to automate, threat modelling or architectural risk analysis being an example, and still recurs again and again as new features are introduced. One part of the solution is to ensure that the activity is so well facilitated and trained that development teams can do it as independently as possible. The other part is to add the activity into the Acceptance Criteria of existing features. The Acceptance Criteria are usually the quality criteria that apply to that specific feature, and whether they have been met is evaluated before the backlog item in question is accepted as being complete.You could think of them as a backlog-item specific Definition of Done.

An additional bonus of using acceptance criteria to drive recurring engineering tasks is that not all features are equally risky. A minor modification of the user interface opens less attack surface than opening a new REST API to the big, bad Internet. You can, for example, selectively apply threat modelling as an Acceptance Criterion to a feature based on whether the feature "looks" risky by some heuristic. Some organisations have a concept of a "Definition of Ready", or "Ready for implementation", which is a kind of checklist whether a backlog item is well enough specified. Triaging features and adding security engineering tasks into their Acceptance Criteria can be done at that stage.

Once an engineering task is part of Acceptance Criteria, the great thing is that if the feature will be dropped, all the now useless work goes with it. Similarly, if you try to estimate the size of a backlog item, having time-consuming security engineering tasks tagged to it will make that work explicitly visible.

"Ways of working" are probably the trickiest software security activities to drive. Many of them aren't "work" in the traditional sense but instead about "how" to do things.

Many ways of working can be distilled – at least partially – into checklists. Coding conventions, training needs for your organisation's top-N bug lists, configuration parameters and API security guidelines can all be checklists. And once you have a checklist, you can define a task to go through the checklist. And finally, once you have this (recurring) task, you can treat it as an engineering task that you can add to the Acceptance Criteria like any other engineering task as previously discussed. Typically, coding conventions could also be automated through introduction of static analysis or linter tools.

This leaves a category of **"software security activities that are too fluid or poorly understood"** to be checklists, and that aren't really tasks. One example could be how fundamental technologies are selected. For example, some web frameworks take care of various web vulnerabilities out-of-the-box, and this could be an important consideration when selecting what technologies to use.

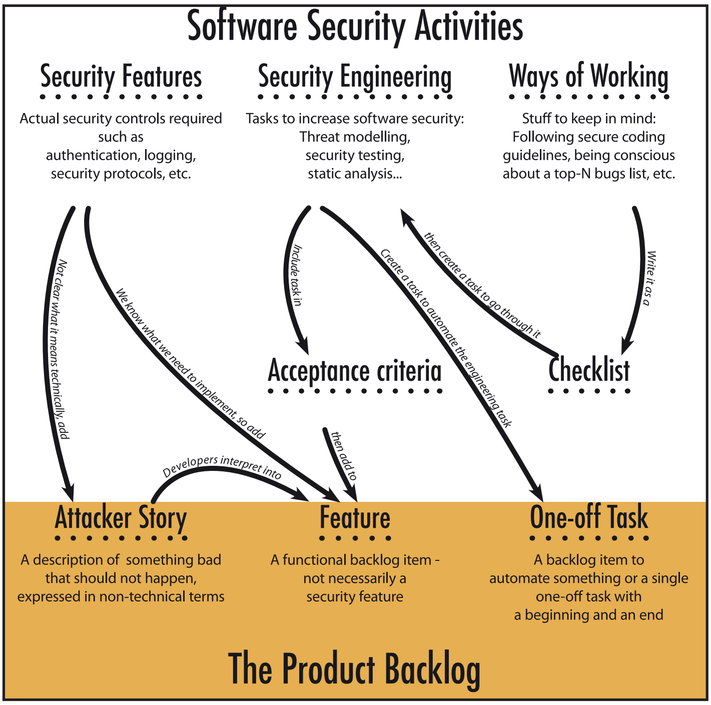
There is no really good mapping in a typical backlog process for this sort of activities. Some organisations running Scrum have something called "Sprint Zero", which is a themed sprint used to do stuff that needs to be done only once and preferably before doing anything else. However, this category of activities is clearly in the minority. Selecting the right underlying technologies is a fairly effective and critical software security decision, but beyond that, most important software security activities fall into one of the earlier categories.

Evidence and assurance

Agile methods are often accused of being light on evidence. This probably stems from the Agile Manifesto that says "[...] we have come to value [...] working software over comprehensive documentation". This invokes a mental image of a chorus of engineers singing "we don't need no documentation", although all it actually says is that the Manifesto signatories value getting things to run more than they value paperwork.

If you don't need to prove anyone but the immediate development team that you are secure, then you actually could throw documentation out of the window.

However, if you have a customer, a certification body, a regulator, your Privacy Officer, or some company internal control function that wants to see some evidence, you need to generate that evidence. The good thing is that the evidence does not necessarily have to be a traditional paper document, and even if such a thing would be needed, you can run it through the backlog. Let's first consider a case where you need to actually produce a paper document for a third party, say, a regulator, for a one-off certification effort. In the classification of software security tasks, above, that would be a non-recurring engineering task. This means that you can just stick this work on the Product Backlog and prioritise it accordingly. If the document is a collection of design specifications, you can also drive it as a set of Acceptance Criteria spread over all the features that need to be documented. This makes the document to appear as a proper project deliverable, and not as additional overhead. It makes the work visible, and the time investment has to compete with other activities that produce value. Let's say that the document would be required for just a single customer; this would force product management to decide whether the customer is important enough to warrant the work. Beyond traditional documentation, agile methods offer ample possibilities for extracting evidence of software security activities. When designing your evidence gathering, you should think about:


Software Security Activities
Security Features
Actual security controls required such as authentication, logging, security protocols, etc.
Security Engineering
Tasks to increase software security: Threat modelling, security testing, static analysis...
Ways of Working
Stuff to keep in mind: Following secure coding guidelines, being conscious about a top-N bugs list, etc.
Not clear what it means technically, add
We know what we need to implement, so add
Include task in
Create a task to automate the engineering task
then create a task to go through it
Write it as a
Acceptance criteria
Checklist
then add to
Developers interpret into
Attacker Story
A description of something bad that should not happen, expressed in non-technical terms
Feature
A functional backlog item - not necessarily a security feature
One-off Task
A backlog item to automate something or a single one-off task with a beginning and an end
The Product Backlog

Summary of how security activities end up on the product Backlog

- Do you need to present the evidence proactively, or is it enough if you can locate evidence during an audit or when requested? (I.e., can we have "lazy" evidence – if you can pull this off, this is likely to be more cost-effective.)
- How far away from the actual end product – typically executable code or system configuration – does the evidence stray? Is the evidence about the end product or the creation process? Which one do you need to collect?
- Can the evidence stand up on its own so that a third party can plausibly believe in it, or would the third party have to have full trust in you before they can trust the evidence?

With this in mind, you can use the following sources of evidence in an agile process. The fun thing is that unless you need to collect this data up front, the overhead in these is either minimal or effectively zero during the development.

- Product Backlog items that have been completed and archived. These can be probably retrieved from the backlog management system as required. The author was made aware of a company who used paper-based kanban cards that, once the task was completed, were physically signed off by the product management and the lead engineer, scanned, and archived. Although this sort of evidence requires a level of trust in the overall process, the granularity and incremental sign-off of tasks makes it easier to trust than a single, all-encompassing claim that "we've done what we are supposed to have done".

>> Product Backlog items that have not been completed. If, for example, you run a threat modelling activity and as a result, generate new Product Backlog items for controls, and those haven't been done, they still count as evidence of risk management activities having been performed. If all the risk controls have been put on the backlog, the security controls that haven't been done are your residual (remaining) risk. You can therefore more easily quantify your system's residual risk in technical terms.

>> Test cases (perhaps in test automation) that pass. You can show what security tests you are doing, and which of them pass. This is evidence that directly relates to the executable code.

>> Static analysis findings (perhaps run in continuous integration). You can either get a report of raw findings, or you can set a baseline and show that the error density has not increased, or has decreased. This, again, is direct evidence of source code security.

References

Bryan Sullivan: Streamline Security Practices for Agile Development. MSDN Magazine, November 2008. http://msdn.microsoft.com/en-us/magazine/dd153756.aspx

Rohit Sethi: Managing Security Requirements in Agile Projects. InfoQ, June 2012. http://www.infoq.com/articles/managing-security-requirements-in-agile-projects

SAFECode: Practical Security Stories and Security Tasks for Agile Development Environments, July 2012. http://www.safecode.org/publications/SAFECode_Agile_Dev_Security0712.pdf

Antti Vähä-Sipilä: Software Security in Agile Product Management, 2011. http://www.fokkusu.fi/agile-security/

Notes and presentations from the workshop on secure software development and agile methods, organised by the Finnish Information Security Association and Agile Finland, April 2010: http://confluence.agilefinland.com/display/af/Secure+software+development+and+agile+methods+-+notes

Guttorm Sindre and Andreas L. Opdahl: Capturing Security Requirements Through Misuse Cases, 2001. http://folk.uio.no/nik/2001/21-sindre.pdf

Security activities in scrum control points

Ville Ylimannela, Tampere University of Technology
Marko Helenius, Tampere University of Technology

Executive summary

In this chapter are mapped important control points in Scrum and described what security activities should be performed in the control points. The security activities discussed are: security templates, threat assessment, feature flagging and residual risk approval. The criteria used to judge the activities are: documentation, customer interaction, speed of execution, simplicity and security. Control points in which the security activities are applied to are: the product owner, sprint planning and sprint review.

Scrum control points

By a control point we mean a point that can be used to affect the Scrum process. Both people and meetings can be seen as control points in the process. In Scrum there is one crucial role, which is the product owner. Product owner is single-handedly responsible for adding items to the product backlog and constantly prioritizing and re-organizing the backlog. It is also the product owner's responsibility to decide whether the software goes out at the end of a sprint <<FootNote(see Microsoft 2012a)>>. In short, the product owner is the gatekeeper for everything that comes in and goes out. This is a crucial control point in the Scrum process.

The problem here is that the product owner, who is responsible for the most important thing in the process, does not usually receive adequate security training. The product owner is also the only person in Scrum who does not really know what goes on inside the Scrum team. This means that the product owner is probably the worst person to decide if the software is secure enough. There are few different ways of solving this:

- Get the product owner more involved in the process itself and especially security activities.
- Make sure everything security related is documented and presented to the product owner.
- Give someone in the team the right of veto to security related issues. This would mean that the person deciding whether the software is secure enough needs to have in-depth knowledge of the software and its security.

In Scrum, the team is composed of people who have the required skill set for creating the software needed. The idea is that everyone can use their own expertise to complete the tasks they have been given. As the tasks have been completed the person will move to the next one. This leads to the fact that every person in the team is also responsible for security. The only person who knows if the task meets the security requirements of the software is the person performing the task.

[1] see Microsoft, 2012a

There are also other control points than the team, product owner and sprint review. Daily Scrum could be considered as one. During daily Scrum the team gathers up for a quarter of an hour meeting to discuss recent events and report on personal process on tasks.

Sprint planning is a crucial control point. This is where the team chooses which features they will implement during the upcoming sprint. If SDL-Agile is used all new features chosen will have to be threat modelled. Nevertheless this would be a good time to threat model all the chosen features, since the product owner would also get involved in the identification process, meaning the users' representative would also be heard. Most of the every-sprint activities in SDL-Agile are close to implementation, thus they really are not issues which one might bring up during the meetings. [2]

Scrum of Scrums (SoS) meeting is also a control point for the software development process. This means that teams working on the same software gather up to discuss about large-scale decisions. The higher we go on the process, the more general the security requirements discussed during the meeting will be. Daily Scrum is on the low end of the scale while SoS meeting is at the other end, as shown in Figure 1.

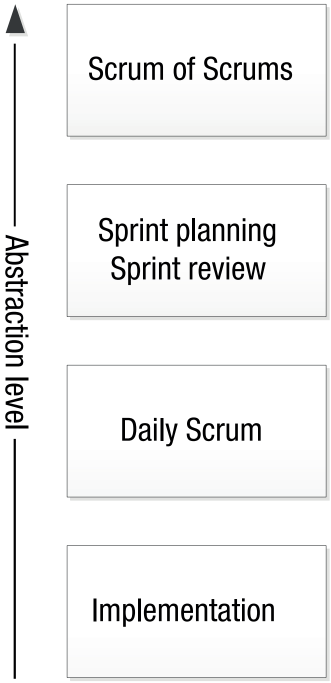


Figure 1: Abstraction level of security issues discussed

Below daily Scrum would be the actual implementation of the security features and testing required for creating secure software. The basic security requirement can be the same in all abstraction levels, however, the way it is presented is different. For example, if the security requirement is "Software need to be robust", this could take the following forms:

- >> SoS: When reading initialization file, the software must be able to handle securely any file, including corrupted and potentially malicious one.
- >> Sprint review: Features foo and bar were implemented to prevent crashing if the initialization file is corrupt.
- >> Daily Scrum: Yesterday a team member started implement ing feature foo and today there will be fuzz testing with corrupted files.

Security requirements and controls

The definition of security requirement varies. Microsoft SDL (Security Development Lifecycle) identifies security functional requirements as follows: "A security risk assessment (SRA) is a mandatory exercise to identify functional aspects of the software that might require deep security review". In SDL's case security requirements are strongly linked with risk assessment. The security requirements are risks identified during the assessment. All the security activities in SDL-Agile could also be considered as security requirements, which must be met in order to the software be released at the end of the sprint.

In SDL-Agile, the baseline for the threat model is usually created at the beginning of the project. As new risks emerge the model will be updated every sprint. What this means is that security requirements will continue to evolve throughout the software development.

There are different ways of identifying security requirements. As risk management in general, the process of identification usually starts with gathering required material of the software.

[2] see Microsoft, 2012c

This should include at least: [3]

1. External entities
2. Processes
3. Data stores
4. Data flows

When this is done, it is time to start identifying threats. This can be done using STRIDE or some other security risk identification method. The focus here will not be in functional requirements identified at the risk assessment, but the risk assessment itself, which is also a security requirement. [4]

Another way of identifying security threats are security stories and abuse cases. Security stories are templates which address users' security needs such as "I need to keep my contact information hidden." or "I want to know where my account was used last time". These requirements are later translated to positive functional requirements such as "keep track of the user's IP address". Security stories work well in identification of privacy issues, but might not be quite as useful against other security problems, such as buffer overflows or cross-site scripting vulnerabilities. The abuse cases are very similar, but they are from the attacker's point of view; the product owner might lack the expertise to solve a problem, but he or she can tell what should not happen. [5]

Despite the method used in identification, after the identification it is time to apply mitigations to the threats discovered. This could mean using encrypted connection for data flows, re-designing old code or even accepting the risk, when nothing can be done with the given resources. Having identified the threats does not make the software secure. It also requires a working implementation of the mitigation chosen.

The mitigations used are dependant on the requirement, thus a deeper look for potential mitigation methods is out of scope . The bottom line is, however, that there are two parts during which the security requirements will consume resources from other tasks: Identifying the security requirements/threats and applying controls.

All requirements can be split to functional and non-functional requirements. Functional requirements are often more specific and tell how the software should act, when doing something. An example of a functional requirement could be "When a user presses ESC a menu window will pop-up". Functional requirement should always be positive and it can be tested somehow. This is not the case with non-functional requirements.

A non-functional requirement specifies criteria which the software must meet. "Software needs to be secure" or "Software needs to be able to handle 100 simultaneous users" are both non-functional requirements. Non-functional requirements can often be translated to one or more functional requirements, which the team will implement during a sprint.

Security activities within control points

The focus in Scrum meetings is on large scale issues rather than on single vulnerabilities.This means, that the implementation of the chosen security controls is not usually discussed. The idea is to identify security risks and assign the right tasks to the right people. Moreover, sprint review serves as a point during which risks are accepted or the software is not published, if residual risks are too high. The most important control points in the Scrum process are:

1. The product owner, the person who is responsible for features which are added to the backlog and accepting the end results of a sprint.
2. Sprint planning. This is where the team decides what features will be implemented and who will implement them.
3. Sprint review, presenting the result of a sprint to the product owner

[3] see Shostack, 2007

[4] see Microsoft, 2005

[5] see Vähä-Sipilä, 2010

There are also other control points in the process. However, they are not so crucial from the security point of view. The process could also be modified if needed, for example, by adding an additional weekly security meeting. This could be useful when creating software which requires high level of security. The goal is not to create new control points for Scrum, but use the three existing control points. The focus will also be on understanding how the non-functional security requirements in Scrum control points will affect security of the software and how much resources will be required.

The next question is what type of security activities can be deployed in the control points. In the control points mentioned above, the security activities should be as non-technical as possible. Technical problems can be solved by transforming non-functional security requirements to functional requirements and implementing them. The security activities chosen are:

1. Security and abuse cases: These can help the product owner to identify privacy related security problems.
2. Threat modelling/Risk assessment: This is one of the key activities in SDL-Agile, which should be performed each sprint. According to SDL-Agile, all new features need to be threat modelled. Naturally some features will take more time than others.
3. Approving residual risks, an activity for sprint review: If a risk is identified it will be either mitigated or accepted. Ignoring to do anything is the same as accepting it.
4. Filter, which can be used to flag potentially risky features or tasks.

Most security activities can be done as tasks during the sprint as opposed to the Scrum meetings. We will next present what effect they might have in a specific control point and would they be better as separate tasks. The criteria used for judging is loosely based on the work of Keramati and Mirian-Hosseinabadi [6]. The criteria is shown below in Table 1.

Criteria, the level of...	**Reasoning**
documentation	Creating and reading through documentation is a slow process, this should be avoided when several people are gathered.
customer interaction	The representative of a customer usually attends the meetings. This could be a good time to ask questions.
speed of execution	Things which consume lot of time should be avoided during Scrum meetings.
simplicity, minimizing extra work	Reducing useless overhead is what agile software development is aiming for.
security	All the security activities will increase security, some more than others. Privacy issues are included in security.

Table 1: Criteria and reasons.

Product owner

The suggested security activity is to use security templates and to guide the product owner in creating non-functional security requirements, which can later be translated to functional requirements by the team. This should be a relatively fast procedure, especially if the security template is one-pager as Vähä-Sipilä suggested [7]. The criteria is in Table 2.

[6] see Keramati and Mirian-Hosseinabadi, 2008

[7] see Vähä-Sipilä, 2010

Criteria	Properties
Documentation	The required documentation is a short checklist, which fits the agile principles well. The documentation created consists mainly of new features.
Customer interaction	Customer interaction is needed when identifying security issues. However, the person performing the identification is the representative of a customer.
Speed of execution	Skimming through the features and trying to identify privacy risks is a relatively fast process. The list is not perfect, so it does take some time.
Simplicity, minimizing extra work	Product owner understands the privacy requirements of the software the best, meaning he or she is the perfect candidate for the task.
Security	Security templates help greatly in identification of privacy issues. Abuse cases can also be used to identify security vulnerabilities which should be addressed.

Table 2: Security templates

It would seem that security templates are a great fit for the product owner. The same principles apply to abuse cases, which is just a policy decision, allowing the product owner to create negative requirements. If this was done as a task by a team member instead of the product owner it would require constant interaction with the customer to understand the privacy requirements. As a consequence the customer interaction weight would be lower. This would inevitably lead to extra work as well. The return of investment is expected to be good for security stories, if used by the product owner.

Sprint planning

For this activity there are two tools. The first one is to perform the actual threat assessment during the sprint planning and the other is to the flag potentially risky tasks or features and perform the threat modelling as a task before starting to work on a feature. The criteria for threat modelling during the sprint planning is investifated in Table 3. The problem with integrating security to sprint planning is that it is a slow process, which also creates some heavy documentation. It is, however, a cornerstone for good software security, which makes it mandatory in software that require high level of security[8].

Criteria	Properties
Documentation	Threat modelling creates a lot of documentation, which is not suited for agile or control points.
Customer interaction	The product owner needs to be present only when the threat is at product level, and not technical. The product owner's presence would also be useful if attacker profiles are created.
Speed of execution	A slow process and a lot of documentation needed. Using methods like STRIDE for identifying the threats can speed up the process, but it can still take days.
Simplicity, minimizing extra work	Security will always need resources. One of the lean principles is "earlier is cheaper", which means that also threat modelling will, in a long run, prove to be worth the extra work at early stages.
Security	Threat modelling is a cornerstone for security.

Table 3: Threat assessment

[8] see Microsoft, 2012b

Despite brainstorming being a great way of identifying threats, it is faster than heavy methods like STRIDE. The other way of doing threat assessment is to perform it as a task before the implementation of the feature. This means, that features which include high-risk code should be flagged and assessed separately. Flagging the high-risk features and then performing a separate threat assessment is presented in Table 4.

Criteria	Properties
Documentation	Feature flagging does not create much documentation. However, the documentation will be done during the sprint.
Customer interaction	The product owner's presence is rarely required during the threat assessment.
Speed of execution	Flagging potentially risky features should be a fairly fast procedure.
Simplicity, minimizing extra work	Security will always need resources. One of the lean principles is "earlier is cheaper", this way however does eliminate some overhead during meetings.
Security	If the assessment is performed by several people (during the planning) security will most likely improve.

Table 4: Feature flagging

Sprint review

The activity chosen here was the approving of residual risks. The product owner should approve all the risks which have been identified and have not been completely been erased. The assessment is in Table 5.

Criteria	Properties
Documentation	Risk management creates a lot of documentation. Risks should be explained at feature level, but this might lead to reading thought some documentation to completely understand the risk.
Customer interaction	People using the software need to understand the risks which are involved.
Speed of execution	The product owner needs to understand the risks before he or she can approve them. Explaining things might be a waste of time in a Scrum meeting.
Simplicity, minimizing extra work	If a risk can not be approved, this should be known as soon as possible, meaning that the product owner should approve the residual risks before the meeting. Once again, earlier is cheaper.
Security	The product owner knows the required security level, so understanding the risks accepted is important.

Table 5: Residual risk approval

Customer interaction is in a crucial role when accepting risks on product level. The product owner is the only person who can accept such threats. Since the product owner will attend the review meeting, it is a good place to accept risks. The problem might be that the product owner will not accept all the residual risks. This means that the software will not go out. Ideally the risks should be accepted before the meeting, to avoid any unnecessary delays in the software development.

References

Microsoft. Microsoft SDL. 2012. Available: http://msdn.microsoft.com/en-us/library/cc307412.aspx#ED1 (Cited 11.12.2013).

Shostack Adam. Threat modelling. 2007. Available: http://blogs.msdn.com/cfs-file.ashx/__key/communityserver-components-postattachments/00-07-70-23-05/TM-Series-2007.doc (Cited 11.12.2013).

Vähä-Sipilä Antti. Product Security Risk Management in Agile Product Management. 2010. Available: https://www.owasp.org/images/c/c6/OWASP_AppSec_Research_2010_Agile_Prod_Sec_Mgmt_by_Vaha-Sipila.pdf. (Cited 11.12.2013).

Keramati H. Mirian-Hosseinabadi S. Integrating Software Development Security Activities with Agile Methodologies. 2008. AICCSA 2008. IEEE/ACS International Conference on Computer Systems and Applications Microsoft. High-risk code. 2012b. Available: http://msdn.microsoft.com/en-us/library/ee790613.aspx (Cited 11.12.2013).

Microsoft. SDL-Agile. 2012c. Available: http://msdn.microsoft.com/en-us/library/ee790617.aspx. (Cited 11.12.2013).

Microsoft. STRIDE. 2005. Available at: http://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx. (Cited 11.12.2013).

Risk management

Ville Ylimannela, Tampere University of Technology
Marko Helenius, Tampere University of Technology

Executive summary

Risk management in agile software development is not an easy as risk management is a heavy process. However, risk management is a key to increase security and there are ways to integrate it so that the agile nature of the development does not suffer. In this chapter is presented how to integrate security into agile development. The most notable changes are summarized in Table 1.

Activity	Suggestion	Timing
Plan	Identify security needs, make sure the team receives adequate training, create security cases and create attacker profiles. Create a checklist for high-risk features.	Once before the project starts
Identify	Decide whether the feature is high-risk or not. If the feature is a high-risk, book risk identification to sprint backlog, otherwise use light methods like brainstorming or delphi technique. Add identified risks to the risk board.	Before implementation of the feature
Assess	Try to assess impact and probability of the identified risks. Update the risk.	After identification
Respond	Create responses, make sure they're in proportion to total risk and the importance of the feature. Estimate the time required to implant the feature with the added time from response. Choose the best responses.	During feature implementation
Accept	Make sure all the risks are accepted by the risk owner, product owner or client.	Sprint review
Monitor	Digitalize the risks and remove the risk from board. Make sure to include all the information. Re-evaluate larger risks between few sprints.	Between 2nd to 5th sprints

Table 1: Summary of activities

Introduction

Risk management means identification and prioritization of risks based on their severity, then creating and applying controls to reduce identified risks. Monitoring risks is also a crucial part of risk management. There are two different approaches to risk management, proactive and reactive. Proactive means planning in advance to avoid risks. Reactive means reacting to risks when they are close to or already have become issues. Focus here will be on the proactive risk management.

Any introduction to risk management should include the definition of a risk. It can be defined as combination of probability and impact, which means a risk has two parts: the probability that something will happen and the actual loss which is suffered, if a risk occurs. Risk has effect at least on one project objective, which can be anything from consuming extra resources to missing deadline or affecting quality. A risk which has actually occurred could be referred as an issue. It is important to remember that risk management is an ongoing process throughout the whole project.

There are several different frameworks for risk management. One in popular use is PMBOK (Project Management Body Of Knowledge, Figure 1), which, in addition to other things, includes a framework for project risk management. It is written in very general level, so it can be applied to almost any project, including software projects.

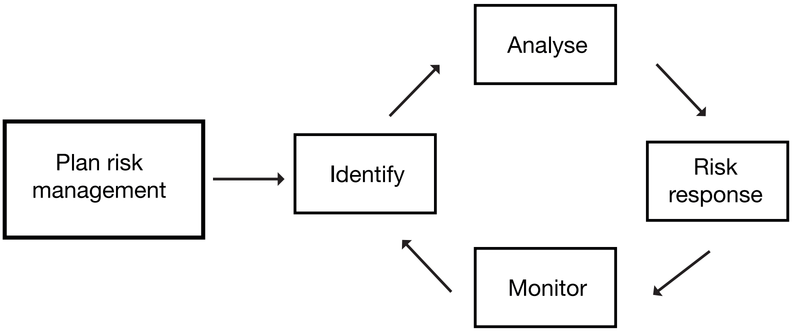


Figure 1: Risk management framework as defined in PMBOK

Planning risk management

Planning risk management refers to activities which define how to conduct risk management in the project, meaning it is done before the actual risk management. Careful planning increases the chances to succeed in risk management. Planning the risk management should be done as soon as the project starts. To perform the planning effectively, the risk management team requires the following information:

- Understanding of project scope. Meaning, they need to understand what the project includes and how important part the risk management is.
- Risk management requires resources, meaning money, time and people. Resources are important information when planning risk management.
- Communications management plan is a document, which defines how interactions will occur in the project. It determines who will be responsible for risks and responses at different times.
- Schedule management plan defines how overruns in schedules will be dealt with.
- Organizational factors, which include the organizations ability to withstand risks or attitudes towards risk management. Also there might be some organizational guidelines regarding risk management, like pre-made risk categories, standardized terms, concepts or templates, roles in organization and authority.

After the required information is available, the project team should have a risk planning meeting. During the meeting the team will decide on methodologies used, roles and responsibilities, budget, timing and definitions regarding risks. The output from all this is called risk management plan.

Identifying risks

During this part of risk management the goal is to gather participants and start identifying existing risks. These participants might include the project manager, members of the project team, customers, experts and stakeholders. The important thing is that everyone participating to the project should be encouraged to identify risks.

Risks are at different abstraction level, which means they are understood better by the people who understand that certain abstraction level. This is why risk identification should include people from different backgrounds:

- An engineer would notice a design flaw or bad solution, which would go unnoticed by others, since they are not familiar with design documents or engineering solutions.
- A consumer would notice a problem in usability. For someone who uses computers for a living, it might be hard to understand how to design usable interface, and not the most efficient one.
- Project managers would notice an error in badly planned time table.
- A stakeholder is the best person to ask: ''Are we doing the right thing from the business point of view?''

There are a whole range of techniques for identifying risks. These include but are not limited to: documentation reviews, information gathering techniques, checklists, diagramming techniques, SWOT analysis (Strengths, Weaknesses, Opportunities and Threats) and expert judgement.

A common information gathering technique is brainstorming. It would be a good that the team performing brainstorming is multidisciplinary. This basically means that the team should also include people, who are not part of the project team. There are few different approaches to brainstorming. One is the normal free-form approach, in which the team just throws out ideas and improves and sharpens them. Another approach is a mass-interview, where a moderator asks the team a set of questions. Other information gathering methods would be interviewing project participants, root cause analysis or delphi technique. The idea of root cause analysis, is to take a risk, and locate the underlying causes which might cause the issue. Deplhi technique is an anonymous questionnaire to project experts.

Checklists can be based on historical knowledge that has been gathered from similar projects. While checklists are handy, they are far from perfect. The problem is that checklists are never complete. There are always risks, which are not directly associated with any of the checklist points. It is important to consider risks which are not on the list. A checklist also needs to be updated as soon as new risks are found.

SWOT analysis is used to evaluate strengths, weaknesses opportunities and threats involved in a project. The analysis usually starts by identifying the strengths and weaknesses of the project organization. From these it should be easier to deduct any threats or opportunities which arise from strengths or weaknesses. Identification is usually done by brainstorming.

No matter what methodologies are used to identify risks, the output of this phase will always be the same: a risk register. A risk register is a crucial part of risk management. It includes a list of identified risks and a list of potential responses. While it is not part of risk identification phase to create responses, it is useful to include any suggested ideas.

Qualitative and quantitative risk analysis

During qualitative risk analysis, the team will prioritize risks based on their probability and impact. It is important to focus on the highest identified risks first. After the highest risks have been mapped, the team will perform quantitative risk analysis. The key artifact in risk analysis is the risk register created as a result of risk identification. For each risk item in the register, the team will assess the probability and impact. The probability can be measured by estimating occurrences per year. It can also be done by doing a rough estimate about the chance of the particular risk becoming an issue during the project. The impact can be measured in several different ways. It can be money, time or resources. Whatever the method used to describe the size of the impact is the total risk is calculated by multiplying the impact and probability. The information gained is usually placed to a matrix, which is called as a risk matrix. As a result of qualitative risk analysis is an updated version of the risk register. In the updated version risks are in order of magnitude.

In quantitative risk analysis it is numerically analysed the effects of identified risks to a project. This is only done to the risks of highest magnitude. In some cases this step can be completely skipped.

Risk response

This is the phase of risk management where actions to reduce risks are created. It is important to remember that the risk response must be in proportion to the risk. Often there will be several different potential responses to choose from, some being more efficient than others. There are different strategies for dealing with threats. The risks can be avoided, transferred, mitigated or accepted. Each risk will be given a risk owner meaning the person who is responsible for reducing and monitoring the risk. Avoiding risks means changing the project management plan in a way that completely eliminates the risk. Usually it is not possible to completely avoid a risk. However, if a risk arises early in the project, it might be completely avoidable by changing the strategy, clarifying requirements or acquiring some special knowledge.

Risk transfer means outsourcing some of the negative impact to a third party. This is often done with financial risks via insurances, or buying an extended warranty for devices. The idea of risk mitigation is to effect the probability or impact of the risk. Since the total risk is calculated from both of these factors, reducing either one will lower total risk. If possible, one could also reduce both factors. In some cases it is not possible or economically viable to do anything for the risk, which means the risk is accepted. The accepted risks are called residual risks. At the end of this phase, the risk register should once again be updated. The update should include:

- Risk owners and their assignments
- Budget for the chosen responses
- Chosen risk responses
- Accepted residual risks

Also, if there are changes to the project management plan due to the chosen risk responses it should be updated as well.

Monitoring and controlling risks

In this phase, the risk responses created in the last phase will be implemented. After the implementation the monitoring will begin. The person(s) chosen in the last phase as risk owners will be responsible for applying risk reduction procedures.

Risk register must be updated during this phase as well. The monitoring might often lead to identifying new risks. Some of the risks in the risk registerer might be outdated due to effective risk management procedures. It is of importance to audit whether the applied risk reductions are working. Monitoring should be done until the project ends. Risks should be reassessed on several occasions, since new risks will arise and old ones will disappear. Reassessing the risks would basically mean jumping back to identifying risks.

Existing frameworks for risk and security management in agile software development

There have been some suggestions for merging security, not necessarily just risk management, with agile practises. Despite the fact that they are not directly about risk management, the models present some very useful ideas. This chapter presents four models for adding security or risk management to agile software development.

MSDL agile

MSDL (Microsoft Secure Development Lifecycle) is a security assurance process, which focuses on software development. It works best when used in waterfall or spiral based software development cycles, and not in agile methods. MSDL has played large role in integrating security as a part of Microsoft's products. The goal is to reduce the number and severity of vulnerabilities in software.

Microsoft has developed an agile version of MSDL. The difference between normal MSDL and MSDL agile is that not all requirements are meant to be completed every iteration. The idea behind MSDL agile is to fuse the concept of secure software development with agile methodologies. Since there are no development phases (design, implementation, verification, release) like in classical waterfall-based development, some adaptations have been made. Also the short iteration time makes it impossible to complete all tasks on every sprint.

MSDL agile divides requirements to three categories: [1]

- One time requirements.
- Bucket requirements.
- Every sprint requirements.

One time requirements

As the name implies, these tasks are completed only once during the project lifetime. These requirements are completed at the start of a new project, or when you first start using SDL-Agile with an existing project. The SDL-Agile one time requirements are fairly simple, apart from creating a baseline threat model. All SDL-Agile one time requirements are not meant to be completed during the 1st sprint, since in addition to one time requirements, the team would also have to complete every sprint requirements and a bucket requirement.

Threat modelling can be considered as a cornerstone for SDL. The idea is to create a baseline threat model at the beginning of the project. This is then updated every sprint with new risks found from those parts of the product which are under development. The threat model is a critical part of SDL, because it helps to determine potential design issues which might effect security and it helps to drive attack-surface analysis and fuzz testing. [2]

Every sprint requirements

Some SDL requirements are considered so important, that no software should be shipped out without them being completed. No matter how long sprint is, all every sprint requirements must be met before releasing the product. This means all external audiences, so whether the program is a box product release, web release or an alpha release, it cannot be released without these requirements being met. Examples of every sprint requirements: [3]

- Threat model all new features.
- Run analysis tools on build.
- Only use strong cryptology.
- Do not use banned APIs (Application programming Interface) .

[1] see Microsoft, 2012a

[2] see Microsoft, 2012b

[3] see Microsoft, 2012c

In the list above, all points are requirements that must be done. However there are some optional suggestions as well, such as using /GS-option in compiler, which means buffer security check. The option attempts to detect buffer overflows. [4]

Bucket requirements

Bucket requirements are such requirements, that they must be done several times during product lifetime. One does not need to complete all of them on every sprint. The idea is to have three different categories of requirements, verification tasks, design review and planning. Each category includes several different tasks.

The team would be required to complete one tasks of each bucket on every sprint. These are done in addition to every sprint requirements. There is no order in which the bucket requirements must be completed. It is up to the team to decide what tasks to do during sprint. However, no requirement can go further than six months without being addressed. This is due to fact that all SDL requirements have been shown to reduce the amount of security problems.

OWASP conference

During OWASP (Open Web Application Security Project) conference 2010, Vähä-Sipilä introduced product security risk management in agile environment [5]. Scrum was used as an example of an agile method.

Scrum was devided into three different phases:

- Requirement engineering. This means the part where user stories are broken down to features, which in turn are broken down to tasks.
- Development phase. This is the part where Scrum team implements to chosen backlog items.
- Integration and releasing. In this phase the code is released to internal or external groups.

It was also noted, that the parts are not phases, but the whole thing is working like a machine, meaning everything is done simultaneously, rather than in chronological order.

Requirement engineering

There are two different suggestions for this phase, security stories and abuse cases. Security stories are generic user case prototypes on security aspects. An example of a security story could be: "As a user, I want my personal information be hidden by default, so it can't be misused". These examples aren't meant to be used directly as requirements. These scenarios are supposed to be very general, so they can be applied to several features. Security cases are good for understanding and doing the correct things from privacy point of view. [6]

As product owners might not be technically capable of creating positive security requirements for existing problems we suggest using abuse cases. These are scenarios which shouldn't be allowed to occur. Abuse cases are broken down to positive functional requirements.

Development phase

Three different ideas were suggested here. First of all, security criteria should be added to the sprint review's DoD (definition of done). This means someone in the team needs to ask whether they're done from a security point of view. The team needs to have the power to say "no". It could often be, that people responsible for getting financial gain out of the sprint might be tempted to ignore security issues in order to get results. The second tool which should be used every sprint is threat modelling. This was suggested to be done in research spikes. This means, that for every potentially risky task, there should be an additional task in the sprint backlog to do threat analysis. Threat analysis is scheduled as a task in the backlog

[4] see Microsoft, 2013a

[5, 6] see Vähä-Sipilä, 2010

before the actual task which is being modelled. It doesn't create any deliverable code.

To identify potentially risky tasks, the team should have a checklist, which mirrors typical problems in the past or anticipated problems in the future. The good thing about adding threat modelling as research spikes is that it makes risk management visible, as opposed to creating invisible overhead.

Integration and release

During this phase, it is the product owner's responsibility to accept residual risks. If the remaining risks are too high to be accepted, the product owner needs to schedule new threat modelling for remaining.

Stockholm University research

A research by Stockholm University describes an integrated model for merging risk management with agile practises [7]. Both business and engineering risks were considered. The suggested risk management framework has four different organizational layers: risk management forum [8], business manager, product owner and the team. Risk management forums is an organization wide way of communicating risks. Business manager is responsible for high level planning and managing risks at business level. The product manager is similar to the Scrum's product owner and the team is similar to the scrum team.

The study suggested in the integrated model, that risk identification, assessment and management planning would be done (when applied to Scrum) by the the product owner and Scrum-team. Risk monitoring, sign-off (meaning accepting residual risks) and postmortem analysis would only be done by the team.

As for risk ownership, the study suggests that the product owner is responsible for risks in the requirement engineering phase. The product owner can delegate risk management to other person, usually someone inside the scrum team. During the sprint, or implementation phase as it is called in the study, the team is responsible for identifying and managing any new risks. The study states that: "The team leader supervises the risk management" Which is obviously a problem in Scrum, since there is no team leader, but a self-organizing team. The role could be adopted by the Scrum-master or someone who is interested in being responsible for risk management in the project.

The study also emphasized the importance of proper risk communication channels. There are in total six different channels, as presented in Figure. 2.

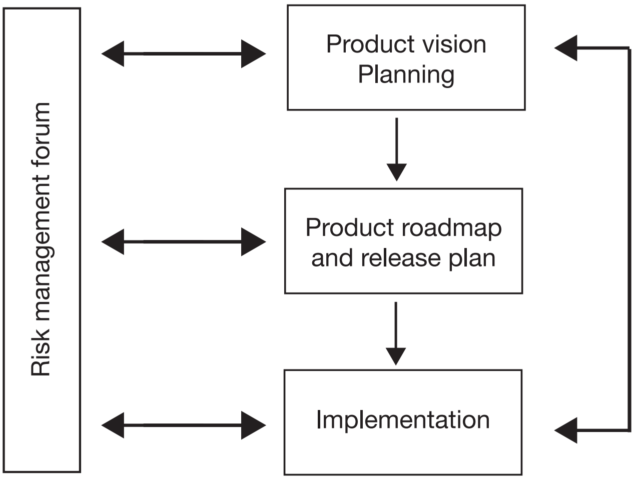


Figure 2: Six risk communication channels defined by Nyfjord and Kajko-Mattsson. [9]

According to the researchers the model was limited, because it didn't provide any suggestions how to actually perform risk identification and assessment. The model was also criticised for compromising agility. Risk management forum was thought to be too authoritative for agile software development, and was suggested to be replaced with an organization wide wiki-platform.

Risk board

There have been suggestions for applying practical methods for managing risks in agile environment by using something called as a risk board [10]. Risk board is simply a whiteboard with categories for different risk types:

[7, 8, 9] see Nyfjord & Kajko-Mattsson, 2008

new risks, mitigated, avoided and transferred. Each risk is written on a coloured paper. The colour has a meaning, it represents the size of the risk, red being the highest risk, followed by yellow and finally green.

Figure. 3 is an example of a risk board suggested by Wijewardena. The same board is used to manage all risks which arise during software development, not just security risks. Risks would be written on a sticky note and attached to the board. WIP stands for work in progress. According to Claude [11] risk management is qualitative in agile development. Risk are identified during all Scrum meetings. Low-tech methods are used when identifying and assessing the risks, such as brainstorming and checklists. Risks are formally part of the sprint planning and review. Risks are also assessed and managed in all Scrum meetings. Monitoring is done by using the risk board. Using these methods risk management becomes collective and highly visible.

New risks	WIP	Ignored	Mitigated

Figure 3: Risk board as suggested by Wijewardena, 2009, WIP = Work in progress [12].

Challenges and limitations of agile security

Agile development is based on short iteration cycles and providing constant flow of program code to the product owner. Traditional risk management is a slow and comprehensive process. Agile processes aren't meant to create heavy documentation, which is usually what risk management produces. When creating a new model to merge agile and risk management worlds, it is important to stay loyal to agile manifesto and lean principles.

The following results are based on interviews of two companies using agile software development process. Companies were questioned about their risk management in agile software development. Interview brought up shortcomings and slight problems in both cases. The biggest problems when conducting risk management in agile project environment were the following:

- Resources are a problem. Since the goal is to create a working increment each sprint, it might be hard to find time and people to conduct risk management each sprint. The general consensus was, that risk management takes five percent or less of the total time available each sprint.

- There is no clear risk owner, which can definitely pose a problem.

- Acceptance of residual risks. Who has the power to accept residual risk at different abstraction levels?

- There was very little security training. Technical understanding of some managers might be lower than Scrum teams'.

- Risk communication is a problem. When is a risk high enough to be reported for a product owner for approval?

[10] see Claude, 2009 and Wijewardena, 2009

[11] see Claude, 2009

[12] see Wijewardena, 2009

- >> If a risk is not directly related to a backlog item, it might go completely ignored.
- >> Sprint's DoD lacks security aspect.

The interviews showed that risk management is considered as an important part of agile software development. Another thing is maturity of the software. There is less time used on risk management as the software develops further. This is natural part of software development. Risks identified early in the process will not cause as much financial loss as the ones found later on. What this basically means is that a good model for integrating risk management to agile needs to address this issue.

A suggested model for agile security

This chapter defines a model for managing risks in agile environment using theories introduced All major risk management phases, from planning risk management to monitoring and re-evaluating risks, are included in the model. The model revolves around a risk board, which is used and updated constantly throughout the software development cycle. The model defined in this chapter has not been tried in any real-life software development scenario. Despite the fact that Scrum terms are used and the model is initially thought to be part of Scrum, there is no reason why it couldn't be applied to other agile frameworks.

The focus is on security risks, however there is no reason why this method couldn't be used to identify other risks as well. It is also good to remember that security and risk management are much more than a backlog item. Just following good principles doesn't guarantee good results.

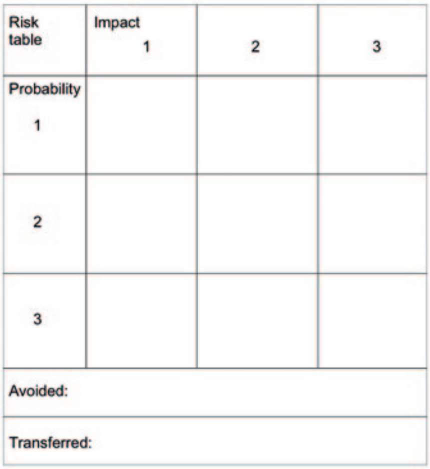
Risk board

The board used to display risks is a modified version of the board suggested in subchapter 3.4. The difference is that the board is in a form of a risk matrix. The matrix form of risk board allows the team and product owner to get a good idea about the general risk level in the project. The board below is a 3x3 risk matrix, however, there is no reason why a larger matrix couldn't be used. The board itself is usually a whiteboard.

The size of the risk is calculated by multiplying impact and probability. The problem with the board might be the estimation of probability. In some scenarios the probability might be hard to assess. If the team feels this is a constant occurrence, they could start using simpler board, with just low, medium and high risks. If a risk is completely avoided or transferred to someone else, the sticky notes should be placed away from the matrix to a dedicated area below. The template for risk notes is defined in Figure 5. The risk notes should have at least the following information: feature ID to which the risk is related, who is responsible for implementing the feature and a short description of the risk.

There are in total two colours of sticky notes, red and yellow. Obviously any other colours could be used. In this case the red notes are risks and yellow ones response solutions. When there is a large cross over the yellow note, it means that the solution has already been implemented. The Figure. 6 defines the template for solution notes. When a solution is suggested, it is written on a yellow note and attached to the risk it relates to. Information on the note should include at least the suggestion and maybe a rough cost estimate in implementation time or in other resource.

The one on the left is an unattended solution and the one on the right is an implemented solution.



Risk table	Impact 1	2	3
Probability 1			
2			
3			
Avoided:			
Transferred:			

Figure 4: 3x3 risk board

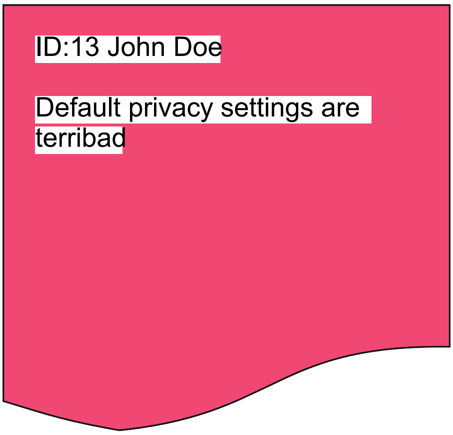


Figure 5: Risk note template

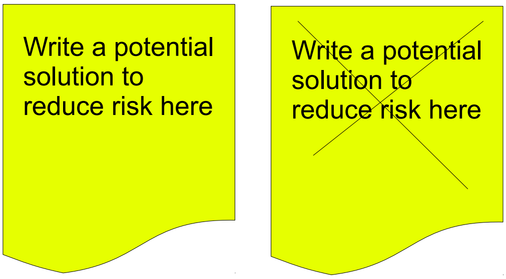

Write a potential solution to reduce risk here
Write a potential solution to reduce risk here

Figure 6: Risk response notes.

When a solution is crossed out, the red risk note should also be moved to a new position on the board. This is done to avoid confusion regarding current risks and to prevent one risk being moved twice, which might drastically affect the accuracy of the process. The strength of the risk board is that it makes risk management highly visible and hopefully easy to understand.

Checklists

To help in identification of high-risk components, checklists could be used. In this suggestion there are two checklists. One is based on the past experience and high-risk components. This list is more technical by nature. The other is based on security and abuse cases as Vähä-Sipilä suggests [6], and it is used as a reference card for general security requirements. One checklist should be based on past experience or already existing listing of a high risk code. It is not always easy to tell what feature is a high-risk one, thus all features should receive a quick risk identification. It could be in the form of a brainstorm or some other quick and light-weight method. The checklist that includes high-risk components should be updated from time time. Security and abuse cases could be another checklist, which sets the general requirements for feature security. As Vähä-Sipilä suggests, the security and abuse cases are a great way to approach security from an outer perspective. Security cases approach the security from a users' perspective, so it is important to have a clear vision about the users' security needs. The baseline for security is set during planning phase. The idea is that the security and abuse cases are not booked as backlog items. They are used as a general reference that is compared to every feature. The person responsible for implementation of the feature should ask:

- What needs to be done in order to meet client's security requirements?
- Does the feature meet the requirements at the end of the sprint?

The latter could be part of the sprint DoD. In order to use security and abuse cases effectively, the team must have understanding of the security requirements and potential attackers. This allows them to direct risk identification to correct software components.

Process

PMBOK defines six different phases in risk management [13]. The flowchart in Figure 7 presents the idea how risks are identified, assessed and approved in the model.

[13] see Project Management Institute, 2008

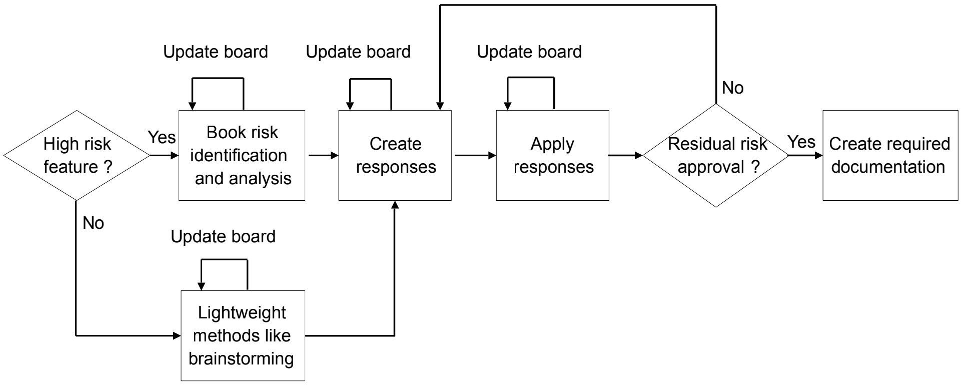


Figure 7: Flowchart of the process

The flowchart does not include risk planning, which is only done once as a part of project visioning. It also doesn't include risk monitoring. The table 1 describes what additional things are done during the Scrum meetings. This does not mean that risk identification, assessment and responses would only be done during these meetings, most of the work will be done between the meetings. "every time" means that the activity should be done every time, while "possible" means the activity is possible if needed.

Risk management in meetings	Identification	Assessment	Create response	Apply response	Risk approval
Sprint planning	every time	every time	every time		
Daily Scrum	possible	possible	possible		
Sprint review	possible	possible	possible		every time
Sprint retrospective					

Table 2: Additional risk management activities in meetings.

Risks can and should be identified during all meetings apart from retrospective. As a risk is identified, one might also suggest the impact and probability it might have on the software. If a quick assessment is not done, then the risk will be attached to the side of the risk board, not inside the risk board. The reason is to avoid confusion. Even if a feature receives a quick risk identification during sprint planning, this still means that further identification and assessment should be done by the person responsible for the feature. Identifying and assessing new risks during the sprint review is possible, but applying any responses will be a task for the next sprint. Applying the chosen responses is done between the meetings. The sprint retrospective is a meeting where the team will talk about what went well and what can be improved in the risk management process, rather than the actual risks. It is possible to identify problems or risks in process itself, but actual product security should not be a topic here. The retrospective is used to improve the process itself, not to talk about actual security issues.

Planning

This is only done once during the project lifetime. While it is not possible at this point to tackle the actual security issues in the software, this is still an important part of risk management. The planning is done when establishing a vision about what the software should be like. From security and risk management perspective it is important to set requirements and goals for security. Security training is an important part of creating secure software. Security training and methods should also be discussed before starting to develop the actual software. Everyone involved with the software should receive security train-

[13] see Claude, 2009 and Wijewardena, 2009

ing, including everyone in the team, product owner and scrum master. How the actual training is done is out of the scope for this thesis. The people responsible for accepting high-level risks should receive an adequate amount of security training in order to make proper decisions. General level of required security should be decided here. Based on who will use the software and what is it used for, one should get a good idea of the general level of security requirements. If the software truly requires high level of security, then attacker profiles could be created. Attacker profile includes attackers' motivation and resources. For example, an insider would have physical access to hardware and an existing user name and password. Someone working for government could have very high budget. Attackers working for organized crime would be after financial gain and relatively well funded. These profiles need to be created if the developers want to make the most out of abuse cases. Depending on the security requirements, general security level wanted and attacker profiles the teams can adjust the time used on risk management and guide the risk identification to items that are most likely attacked. The definition of done it should be discussed here. The question is, when is a risk so high that it prevents the deliverables from being shipped? This obviously depends on the security requirements of the software and will vary from project to project.

Risk identification, assessment and response

This is maybe the most crucial part in the process. There are plenty of methods for identifying risks, as presented in page 30. In addition there are other methods tailored specifically for identifying security risks. All features in the backlog deserve at least a quick risk identification. However, more in-depth identification and analysis should be done to those features that are considered having an elevated risk level. Just by reading the feature it is not easy to tell whether the code is high risk. The technical checklist should be used to identify high-risk components, which might require a comprehensive risk identification and assessment.

Risk management should officially be part of at least sprint planning and review. New risks should be identified when choosing the features the team will commit to. This is also great timing for brainstorming, since all the members should attend. If risk management is added to the agenda of these meetings, then the meetings will be slightly longer. Risks can be identified during any Scrum meeting, however daily scrum being only maximum of 15 minutes, it might end up consuming a large portion of the time. This is, however good time to seek consultation since the whole team is gathered. Sprint retrospective can be used to hone the process itself. Risk identification and assessment The person responsible for implementing the feature is also responsible for making sure it receives risk identification, and if risks are found, assessment and risk response implementation. As soon as a risk is identified, it is written on a red note described in Figure. 7 and placed to the risk board. The same is done with risk solutions, however, yellow note is used and it is attached to risk it potentially can solve or mitigate. When a feature is consideras a high-risk, it means a new task is booked for the risk identification and analysis. This task is added to the sprint backlog. This is done before starting to work on the actual feature itself. It should be done by the same person or people who are responsible for implementing the feature. The methods for identifying and assessing the risks can be more formal and comprehensive than brainstorming. Such methods include for example STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial-of-Service, Elevation of privilege), DREAD (Damage, reproducability, exploitability, Affected users, discoverability) or Trike (see OWASP 2010). Expert consultation should also be considered, depending on the experience and field of expertise of the people assigned responsible of the feature and nature of the feature.

Risk response

The person responsible for a feature is also responsible for applying necessary risk responses, meaning he or she is the risk owner. The lack of clear risk ownership was considered as a problem during the interviews. Choosing which responses to apply is a crucial part of effective risk management. The following should be considered when creating and applying risk responses:

- Size of the risk.

- Risk response should be in proportion to the risk and feature.

- How important is the feature the risk is related to? There might be large and problematic risks related to a feature which is not high priority. Creating and applying responses would take a large amount of time to complete a feature which is not all that important.

After a response has been applied, the note will be crossed over and the risk itself is moved to a new spot on the board. In a case where there are several responses to a risk, it is up to the same person to decide which ones are implemented and which ones are ignored. The best response depends on the security requirements of the software and the efficiency, resources versus mitigation, of the response. If in doubt, the person can consult with experts, scrum master, product owner or the team.

Risk approval and monitoring

One of the problems that came up during the interviews was that it was not always clear who is responsible for accepting residual risks. Here is one possible solution for the problem. When accepting risks two things need to be considered, the nature and size of the risk.

At this point all risks and applied responses should be visible on the risk board. The board tells how high is the residual risks after the crossed over responses have been applied. The idea is, that if residual risk≤2 then the person who implemented a feature and managed its risks can approve the residual risk. In case of a residual risk the product owner must approve the risks. The product owner should should receive explanation regarding any risks greater than 2. The reason for this is that it is the product owner's responsibility to decide whether the new increment goes out. The risks should be explained to the product owner on feature level, meaning how the feature might malfunction and what is the chance for it actually happening. This should address an issue which came up during the interviews; some managers do not have the technical understand to make decision based on technical details. The risks should be presented to and accepted by product owner during the sprint review. If the product owner feels like he or she can't make the decision about accepting the risk, then the next approval depends on the organizational structure. It could be internal or external client, business decision maker or SoS-board (Scrum of Scrums). Problem with getting approval from them is that it is unnecessary bureaucracy, which is something lean software development should avoid. In case security risks, the ownership should follow the code in case security problem arises. If the residual risk can't be accepted, the feature will not go out. When this happens it means new risk responses must be applied.

When the risk has been accepted, there is no longer any reason for it to be on the board. The accepted risks should be stored in a digital form for potential future use and re-evaluation. Nyfjord & Kajko-Mattsson suggested a wiki platform to replace the risk management forum [11]. It would also create new possibilities when comparing risk management efficiency and practises between scrum teams. Every team should have their own risk register. This can be done once per sprint, after the product owner has seen and accepted all the risks. The larger risks accepted should be re-evaluated from time to time. The re-evaluation should examine at least:

- Total risk, has the probability or impact gone up or down?
- Are there any new ways to mitigate or avoid the risk?

The re-evaluation should be performed on constant intervals and if possible should be done by the person who owns the risk. The re-evaluation should be booked on the product backlog.

Roles and communication

There are three crucial roles in this model. These are the product owner, team and risk owner. In risk communication other scrum teams and business decision makers should also be kept up-to-date. All the roles are required to make risk management effective.

[14] see Nyfjord & Kajko-Mattsson, 2008

Product owner's role is being especially important if the team is also using SDLagile. SDL-agile requirements should be booked on product backlog to avoid creating invisible overhead. In this model only additional activity booked on the sprint backlog is risk identification and assessment for high risk features.

Product owner also acts as a gatekeeper for quality. At the end of each sprint new features and risks are presented during the sprint review. In the end, in Scrum it is always the product owner who decides if a feature is complete and does it meet the requirements. This means that the risks should be presented in a way that the product owner will understand them. Time should also be considered when presenting the risks to the product owner, meaning that the risk owner who is presenting the risk shouldn't go too deep in technical details.

A scrum team is a self-organizing team. This means they will decide among themselves how to split the features and tasks between team members. Person's background and field of expertise should be considered when handing out high risk features and related risk analysis and assessment. Risk communication within the team should happen without much effort, since risks are discussed during sprint planning, review and occasionally during daily scrum. Risk board is also visible to the whole team.

Risk communication between scrum teams is also important. This could be done during SoS-meeting. The threshold for presenting a risk in the SoS-meeting should be based on other criteria than just size of the risk. The thing that should be considered before presenting a risk in SoS is: could other teams be suffering from the same risk? If risk root cause analysis shows that a risk is caused by bad architectural design, then the team responsible for architecture should be informed.

The scrum master and product owner should handle communication between the scrum team and business decision makers. One of the key principles in Scrum is that the scrum master should handle problems and communication with people who aren't part of the team. Security risks which are often technical in nature shouldn't really be part of business decision makers risk management activities, except in a case where a large security risk could affect long-term business decisions.

References

Project Management Institute inc. 2008. PMBOK: A Guide to the Project Management Body of Knowledge, fourth edition.

Microsoft. SDL-Agile requirements. 2012a. [Cited 11.13.2013]. Available at: http://msdn.microsoft.com/en-us/library/ee790620.aspx

Microsoft. SDL-Agile tasks. 2012b. [Cited 11.12.2013]. Available at: http://msdn.microsoft.com/en-us/library/ee790615.aspx

Microsoft. SDL-Agile every sprint requirements. 2012c. [Cited 11.12.2013] Available at: http://msdn.microsoft.com/en-us/library/ee790610.aspx

Microsoft. /GS buffer check. 2013a. [Cited 11.12.2013]. Available at: http://msdn.microsoft.com/en-us/library/8dbf701c(VS.80).aspx

Vähä-Sipilä A., Product Security Risk Management in Agile Product Management. 2010. [Cited 11.12.2013]. Available at: http://www.owasp.org/images/c/c6/OWASP_AppSec_Research_2010_Agile_Prod_Sec_Mgmt_by_Vaha-Sipila.pdf

Nyfjord J, Kajko-Mattsson M. 2008. Outlining a Model Integrating Risk Management and Agile Software Development. SEAA '08. 34th Euromicro Conference

Claude J. Agile risk board, a way to manage risks in agile enviroment. 2009.

[Cited 11.12.2013]. Available at: http://www.agile-ux.com/2009/07/23/agile-risk-board/

Wijewardena T. Risk Management – where it fits in Scrum?. 2009.
[Cited 11.12.2013]. Available at:
http://projectized.blogspot.com/2010/02/risk-management-where-it-fits-in-scrum.html

Microsoft, SDL-Agile high risk code. 2013b. [Cited 11.12.2013]. Available at:
http://msdn.microsoft.com/en-us/library/ee790613.aspx

OWASP, Threat risk modelling. 2010. [Cited 11.12.2013]. Available at:
http://www.owasp.org/index.php/Threat_Risk_Modeling#Identify_Security_Objectives[13]

Cohn M, scrum of scrums. [WWW]. [Cited 30.11.2010]. Available at:
http://www.scrumalliance.org/articles/46-advice-on-conducting-the-scrum-of-scrums-meeting

First Steps to Consider Privacy

Ari Pietikäinen, Ericsson
Jouko Ahola, Ericsson

Executive summary

Privacy issues and personal data protection has become recently one of top concerns in Europe and United States and interest is increasing also in Far East area. Main reasons for these are recent large scale data leakages and significantly raised privacy awareness due to those leakages. Privacy has become as a basic human right and is recognized as a necessity by the EU (Privacy Directive) and the OECD (Principles of Fair Information Practice). This development has initiated regulators and governments to set new more strict privacy and data protection legislations within e.g. EU and US. Breaking those laws and regulations, especially if those leads to personal data breaches, tends to have meaningful business impacts in terms of financial fees, penalties and direct or indirect business losses. That is also the main reason why enterprises should include privacy in their risk management process as integral part to avoid any unacceptable direct risks to their business and brand.

Introduction

Target audience

Management of all size enterprises (incl. CTO, CIO, CISO, CPO)

What you learn from this chapter

>> Why privacy and personal data protection matters
>> How you can identify if privacy and data protection risks are valid and requires further analysis regarding to your business
>> What are the main initial steps any enterprise shall take to manage privacy issues

Concepts

One most often heard questions related to privacy are: "how do you define privacy?" and "what data should be considered as personal data?" For the first question there is no single widely accepted answer but in ISACA defines it as the "Freedom from unauthorized intrusion or disclosure of information about an individual." To find clear answer to the second question is even more difficult, because different countries are having different specifications, regulations and laws with even more variable interpretations. However, European Union has ended up for the following widely used definition: "Personal information shall mean any information that can be related to an identified or identifiable living natural person ('data subject'); an identifiable person is one who can be identified, directly or indirectly, in particular by reference to an identification number or to one or more factors specific to his physical, physiological, mental, economic, cultural or social identity." It is also good to remember, that the term "natural person" or "data subject" can be for example an employee of an enterprise, end user of a service enterprise provides or mobile phone subscriber, etc. Regardless of the ways personal data is collected, processed, used, transferred and disclosed to 3rd party or the purposes behind, i.e. what are the primary and secondary reasons to manage personal data, the ultimate target of privacy laws and data protection regulations is to protect that data from leakage and/or misuse of it. The main benefits of successful privacy and data protection controls for natural person is to have choice what personal data is collected, how and for what purposes it is used and to whom that data it is disclosed.

How to identify, personal data? The good starting point here is to make inventory of all the data enterprise either collect itself or just store, process, use or transfer on behalf of another company, customer, collaborator, association or end user. Here any widely accepted definition of personal data is good reference regardless of the data format or the purpose of the data existence. Typical examples of personal data are for example: Name, Address, Phone Number, IP Address, Account information, Call history, Trace logs, Government-issued identification numbers (e.g., Social Security Number), Billing information, Employee Files, including evaluations and disciplinary actions, Membership in Professional or Trade Association, or Trade Union, Financial information, such as income, credit or loan records, Medical Records, Criminal records, Racial/Ethnic Origin, Sexual Preferences, Religious or Philosophical Beliefs or Affiliations, Political Opinions, Photographs. In some (or many?) cases one data record does not alone identify an individual because they are attributes shared by many people. However, they are potentially personal information, as they may be combined with other personal information to identify an individual. Examples of this are name and address.

Analysis of personal and related requirements

After any personal data has been identified then further analysis is needed. The main purpose of this basic analysis to find out e.g. who owns the data, what is purpose and legal base of storing, processing, using, deleting or transferring those data records? This initial analysis would already clarify if your role here for the data is either "Data Controller" or "Data Processor". The Data Controller owns the data and is ultimately responsible to manage data in adequate way. The "Data Processor" processes personal data on behalf of Data Controller. Data Processor has also many responsibilities, but not at same extent as Data Controller is. This "role identification" is important in order to understand legal requirements, etc. Most organizations are lacking the needed knowledge and competencies to make complete privacy/data protection analysis in comprehensive way. This is because there is need to go much deeper in analysis to find out for example:

- How sensitive personal data is from privacy point of view – not only security
- Is there need to comply with any specific standards and regulations like PCI DSS (credit card data), SOX (Finance), HIPAA (Health care) or Telecom
- Is the data used for any secondary purposes like direct marketing, service development, etc.
- How long data is needed to retain
- Is the data transferred to other countries and if so, between what countries
- What are the actual data flows and where data is processed to recognize which local (like Canada, India, Germany, etc.) or continent (EU, US) specific laws and regulations are valid to comply with.

This task requires good subject area specific knowledge and should in most cases be given to privacy experts. There are various methodologies for this kind of analysis like formal Privacy Impact Analysis, Data Protection Audits or Compliance Assessments or more unstructured way to find out the needed details. Formal assessments and audits can be done either, as outsourced activity by 3rd party company, or as an internal self-assessment using publicly available check lists, if there are competent resources available within organization. The main output of such assessment or analysis is to identify possible business impacts and total risk to enterprise in same way as used to do in Security Risk Analysis. Each identified risks needs then to be mitigated to acceptable level. In many cases risk mitigation can be done by low cost solution like to stop collecting that part of information which enables identification of natural person or by anonymizing the data. However, in many other cases the solution may be very costly, like needing to implement privacy controls in all key applications, moving data center to other geographical location or establishing the needed legal frame works like "Safe Harbor" or "Binding Corporate Rules" to comply with mandatory regulation.

How to avoid unacceptable risks and how to achieve needed privacy maturity level

Activities mentioned in previous sections, like Privacy Impact Assessment or other analysis, are by nature more reactive than proactive and may cause extensive extra costs to organization. In too many cases Privacy Impact Assessments or Data Protection Audits takes place only after severe privacy incident has already taken place causing personal data breach with many direct and indirect damages to enterprise business and brand reputation. Privacy incidents, if they become publicly visible, including severe data breaches, often includes also regulator fees or other sanctions after legal process. The recommended proactive approach is "Privacy by Design", which means taking privacy business impacts into account at the very beginning of service, product or business case creation. This approach leans on same key corner stones as proactive implementation of Information Security Management Systems (ISMS/ISO 27001); people, processes and technology. People mean here clear privacy roles and responsibilities and systematic training programs. Or like in case of new products or services creation we have been using proactive "Security by Design" approach rather than correcting security weaknesses only as reactive activity after security incidents. Naturally this highly recommended proactive "Privacy by Design" approach, requires systematic work, which is usually done by initiating proper Privacy Program, which has to be managed by fully competent person. As an output of this Privacy Program, there will be many separate activities to establish all needed key elements like:

- Privacy strategy
- Structured Privacy Team
- Privacy policies and processes
- Metrics to measure privacy performance
- Privacy audits / assessments
- Data life cycle management
- Compliance management
- Incident response capability

Experiences and discussion

For writing this article I have used my practical experiences from implementing Privacy Management structures, building security/privacy teams, establishing Privacy by Design processes and initiating/delivering Privacy Impact Assessments in large Telecom Vendor Organizations like Nokia and Ericsson).

References

Finland

Tietosuojalaki: http://www.finlex.fi/fi/laki/ajantasa/1999/19990523

Sähköisen viestinnän tietosuojalaki: http://www.finlex.fi/fi/laki/ajantasa/2004/20040516?search%5Btype%5D=pika&search%5Bpika%5D=s%C3%A4hk%C3%B6isen%20viestinn%C3%A4n%20tieto-suojalaki

EU/ECC

Directive 95/46/EC on the protection of individuals with regard to the processing of personal data and on the free movement of such data (General Data Protection Directive).

Directive 2002/58/EC concerning the processing of personal data in the electronic communications sector (ePrivacy Directive) as amended by Directive 2009/136/EC

Directive 2006/24/EC on the retention of communications data by network and service providers (Data Retention Directive).

US

Sector-related privacy laws e.g. Children's Online Portal Protection Act (COPPA), Health Insurance Portability and Accountability Act (HIPAA)

IAPP Publications

Foundations of Information Privacy and Data Protection: A Survey of Global Concepts, Laws and Practices, 2012 European Privacy;Law and Practice for Data Protection Professionals, 2012 Privacy Program Management; Tools for Managing Privacy Within Your Organization

Standards

ISO/IEC 2nd WD 27018; Information technology – Security techniques - Code of practice for data protection controls for public cloud computing services

NIST Special Publication 800-53, Revision 4 Security and Privacy Controls for Federal Information Systems and Organizations

Security Metrics

Reijo Savola (VTT Technical Research Centre of Finland)
Christian Frühwirth (Aalto University)
Ari Pietikäinen (Ericsson Finland)
Timo Nyberg (Aalto University)

Executive summary

Measurement practices are a key success factor for future, systematic software development approaches. In order to be successful, such systematic approaches require sufficient and meaningful security, trust, and privacy metrics or measurements that enable developers, managers and users to obtain credible evidence of the security level or performance of the system.

This section provides practitioners:

1.) a simple metrics development approach and
2.) a method for aligning security metrics with security objectives

Target audience: SME management, Risk managers

Introduction

As software-intensive systems continue to get more connected, distributed and begin to enter 'the cloud' they increasingly face the ubiquitous threats associated with security breaches. Facing these threats with today's 'add-on' and 'patch-work' approaches to software security is both inefficient and costly for software vendors and customers alike. Hence, there is a combined need that drives the industry towards more systematic solutions to security issues in software engineering and software intensive systems. There are numerous challenges to overcome on the way: Architects who seek to integrate Security in Software Engineering and design processes recognize that security is a cross-cutting field and thus difficult to manage consistently and systematically throughout an entire software system's lifecycle. At the same time, the dynamic nature of security risks also brings up the need to be able to systematically a) measure and b) reason about security solutions separately.

"So what makes a good security metric?"

In a practitioner setting, desirable characteristics of security metrics are:

1.) consistency of measurement
2.) inexpensive to gather
3.) expression as a cardinal number or percentage an
4.) using at least one unit of measure.

In addition to the above, a good metric further needs to be "contextually specific", and "relevant enough to decision makers so they can take action" [1]. Hence, we argue that

1.) in order to be relevant to decision makers, metrics need to be linked to the organization's security objectives and
2.) in order to be able to take action, the metrics need to be in line with (i.e. executable by) the company's security capabilities.

Metrics Concepts overview

The terms security performance and level are commonly used in practice to refer to the security effectiveness of security solutions, the main objective of security work and solutions. In addition, security efficiency is essential because resources are often constrained. In order to be able to reason systematically about security effectiveness and efficiency, and their ratio, there is a need for an abstraction model to explicitly express what kind of solutions are designed and used. The concept of security controls can be used for this purpose. In addition to security effectiveness and efficiency, security correctness is a fundamental objective [2]. Correctness is a necessary but not sufficient requirement for effectiveness: it enables effectiveness. It should be a side effect of good security, not its driver. There are various factors which enable security effectiveness of the SuI: configuration correctness, correct design, implementation and deployment of security controls and proper security assurance and testing activities.

The term security metrics is misleading, since complexity, limited observability, a lack of common definitions and the difficulty of predicting security risks make it impossible to measure security as a universal property. However, measured data does not need to be perfect, provided that it contains the information required, is adequately correct and practically measurable. In context, we employ the most widely used term, security metrics.

As security metrics are challenging to develop, it is important to associate security metrics with metric confidence, an assessed value depicting the metrics developer's confidence in it. The actual measurement results and the metric confidence together indicate security confidence [3], i.e. the belief that the SOs are met.

Concept	Explanation	Reference
Security Objective (SO)	High-level statements of intent to counter identified threats and/or satisfy identified security policies and/or assumptions.	[ISO/IEC 15408]
Security Requirement (SR)	Requirement, stated in a standardized language, that is meant to contribute to achieving the SOs.	[ISO/IEC 15408]
Security Control (SC)	Means of managing risk, which can be administrative, technical, management, or legal in nature.	[ISO/IEC 27000]
Security correctness	Assurance that security controls have been correctly implemented in the SuI, and the system, its components, interfaces, and the processed data meet the security requirements.	[Savola 2009]
Security effectiveness	Assurance that the stated SOs are met in the SuI and the expectations for resiliency in the use environment are satisfied in the presence of actual security risks.	[Savola 2009], [IT-SEC 1991]
Security efficiency	Assurance that the adequate security quality has been achieved in the SuI, meeting the resource, time and cost constraints.	[Savola 2009], [IT-SEC 1991]

Table 1: Key concepts of the study

[1] See Jaquith, 2007
[2] See Savola, 2009
[3] See Kanter, 2004

An iterative process to develop security metrics

A simple, yet effective process for security metrics development is the used of iterative stages that decompose the security objectives, originally described by [4]. Figure 1 illustrates the process in an industry setting.

Description: 2

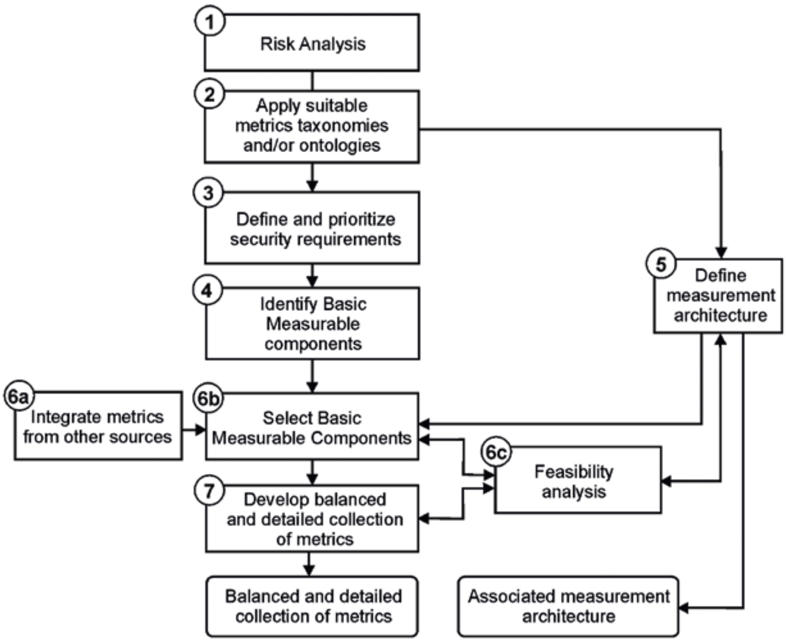

1
Risk Analysis
2
Apply suitable metrics taxonomies and/or ontologies
3
Define and prioritize security requirements
4
Identify Basic Measurable components
5
Define measurement architecture
6a
Integrate metrics from other sources
6b
Select Basic Measurable Components
6c
Feasibility analysis
7
Develop balanced and detailed collection of metrics
Balanced and detailed collection of metrics
Associated measurement architecture

Figure 1: A simplified security metrics development approach based on [Savola and Abie 2010]

Using an iterative process for metrics development enables stakeholders to produce

1.) a balanced and detailed collection of security metrics, and
2.) associated measurement architecture.

In the following, the stages are referred by SMDn, where n is the stage identifier. Fig. 2 illustrates a highly simplified example of decomposition of the main objectives related to the effectiveness of authentication functionality.

Description: 3

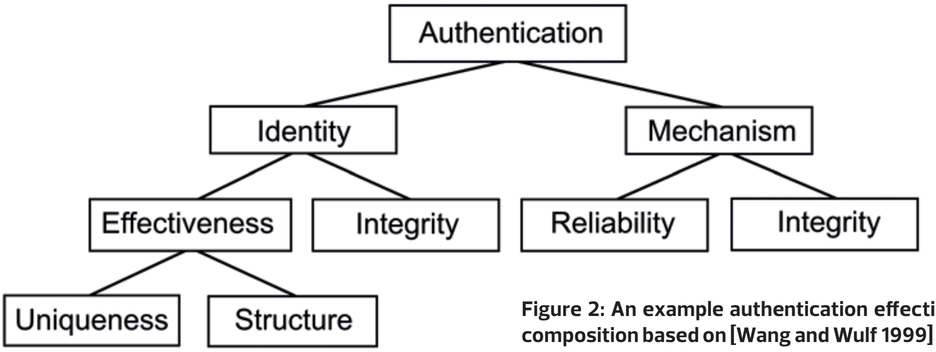


Figure 2: An example authentication effectiveness decomposition based on [Wang and Wulf 1999]

[4] See Wang and Wulf 1997

Basic Measurable Components (BMCs) are leaf components of a decomposition that clearly manifests a measurable property of the system [5]. The BMCs of Fig. 4 are: Authentication Identity Uniqueness (AIU), Authentication Identity Structure (AIS), Authentication Identity Integrity (AII), Authentication Mechanism Reliability (AMR) and Authentication Mechanism Integrity (AMI) [6].

In practical systems, authentication objective decomposition may easily consist of tens or hundreds of sub-nodes, because security configuration correctness and security testing metrics in particular incorporate a lot of details in various infrastructure objects and security protocols.

A workshop method to align metrics with measurement objectives

Organizations that approach information security management from a business perspective, invest in using security metrics to measure the degree to which their security objectives are being met.

The decision however, on which particular security metrics to use, is surprisingly often based on an uninformed process and disregards the company's security goals and capabilities. Existing frameworks such as the SSE-CMM or ISO 27000 series provide generic guidance on choosing security objectives and metrics, but lack a method to guide companies in choosing the security metrics that best fit their unique security objectives and capabilities.

The method presented in this section can act as a tool to determine which metric is 1.) efficient to apply using a companies given capabilities and 2.) provides the maximum contribution to the company's security objectives. The method is supported by existing research in the field of value-based software engineering and has been developed based on the established "Quality Function Deployment" (QFD) approach.

Security objectives and security capabilities

Existing information security frameworks, such as the ISO 27000 series, provide companies with excellent support in defining their security objectives. Traditional security objectives typically include the assurance of confidentiality, integrity and availability of an organization's information assets in a value-neutral form [7]. This section focuses on business-driven security objectives because they enhance traditional objectives with the concept of value. The following example illustrates the difference between value neutral and business driven objectives:

A value neutral security objective: "Protect the integrity of the data warehouse."

A business-driven security objective: "Use 60% of security resources for protecting the integrity of the data-warehouse (and 20% for confidentiality and 20% for availability").

If the introduction of a security metric should not be a waste of money, the metric needs to contribute in some form to the company's security objectives. A metric can contribute to a security objective by strengthening capabilities that are employed by the company to achieve the objective. For example: A metric that measures the number, type or source of malformed data packages received at a network node over time will improve the company's capability to conduct security awareness trainings (by better targeting the training lessons towards real world threats). In return, improved security awareness training is likely to reduce the number of security incidents that need to be fixed by the resident IT department, thus contributing to the objective of lowering incident management costs.

The extent to which a metric can contribute to an objective is determined by the 'fit' of the metric with the objective and by how 'well' it can be applied.

The ability to apply a security metric is determined by the company's individual capabilities. The 'how

[5, 6] See Savola and Abie, 2010

[7] See Neubaner, 2005

well' is measured by the range of results that can be expected from the application. This is in line with a similar description of "process capabilities" by Paulk in the Capability Maturity Model (CMM) [8] Consequently, we argue that a company's security capabilities represent the connection, or alignment mediator between security metrics and objectives. Thus, in order to determine the most suitable metric from a set of candidates they need to be evaluated in the light of these given capabilities.

Figure 3 summarizes this concept and illustrates the differentiation between well-aligned and misaligned security metrics with the example of three different metric candidates.

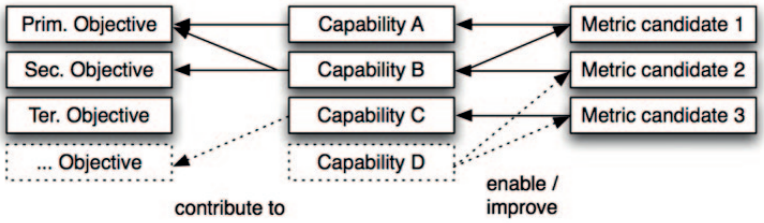


Figure 3. Relation between security objectives, capabilities and metrics; Differentiating well-aligned from misaligned metrics

Metric candidate 1 in Figure 1 is enabled by capability B and contributes to the primary security objectives through improving capability A. Thus we consider metric candidate 1 to be well aligned.

Metric candidate 2 on the other hand is less well misaligned: Even though it is expected to strengthen capability B, which contributes to two objectives, the metric candidate's application requires the, currently unavailable, capability D.

Metric candidate 3 also requires the same unavailable capability D and only improves capability C, which does not contribute to the top security objectives. Thus we consider candidate 3 to be misaligned.

If a company had to decide which of the above 3 security metric candidates to introduce or spend money on, they would probably be quick to rule out candidate 3, but remain less clear on the choice between metric candidate 1 and 2.

How can we support this type of differentiation in a structured process and make it available in a form that supports a company's metric decision?

Structured alignment process and Quality function deployment (QFD)

Alignment processes for the purpose of connecting capabilities with business objectives have been discussed in the literature on software process improvement and value based software engineering. Oza et al [9] developed a suitable solution using the Quality Function Deployment (QFD) approach. QFD structures links between information by creating alignment matrices, called 'Houses of quality' (HoQ) [10]. QFD has been used successfully t in production planning by the Toyota auto company [11] and authors like Richardson et al [16], Liu et al [12] and Oza demonstrated that it can be adapted for new domains like software engineering. In the security context Kongsuwan [13] and Mead [14] recently analyzed its application in optimizing the quality level of information security and security requirements elicitation.

[8] See Paulk, 1995

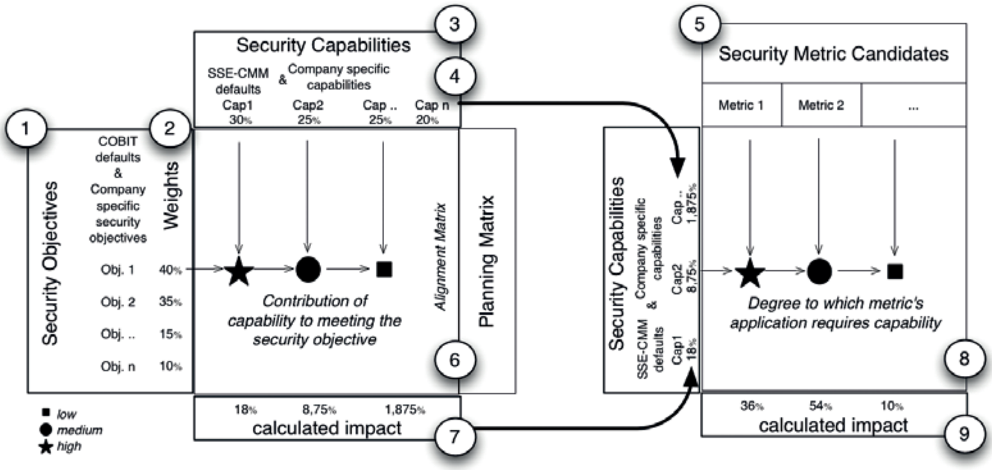


Figure 4: Alignment Method overview. From left to right: Matrix1: Security objectives against capabilities; Matrix2: Security capabilities against metric candidates

We chose to use QFD as basis for the alignment process because of these authors' experiences, its ease of use and proven track record in the industry.

Method Description

In this section we introduce the proposed alignment method and illustrate its usage. The goal of the method is to construct two alignment matrices, where the first matrix aligns security objectives against security capabilities, and the second matrix security capabilities against security metrics. Figure 4 shows a schema of the first matrix on the left side and the second matrix on the right.

The method is executed in 10 consecutive steps, which are divided into 3 phases: 1.) Preparation, 2.) Performing the Alignment and 3.) Analysis. The circled numbers in Figure 2 mark the elements of the alignment matrices that are created during the 10 steps.

The method is ideally executed in a workshop-type setting, where the main stakeholders of the company's security projects are present. The participation of these stakeholders ensures maximum leverage of domain specific knowledge during the alignment process. The following describes the process of constructing and filling the matrices.

Preparation Phase

The purpose of the preparation phase is to elicit and weight the three input-components of the method: security objectives, security capabilities and metric candidates.

Step 1: Elicit Security objectives
The participating stakeholders (or more generally, the user of the method) are presented with a default list of security objectives and asked to expand the list by brainstorming objectives that are specific to their company's context. Most companies will have several security objectives in common (e.g. "protect the customer data from unauthorized access"), thus using a pre-selected list of default objectives reduces the workload on the workshop participants and frees resources to focus on security objectives that are specific to the company (e.g. "minimize the amount of unencrypted communication between on- and off-site staff"). The "Control Objectives for Information and related Technology" (COBIT) framework [Cobit] is used as basis to create the list of default objectives.

Step 2: Weight Security objectives
The combined list of objectives (defaults + context specific security objectives) then undergoes a two-

[9, 10] See Oza, 2008

[11] See Hauser, 1996

[12] See Liu, 2005

[13] See Kongsuwan, 2008

[14] See Mead, 2006

staged reduction process, which consists of a voting and weighting procedure based in parts on Boehm's collaborative requirements negotiation approach [15].

First, each participant is given a fixed number of votes and casts them on the security objectives he or she perceives as most important. The list of objectives is reduced to the top 10 objectives and the rest is discarded.

In the second stage, each participant is asked to distribute a total of 100 points among the remaining objectives, with the more important ones receiving more points. The standard deviation of points assigned to a single objective by different participants is used as indicator for the degree of consensus among the participants about this objective's relative importance. The workshop moderator uses this information to identify and resolve disagreement about the importance of individual security objectives.

The resulting list of weighted security objectives is placed on the Y-axis of the first matrix, as depicted on the left part of Figure 3.

Optional: Benchmark security objectives:
Stakeholders may rate their company's planned and current performance in fulfilling each of the selected security objectives (target/actual comparison). The rating uses a scale from 0 to 5, similar to CMMI levels, where 0 refers to not meeting the objective at all and 5 to meeting it to the fullest extent. The benchmark is based on QFD's planning matrix and not part of the alignment process or matrix output in our method, thus considered optional. It can be useful however to visualize which objectives require the most improvement efforts and which may remain unchanged.

Step 3: Elicit security capabilities
The elicitation of security capabilities follows the same process as the security objectives in Step 1. The users are presented with a default list of capabilities generated from the Systems Security Engineering Capability Maturity Model (SSE-CMM) framework [16] and asked to expand it with company specific capabilities. After expansion, the list undergoes the same voting and weighting procedure as in Step 2.

Step 4: Weight security capabilities
The remaining capabilities are weighted using the same 100-point system described in Step 2, with stronger capabilities (i.e. the expected results from using these capabilities are above average expectations) receiving more points than weaker ones. The resulting list of weighted security capabilities is then placed on the X-axis of Matrix 1, and the Y-axis of Matrix 2.

Step 5: Elicit Security metrics candidates
The set of security metric candidates that are considered by the company are placed on the X-axis of Matrix 2. The composition of a candidate set can either be determined on a case-by-case basis or drawn from examples in the literature, like [17].

Perform the alignment

Steps 1 to 5 completed the frames of the two matrices (see Figure 4): Matrix 1 with security objectives as rows and capabilities as columns; Matrix 2 with the same security capabilities as rows and metric candidates as columns. The cells in the matrices will now be filled with "Low", "Medium", "High" or "None" alignment symbols (depicted in the legend of Figure 4).

Step 6: Align security objectives with capabilities
Each cell in matrix 1 is filled with one alignment symbol 'impact' that refers to the impact of a security capability on the company's ability (columns) to meet a particular security objective (rows). For example: how much does the capability to "Log data-access policy violations on firewalls" contribute to fulfilling the objective of "ensuring customer data base integrity". The process of filling the matrix with impact symbols follows the previous weighting of the security objectives and capabilities, with the more important objectives and capabilities tackled first. The filling process is performed by the method users (e.g. the workshop participants).

[15] See Boehm, 2001

[16] More information on the Systems Security Engineering – Capability Maturity Model (SSE-CMM) is available online at http://www.sse-cmm.org

Step 7: Calculate capability rating

After the 1st matrix is completed, the cumulative impact on security objectives is calculated for each capability:

```
For each Capability(ii){
 Capability(ii).impactSum += (Objective(i).weight *
 Capability(ii).impact(i) * Capability(ii).strength)
}
```

The value of Objective(i).weight and Capability(ii).strength are percentages and were determined in the weighting processes of Step 2 and 4. The value of Capability(ii).impact(i) is determined by factors assigned to each of the three alignment symbols (e.g. 4 for "High", 2 for "Medium", 1 for "Low" and 0 for "None"). The value of the individual factors depends on the company and their specific environment, thus experts of the local domain should be used to set them.

Step 8: Align security capabilities with metrics

The previously identified security capabilities are arranged against the metric candidates. The calculated cumulative capability impact 'impactSum' from Matrix 1 is used as capability weight on the Y-axis of Matrix 2. The alignment is performed similar to Matrix 1, by filling the matrix cells with Low, Medium, High or None alignment symbols. In Matrix 2 however, each cell is filled with two, alignment symbols to reflect the two-way relationship between capabilities and metrics described in Figure 3. The first symbol 'capRequirement' refers to the degree to which the application of a security metric candidate (rows) requires, or benefits from, a particular capability (columns). (E.g. how much does the metric "Logging coverage in % of hosts" [17] require or benefit from the capability to "Log data-access policy violations on firewalls".) The second symbol 'capContribution' refers to the potential contribution of a metric candidate on strengthening a particular capability.

Step 9: Calculate metric candidates rating

After the 2nd matrix is completed, the cumulative alignment score of the metric candidates are calculated:

```
For each MetricCandidate(iii){
 MetricCandidate(iii).alignmentScore += ( Capability(ii).strength *
   MetricCandidate(iii).capRequirement(ii)) *
  ( Capability(ii).impactSum * MetricCandidate(iii).capContribution(ii))
}
```

The factors of for the symbols in MetricCandidate(iii).capRequirement(ii) and MetricCandidate(iii).capContribution(ii) are determined as in Step 7.

Step 10: Analysis

The first matrix helps the company to determine which capabilities have the strongest impact on reaching the set security objectives (see Step 7 in Figure 4). The second matrix enables an evaluation of the metric candidates' capability requirements and their expected contribution to the security objectives (see Step 9 in Figure 4). Higher alignment scores indicate that a metric candidate is more likely to benefit from existing, strong capabilities and contribute to the improvement of capabilities that support the company in achieving their security objectives. A sorted list of metric candidates based on their scores represents an important decision support tool on which metric(s) are the most suitable alternative(s) for introduction in the company. In addition to the prioritized list of metric candidates, different patterns of symbols in both matrices allow for more complementary analysis. (e.g. determining the number of capabilities with a high contribution to a particular security objective vs. objectives that are not supported by at least 1 capability).

[17] For more information see [Jaquith 2007] pp.56, table 3-3 (2)

References

Boehm, B., Grunbacher, P., Briggs, R.O. ''Developing groupware for requirements negotiation: lessons learned,'' Software, IEEE , vol.18, no.3, pp.46,55, May 2001.

COBIT. Information Systems Audit and Control Association, 2012. Available: http://www.isaca.org/CO-BIT

Hauser, J.R. and Clausing, D. The house of quality. IEEE Engineering Management Review 24, 1 (1996), 24–32.

Information Technology Security Evaluation Criteria (ITSEC), Version 1.2, Commission for the European Communities, 1991.

Jaquith, A. Security Metrics: Replacing Fear, Uncertainty, and Doubt. Addison-Wesley Professional, 2007.

Kanter, R. M., ''Confidence: Leadership and the Psychology of Turnarounds,'' Random House, London, 2004.

Kongsuwan, P., Shin, S., and Choi, M. Managing Quality Level for Developing Information Security System Adopting QFD. Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD'08. Ninth ACIS International Conference on, (2008), 19–24.

Liu, X.F., Sun, Y., Kane, G., Kyoya, Y., and Noguchi, K. QFD application in software process management and improvement based on CMM. Proceedings of the third workshop on Software quality, (2005), 6.

Mead, N.R. and INST, C.U.P.P.S.E. Experiences in Eliciting Security Requirements. (2006).

Neubauer, T., Klemen, M., and Biffl, S. Business process-based valuation of IT-security. Proceedings of the seventh international workshop on Economics-driven software engineering research, ACM (2005), 1-5.

Oza, N., Biffl, S., Fr\ühwirth, C., Selioukova, Y., and Sarapisto, R. Reducing the Risk of Misalignment between Software Process Improvement Initiatives and Stakeholder Values. Industrial Proceedings of EuroSPI, (2008), 6–9.

Paulk, M.C. and Paulk, M.C. The capability maturity model: Guidelines for improving the software process. Addison-Wesley Reading, MA, 1995.

Savola, R., ''A Security Metrics Taxonomization Model for Software-Intensive Systems,'' Journal of Information Processing Systems, Vol. 5, No. 4, Dec. 2009, 197–206.

Savola, R., Abie, H., ''Development of Measurable Security for a Distributed Messaging System,'' Int. Journal on Advances in Security, 2(4), 358–380.

Wang, C., Wulf, W. A., ''Towards a Framework for Security Measurement'', 20th National Information Systems Security Conference, 522–533.

Fuzzing

Juho Myllylahti,
Oulu University Secure Programming Group

Pekka Pietikäinen,
Oulu University Secure Programming Group

Executive summary

Fuzzing is an effective and practical form of fault-based testing which is especially popular in security and robustness testing. Fuzzing tools widely used by security researches and other parties who try to uncover vulnerabilities in widely used real world software. This chapter presents the general idea, terminology and tries to give insight on how to combine fuzzing and agile development practices.

Concepts

Genesis

Professor Barton Miller coined the term fuzzing, and there is an interesting backstory on how he first got interested about the idea in late 80s. In his website [1], he tells how he was logged into a remote UNIX system one night, trying to work over a modem connection whilst a thunderstorm was raging in Wisconsin. The storm caused noise on the phone line, distorting the input he was sending to the server. During this he noticed, that the programs he was using – common UNIX utilities – seemed to crash frequently and he was racing time to write the commands before the storm could intervene and cause the utilities to crash. This experience made him curious of how common of a problem was this type of program instability. The result? Very common: they could crash or hang 25–33% of the tested UNIX utilities of seven different UNIX variants – a respectable figure considering many of those utilities were very mature and widely used pieces of software, written by very talented developers. Barton and his colleagues later reproduced their tests in 1995, 2000 and 2006 testing X-windows system, Windows DLLs and Mac OS X – still finding bugs.

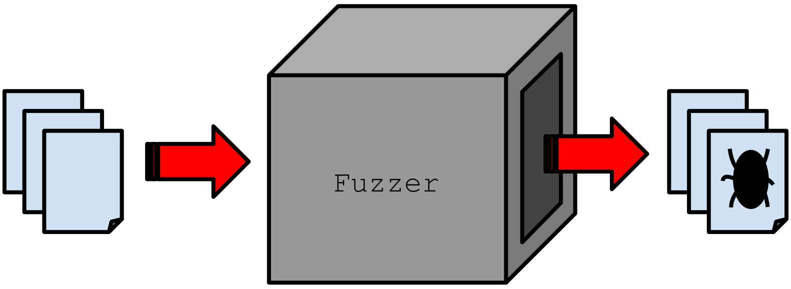
The general idea

The general idea of fuzzing can be described in many different styles, formally and less formally. A formal way of describing fuzzing could be that it is a search problem where we try to find faulting inputs (and thus bugs they reproduce) from the program's input space using randomized search. A less formal way could be that fuzzing is a way to subject software at the mercy of a horde of four-year olds who relentlessly torture the software until it fails. A practical description is that a fuzzer takes in sample files – or a description of the format of the data – and outputs data that is sort of correct but a bit wonky. These files can then be fed to the software we want to test to see if it can handle them without crashing or behaving oddly. This testing method is sometimes called fault injection. Since programs are seldom coded with the "what-if" mindset, this simple (but devious) method of software testing quickly begins to unravel those "what-ifs".

[1] http://pages.cs.wisc.edu/~bart/fuzz/Foreword1.html

There are many different ways to fuzz

Simplest form of fuzzing is to just create completely random data (white noise, basically) and use it as the input of the target program. Conceptually this maps to completely random selection of different inputs from the input space. Even though simple, even this method can (and given time, will) uncover defects. Depending on the software we want to test, however, there might be need to concentrate on certain parts of the input space. For example, if we were to test a HTTP server and we presume a structure where the first bytes of the request are parsed to determine which HTTP method does it map into if no match is found, the connection is closed. In this situation the first bytes that are randomised need to form string "GET", "POST" or some other HTTP method in order to "get" into the more complex parts of the HTTP request parser. It is obvious that this seldom happens. In these situations, where the input is structured it is sensible to make data that has bigger probability to resemble valid HTTP requests in order to achieve better code coverage by getting bigger percentage of the test cases to get past the initial barrier.



To get there, several different approaches have been developed. One approach is to take valid test cases (valid HTTP requests in this case) and to cause simple mutations to them. One simple approach is to walk through the file bit by bit and flip each bit with a certain probability. More complex techniques include mixing data from several different valid requests, copying a sequence of bytes few times, deleting rows, swapping integer values with random ones and so forth. Another approach is to create a grammar of the data either by hand or automatically and then either create data based on the grammar or first mutate the grammar and then create data based on the mutated grammar. Naturally it is also possible to mix these two approaches, or to write a specialised domain-specific fuzzer with fancier features, such as calculating checksums for the random or semi-random data and emitting them to the correct places.

What's there to like about fuzzing?

For one, fuzzing is a really, really cost-effective way of hunting bugs. One good indication of this is that many of the professional bug hunters use fuzzing as their main tool; they would not be using fuzzers if there was a better and more profitable way to catch bugs. Many companies have also realised this, for example Google (fuzzing cluster consisting of thousands of machines) [2] and Microsoft (fuzzing research and requirement of fuzzing in their Security Development Lifecycle) [3] [4]

Fuzzing is also really effortless to set up compared to many different testing methods: download a fuzzer, point it to some sample files and create to script to feed the generated test cases to the application you want to test and devise a way to catch crashes (exit codes and system logs are some notable candidates for this). Set the script running and go about your day: let the computer do the heavy lifting.

What kind of software should be fuzzed?

Naturally it is not a bad idea to fuzz any software, but if the software fits the bill for one or more of the

[2] http://blog.chromium.org/2012/04/fuzzing-for-security.html

[3] http://msdn.microsoft.com/en-us/library/cc162782.aspx

[4] http://www.microsoft.com/security/sdl/default.aspx

following criterion, the fuzzing is probably a good idea:

- software with components written on languages that use manual memory management
- software with parsers
- software which is widely used
- software which should be robust
- software – or systems – which consists of several separate components interacting with each other

It should be noted that many of the modern software components fulfill part of or the whole criteria, especially if the software is part of cloud stack or integrates part of its functionality with components running in the cloud.

Phases of fuzzing

Sutton et al. [5] identify following phases of fuzzing:

1. Identify target
2. Identify inputs
3. Generate fuzzed data
4. Execute fuzzed data
5. Monitor for exceptions
6. Determine exploitability

In phase one, the testing target is examined to find out which kind of an application it is and to determine which kind of fuzzing tool or technique would be best fit for it. For some applications there might exist domain-specific fuzzers which might be worthwhile to try out. For example, if the application deals with network protocols, there are existing fuzzing tools which are specialized in testing protocol implementations. In other cases, a general-purpose fuzzer is the best bet. Note that it is always worthwhile to also try the general-purpose fuzzer, even if domain-specific tools exist.

After the target has been identified, it is time to continue to the next phase to identify the input vectors of the target. In practice this means that we try to find out all the different channels and formats the programs reads and uses. Although this might sound like a cakewalk, it is easy to omit some input vectors which are hidden in plain sight: things such as the name of the executable, favicons, environment variables, file formats that are "unofficially" supported, and so forth. Another thing to note is the different level of the "structuredness" of the data. For example, a jar-packet of Java code might contain magic binary values as a value in XML-markup in turn embedded to java code, which lives inside the zip-encoded archive file. It might be profitable to decide which of these levels we want and do not want to fuzz.

Once the input vectors have been identified it is time to begin fuzzing by generating test cases using the fuzzer of choice. Usually the steps from 3 to 5 are scripted so that we create some amount of test cases, feed them to the application and monitor if the application seems to hold up. If faulting test cases are found, they are stashed along with the instrumentation data about the type of the crash. Otherwise the created test cases are deleted and process begins again from step 3.

Once the system is up, the tester's job is to occasionally check the cases found to determine if they include critical bugs. Otherwise the tester is free to commit to other productive tasks, as the computer is doing the heavy lifting.

[5] Sutton, M., Greene, A., & Amini, P. (2007). Fuzzing: brute force vulnerabilty discovery. Addison-Wesley Professional.

Instrumentation and testing oracle

One critical piece of fuzz testing is the instrumentation of the software we are testing. The application will be bombarded with large amounts of test cases and thus it is necessary to automate also the identification of faulting behaviour of the software since it is not feasible to require a person to sit by the machine and try to catch errors as they emerge. Fuzzing is very much about automating as much as possible of the testing, since the manual phases often become performance bottlenecks – in addition of often being really boring.

Creating the instrumentation and automating the testing oracle – the part of the testing framework that tells us if the program functioned correctly or not – is usually quite simple, especially when testing self-written software: wire up the error output and the exit value of the software to the testing script and based on that determine if the software functioned correctly or incorrectly. Naturally there are a wealth of other potential sources available if needed, such as system logs, memory error detectors, triaging tools and kernel message buffers – or just periodically checking with a valid test case that the software is still responding correctly.

It is good to note that the testing oracle needs to function well enough (=not miss too many faults), since the effectiveness of fuzzing is directly linked to the the effectiveness of the testing oracle: if the instrumentation fails to catch half of the faults, then half of the effort is wasted. Thus it is important to inspect the instrumentation, if the results of the fuzzing seem to be lacking in quality or quantity. Often the testing oracle consists of tools such as ?AddressSanitizer, which catch most test cases which might potentially include security bugs, and test case filtering, which tries to filter out false positives (such as out of memory bugs) and duplicates of bugs already found. If the filtering is not tuned correctly, it can miss bugs or fill the logs with duplicates and false positives.

Fuzzing, improving security and agile software development

Fuzzing and agile software development are a young couple. As a concept, however, they do seem to fit together: both value quick results, both thrive on continuous adaptation as the conditions change and both strive to automate and cut away unnecessary externalities to improve the testing or development process as far as possible. However, fuzzing is seldom mentioned when talking about testing in agile software development. Why, one might ask? Most probable reason is that fuzzing is relatively unknown method of software testing and is mostly known by security testers, often domain experts who are – sadly – rarely working as a full-time part of software development or testing teams who plan and execute the day-to-day testing and quality assurance the software receives. This does not aid in spreading the word about fuzzing. Additionally fuzz testing might be misunderstood as a testing technique only applicable as the means of finding security-related issues, and because of that is ignored by the general software developer or tester, which is unfortunate, since fuzzing also finds general bugs. Nowadays more and more software is or depends on web services and cloud infrastructure, so even the common developer should not ignore testing method which is also effective in uncovering potential security issues.

One interesting issue people might have with fuzzing – or other effective forms of negative testing – is that it might be "too good" in finding bugs: positive test bias is very much an existing phenomenon in software testing. This also applies in agile software development: it is easier to come up with positive user stories ("as an administrator, I can reset the user password") than negative ones ("as a user, I cannot crash the system if I type random garbage to the login field"), or positive unit tests than (good) negative ones. One might even want to eradicate most (if not all!) the negative tests and user stories if they form part of the specification of the software, since they seem like useless and obvious cruft. However, if the aim is to create robust and well-behaving software in the long run negative testing needs to be addressed as well – and it is best to start early to minimise the accumulation of bugs and faulty behaviour. A word of advice though: it is best to make the features somewhat ready before beginning to fuzz them, since it discourages the development team less and makes the filtering of false positives easier.

Since in agile software development testing is preferably automated, if feasible, there is also the question of integrating the fuzz testing to other testing infrastructure: fuzzing should be automatically executed at a suitable phase of the development and possible issues should be automatically committed to the appropriate location, such as the project's bug tracker. Traditionally fuzzing is isolated from other test effort and has own testing pipeline, but this is probably because of the usual background of persons engaging in fuzzing – security testing. However, if we wish to integrate fuzzing, we need to come up with solutions how to do it. At the moment, there are no a drop-in modules that could be dropped into the CI framework with minimal configuration and suddenly everything would be fuzzed, but this does not seem to be the path to take due to the characteristic of fuzzing. In order for fuzzing to be useful, it should be targeted to the parts of the program which handle unverified data, such as user input or data pulled from the network (if all the functions of the program are fuzzed, we end up in situation where every function needs to be written defensively, which is usually not necessary). Another reason is that the developer usually knows best what kinds of inputs each part of the code might and should deal with and has or can easily acquire the appropriate test cases for it (which we can use as basis for fuzzing). Finally, the developer usually knows best how the testable parts of the software work and should work, and how to build a working test oracle for it, as automating the creation of testing oracles is not an easy task. Thus it seems to be the best solution to offer good general-purpose fuzzing tools and frameworks for the developers to use to build their own fuzzing and the required handling of found test cases and to provide the missing glue that leads to the integration of fuzzing to the rest of the project's testing effort. Practically: provide the testers with fuzzing tools that take in valid cases and output fuzzed ones; the tester can then relay the fuzz cases to the software and build a testing oracle that will catch the faults and relay the findings to the appropriate destination, such as bug tracker.

Experiences and discussion

Fuzzing has been successfully applied while testing many different flavours of programs, uncovering a hefty amount of bugs in production software. In the beginning it was used to test UNIX utility programs, but since then it has been applied in testing domains such as protocol implementations, large desktop applications (such as Microsoft Word), operating system APIs and web browsers. In general you can presume that if the software component or application has not been fuzzed before you can be almost sure that you will uncover bugs from it. Because of this fuzzers are very popular among people who hunt bugs professionally or semi-professionally (e.g. browser bug hunters or people who try to find the necessary bugs in order to jailbreak the latest gadget), since fuzzers provide good bang for their buck. From our viewpoint it does not seem to be that the mileage of fuzzing would run out in the near future, in general. Fuzzing does have the property that you need to be able to construct the testing oracle for the software tested, so in principle fuzzing might become more difficult if the software and platform vendors decide to obscure the debugging interfaces and other components used in building a testing oracle...but that makes the testing more difficult in general and might in the long run even alienate the users and developers depending on the software or platform.

The Radamsa tool developed by us has already been used to find over a hundred previously unknown vulnerabilities in browsers. All these vulnerabilities have been reported to the manufacturers at once so that they could be fixed as quickly as possible. Vulnerabilities have been found in anti-virus programs and widely used image and audio formats as well. Radamsa is a completely automated data security testing tool, which is the architect of the structure and the creator of testing events. In it, the best properties of previously developed automated data security testing tools have been collated. The Radamsa software has been developed in the course of a four-year Cloud Software programme. Business partners in the project have included Ericsson, Nokia, F-Secure, Google, the Mozilla Foundation and WebKit.org. Radamsa is developed as an open source project. Many of the tested programs, such as Firefox and Chrome, are wholly or partly source projects and use a lot of shared libraries. Due to this, vulnerabilities that have been fixed usually help to indirectly improve data security in other projects, such as almost all Apple devices, Android phones and smart TVs.

References

Sutton, M., Greene, A., & Amini, P. (2007) Fuzzing: brute force vulnerabilty discovery. Addison-Wesley Professional

Takanen A., DeMott J. D., Miller C. (2008) Fuzzing for Software Security Testing and Quality Assurance, Artech House Print on Demand, ISBN 978-1596932142, 287p.

Pietikäinen P., Helin A., Puuperä R. et al. (2011) ``Security Testing of Web Browsers'' Comm. of Cloud Software, vol. 1, no. 1, Dec. 23, ISSN 2242-5403. Available: http://www.cloudsw.org/current-issue/201112226146

Helin A.(2013) Radamsa fuzzer. Available: https://code.google.com/p/ouspg/wiki/Radamsa

Dynamic Trust Management

Sini Ruohomaa, University of Helsinki
Lea Kutvonen, University of Helsinki

Executive summary

The system security life cycle does not end at software development, and is not maintained by technological considerations only. This chapter takes a look at enhanced security during operational time when a composition of software services run together forming a system that spans over organizational boundaries, in a shared cloud or between clouds. The take-home message of trust management is that the world is not divided into black and white of authorized and unauthorized users any more, but you need to be able to deal with uncertainty and risk while collaborating with services provided by another organization with different goals, incentives, norms, and practices. Making trust decisions in this kind of environment under changing business situations requires some tool support: contracting, monitoring, reputation systems and risk-aware trust decision systems.

Target audience: SME management (incl. CTO, CIO), service engineers

What you learn from this chapter:

- What is reputation-based trust management about,
- What does it need around itself to be successful, and
- What are the key points where things can go wrong.

Introduction

System security is about setting up multiple layers of defense: when one security mechanism fails, all is not lost. For example an Internet service provider (ISP) needs to identify its customers, to whom it provides access to the Internet. The means to sort paying customers from outsiders is a security mechanism: identity management and authentication give the service provider a way to figure out who it is dealing with. To allow users to change their service settings securely, the ISP may provide an encrypted connection between the customer and its server, in order to stop outsiders from interfering with the communication.

Individual security mechanisms may turn out to be insufficient when the users, outsiders and even the service provider's staff do not follow the system designers' expectations. The user's computer may be infected with malware, and the ISP suddenly becomes a service provider for a malicious botnet operated from abroad. Alternatively the user may simply be a great fan of streaming video, and overload the ISP's network. Normal people who just want to get things done may also circumvent a security mechanism that seems to get in the way: not bother to configure appropriate logging, bypass an access control step for example to let a friendly stranger through the door, or save credit card data or passwords unencrypted just until there's time to do it properly. No single mechanism can protect a system, because the very idea of security as stopping bad things from happening means that the different possibilities for failure are infinite.

In collaborations between services, setting up identity management and authorizing a partner for access to the service is not enough by itself. For cloud infrastructure services, Service Level Agreements (SLAs) are commonplace to specify the duty of the infrastructure provider to deliver uninterrupted service under specific terms. Similarly, combining services like flight reservation, a travel agency website, hotel booking and online payment handling require contracts to be specified both on how the service is to be used and how it is to be provided. Legally binding contracts, together with the judicial system, form another security mechanism which focuses on behaviour after the user has been allowed access into the system. Together, these different mechanisms support inter-enterprise collaboration governance.

The inter-enterprise collaboration situation is illustrated in Figure 1 where entering a collaboration as a travel agent requires the travel agency itself to trust in business terms that the bank, hotel and flight service operators are intending and capable of providing the contracted services, and that their organizational and IT governance are solid enough to overcome any security or operational faults without letting them affect their collaboration partners' needs. The end result here is the enhancement of traditional security levels to risk management and trust in business situations.

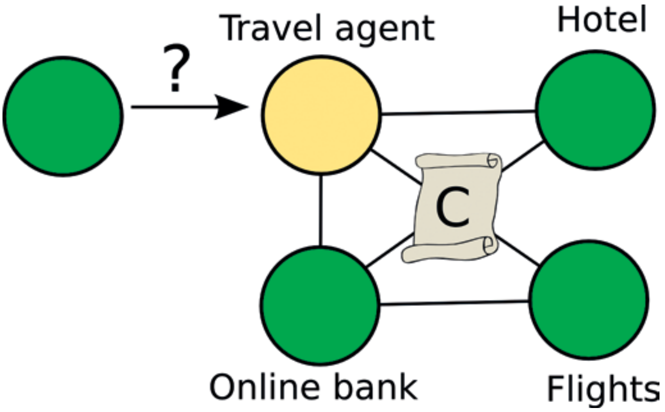


Figure 1: A contract-governed collaboration consists of roles fulfilled by services.

Trust is needed wherever security cannot provide a control against misbehavior. For example a hotel, after having confirmed a room booking, must also deliver the service to the user of the joint travel service when she shows up a month later; otherwise the travel agency will have an angry customer in its hands. The travel agency has a dependency relationship to its hotel partner; this kind of problem cannot be protected against by standard information security, because the collaboration itself inherently causes the risk situations.

In a broad sense, trust management is a security mechanism that aims to support educated decisions about these risks. Like other security mechanisms, it cannot solve system security by itself, but provides another layer of protection. This layer is directed at what authorized actors do or fail to do when interacting with the protected system under a collaboration contract (whether it is explicit or implicit). The basic idea can be generalized to monitoring any actor behavior in a system, although with caveats. From the point of view of management, trust management falls and to a degree bridges between purely technical security solutions, such as channel encryption, and governance mechanisms, such as contracts.

The following sections present the concepts relevant for trust management, describe how trust management fits into a larger service ecosystem infrastructure, and discuss lessons learned from research on reputation-based trust management.

Concepts

Trust decisions are made by a trustor service about a trustee service. The actors we look at through this are policy-governed and software-supported services, i.e. software agents. We look at services instead of the providers behind them because two different services from the same service provider may have different decision policies, different reputation due to different behavior, or different capabilities. From the point of view of automation, the service provider – which is a legally responsible actor, such as an organization or a company – expresses its values and intentions through policy, which is then interpreted by software governing the services.

Trust is defined as the willingness of a trustor to enter into or continue in a collaboration with a given trustee, considering the risks and incentives involved. An alternative common definition of trust is based on "subjective probability" of the trustee behaving well from the point of view of the trustor. We encode this probability in a risk evaluation, and therefore consider willingness to depend the core of the trust concept.

Trust management is defined as the activity of collecting, maintaining and processing the information that trust decisions are based on. While computational trust management is a security mechanism, security does not generate trust as such. It reduces risks, and thereby leaves a smaller gap between risks and incentives for trust to fill: in other words, security reduces the need for trust. There is an alternative definition of trust management as well, first used by Blaze and Feigenbaum, that refers to certificate-based access control: i.e. whether I am willing to accept that the credentials you have provided mean you should have access to this service. The two should not be confused; for clarity, the brand of trust management discussed here is also referred to as reputation-based trust management.

Trust decisions are based on balancing between the estimated risks and risk tolerance for the situation. In the context of our work, risk is defined as the cost-benefit estimate of the action being decided on. Risk tolerance is a specification for the risks that are clearly acceptable, not acceptable, or somewhere in the gray area, which means that a human user must make the decision. It is influenced by the business importance of the action being considered: as an example, we may not be usually willing to sell for credit, but if we need to be freeing up space in our warehouse for a newer product, we might be more willing accept the risk that that payment is delayed or leads to debt collection. The decision context affects the different factors; for example having an insurance in place for a specific contract may reduce monetary risks for that without affecting other collaborations.

Figure 2 captures the essential trust decision information elements.

Trust decisions are first made when committing to a given collaboration contract, and after that regularly during the collaboration, whenever additional resources are to be committed. The routine operational time decision may change from positive to negative if new experiences or the size of the required commitments indicate there is no longer sufficient trust to continue collaborating with the partner. In this case, the partner may need to be replaced if possible, or if the trustor decides to withdraw from the collaboration altogether, it must weigh in also possible compensation it must pay as a result.

In both decision-making points, trust decisions are determined by private policies local to the service provider organization and the specific service that is acting as the trustor. This kind of decision-making is too close to the core business of an enterprise for it to be directly delegated to global policies set by other actors, and the basis of the decisions is sensitive enough information that it cannot be published lightly in the general case either. Deviations from this base division of concerns should be made with great care.

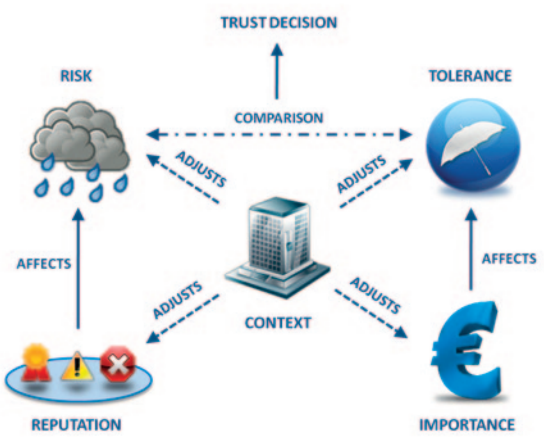


Figure 2: Trust decisions balance risk and risk tolerance.

We evaluate risks from the point of view of effects on different types of assets, which are understood in a broad sense. For example the monetary asset type covers gains and losses that can be expressed in terms of money, while the reputation asset covers the trustor's "good name", positive publicity and standing in reputation systems; this is typically difficult to measure in terms of money, but is threatened in the case of the travel agency by a subcontractor hotel even if the hotel refunded the reservation fees. The control asset type covers immaterial assets of physical and information security, organizational privacy and autonomy; it can be threatened by actual attacks or negligence from a partner, for example, or in the case of autonomy, contracts that have so tight withdrawal terms that they are in practice impossible to withdraw from later, no matter how much resources they tie up.

For covering the competitive benefits of maintaining a "good reputation" and the related risk evaluation, we have also included in the risk evaluation an endangered 'asset' type which does not directly connect to losses or gains. The satisfaction asset type refers instead to whether the collaboration fulfilled the contract and went as expected. This is tracked because aspects such as quality of received service, prompt responses and gaining what was expected from the collaboration may not be reflected as directly measurable monetary, reputation or control effects, but are clearly a strong influencer in the willingness to collaborate in the future.

The risk evaluation then estimates how each asset type might be affected by joining or continuing in the collaboration, for a given commitment. This is based on experiences of past collaborations, which record the estimated effects equivalently.

To limit the complexity of the model from the point of view of the user, we have divided these effects into discrete categories: major negative effect, minor negative effect, no effect, minor positive effect and major positive effect, complemented by a category for 'unknown effect' in the case of experiences, if the effect cannot be determined. The risk evaluation considers the probability of each different outcome; the unknown effect does not have a probability as such, but a high number of experiences recording an unknown effect indicate that the quality of the risk analysis for this particular asset may not be very high and should therefore be taken with a grain of salt.

Experiences of past collaborations are encoded as the trustee's reputation. It is a combination of local experiences gathered through first-hand observations, and external, third-party experiences shared through reputation systems. The value of third-party experiences lies in complementing local experiences when a) a trustor has no or very little previous own experience about the trustee, or b) to serve as a warning when the trustee's behavior has suddenly changed elsewhere, particularly for the worse. From the point of view of the entire community, reputation systems make it more difficult for an actor to reap great benefits from misbehavior, as the first few victims can warn off others for a fraudulent service provider. They act as a shared memory that makes it possible to learn from others' mistakes and make the system as a whole more secure.

While reputation systems can be seen as another security mechanism, they also bring their own problems: when a good computational reputation becomes worth money due to "buying" entry into new collaborations, there will also be an incentive to commit reputation fraud, i.e. to gain reputation without deserving or to "steal" it from others. Reputation systems must therefore have additional layers of security to defend against attacks towards themselves. When experiences are shared between different actors, they must be evaluated locally for credibility before using them in decision-making, because incoming experiences may be both too subjective to use and downright false - two competitors might provide negative feedback on each other or undeservedly praise their affiliates, for example. This critical input processing is directed by its own set of reputation update policies, which together with the trust decision policies form the heart of the trust management system.

Service ecosystem engineering for trust management

In order to talk about trust management as a part of the different layers of security and governance mechanisms it operates in, we will need to take a step back and look at the service ecosystem for a bit.

The individual collaborations between services (e.g. the travel agency, flight company, payment handler and a hotel chain) operate as a part of a service ecosystem. The service ecosystem consists of infrastructure that enables these collaborations and which can have varying degrees of flexibility from the point of view of individual collaborations. The elements of an ecosystem include [1]

- infrastructure services (e.g. means of finding suitable business services for a given need or a reputation system) and information repositories (e.g. available service offers),
- shared norms and standardization (e.g. (de-facto-)standard template contracts such as are already used in e.g. construction, or shared business process model templates),
- the actors/stakeholders (e.g. service providers, service consumers, infrastructure providers, domain experts) and
- the actual business services and the collaborations formed from them.

We believe that the greatest benefits from computational trust management are realized when it is brought together with other infrastructure. This is because automated decision making is highly dependent on its input data and policy configuration; the cost of setting up the necessary support for trust management is likely to be too high in relation to the marginal benefit. For this reason, we have focused our work on studying how it works together with other infrastructure services and how it can benefit from the same input data that they need.

For example contract templates, which are already in use in inter-enterprise collaborations, involve an establishment of some best practices on agreed-upon norms. For example a union of service providers, such as in the Finnish building and renovation industry, can establish a standard contract template where the details of the work, such as pricing and stakeholders, are filled in. This kind of standardization saves time when setting up new contracts. Similar best practice template policy sets could be provided for easy adjustment of service providers to simplify the configuration of trust decisions.

One core service that only trust management relies on is the reputation system for sharing experiences. We have observed a need for organizations to learn about different service providers, but there is little support for it currently: the provider can make claims on its own website about who it has worked with, for example, but this information is set up for the purposes of marketing, and is almost certain to omit any negative experiences. Unions of service providers with a common interest to improve the quality of the market can define certification programs and databases of financial stability (e.g. Rakentamisen Laatu, http://www.rala.fi/ and Tilaajavastuu, http://www.tilaajavastuu.fi/). The databases can also be used as a more neutral location for storing references and even feedback of earlier partnerships, as Rakentamisen Laatu does.

In the short term, publically shared reputation information such as the aforementioned references can be made visible for example through web portals for domain-specific partner searches. In the long run, we expect services to emerge that interpret the available information into more refined recommendations, and potentially support more fine-grained experience sharing.

Figure 3 illustrates the connection of software development lifecycle, the composition of collaborations at the operational time to produce more advanced services, and the connection to the ecosystem-wide reputation information flow as an element of trust decisions. The ecosystem infrastructure keeps the produced software-based service components, collaboration templates (such as business process models), and availability information of the variety of services from different providers online. Each collaboration establishment and continuation is dependent on the organizational or company actors' trust decisions on the specific situation, affected by on one hand their private knowledge about their business drivers and risks, modeled risks visible from the ecosystem knowledge, and the changing flow of reputation information visible for all ecosystem members.

[1] see Kutvonen, Ruokolainen, Ruohomaa, Metso, 2008.

Experiences and discussion

The two parts that make or break a trust management system are its policies and input data. While the trust management system prototypes that have resulted from research are not finalized products that could be tried out in real environments, they help in gathering experiences on the general principles of building these kinds of systems. In addition to the results reported here, research is ongoing on how publicly available information could be automatically compiled together into a mashup for a partner search portal [2].

Policy configuration

A policy-based automated system can only make decisions and process its input based on the explicitly configured policies. The key requirements for policies can be divided into three categories: 1) sufficient expressiveness for the policy language, 2) ability to configure and modify the policies, and 3) acceptable performance costs for processing them.

The internal data model of the system must be able to support expressing sensible policies. In our research work, we have taken a step forward from the state of the art by separating risk from reputation and business importance in order to make their relationships explicit. We have also advanced from calculating pure probabilities for "good" or "bad" behavior to focus on the effects that different good, bad or gray-area actions have some kind of effect on **assets** the system is set up to protect.

Most software engineers have been told how it is important to separate policy from implementation. It must be possible to change the entire trust management system from one configuration to another while it is operational, because a security system handling actor behavior must be particularly able to adapt to different situations. We also believe that trust decisions are not a straightforward process with clear cases, so we have made sure to add a means to set up "special cases" with different triggers, such as the identity of the actor, a specific contract or the point of time of the decision. These special cases are referred to as context filters: they tweak the decision inputs or output according to the surrounding context.

As a final precaution against the inherent unpredictability of special cases in behavior control, we conclude that the decision system can only automate routine decisions, while unclear situations must be resolved by a human user. For this purpose, we have designed the system to be aware of a set of signals that indicate a decision situation is not routine and must be forwarded to a human user, and automate only those decisions that have sufficient and clear input data to support them.

Our evaluation so far has shown that an extended information model allows us to address certain kinds of attacks and misbehavior that could not previously be identified at policy level at all. On the other hand, the cost of having more sensitive adjustment knobs to turn is that the system may become more complicated to configure, to the point of being critically less usable than a more coarse system that demands hardly any user effort to use at all. The valuation of which adjustment knobs are worth the effort to take into use also depends on the user of the system, while on the other hand it is impossible to prove that a current set of adjustment controls are sufficient for all reasonable needs.

We find that the configurability and ease of configuring the trust management system is a key usability factor and a requirement for balancing between the conceptual complexity of the model and the quality of the decisions made. This can be further divided into the need for configuring effort, and the usability of the configuration interface. We have focused on identifying the minimal configuration that the system can operate on, to ensure that the user has the choice of taking the additional expressiveness of the policies into use. In terms of user interfaces, we have so far collected some early feedback on how easily users can understand the concepts behind the prototyped system in a decision-making situation [3].

In a perfect world, the system would configure itself to do the "right thing" based on what the contracts and internal business policies say, but even the age-old legal tradition cannot produce contracts that

[2] BPaaS-HUB, http://193.166.88.67/bpaas/

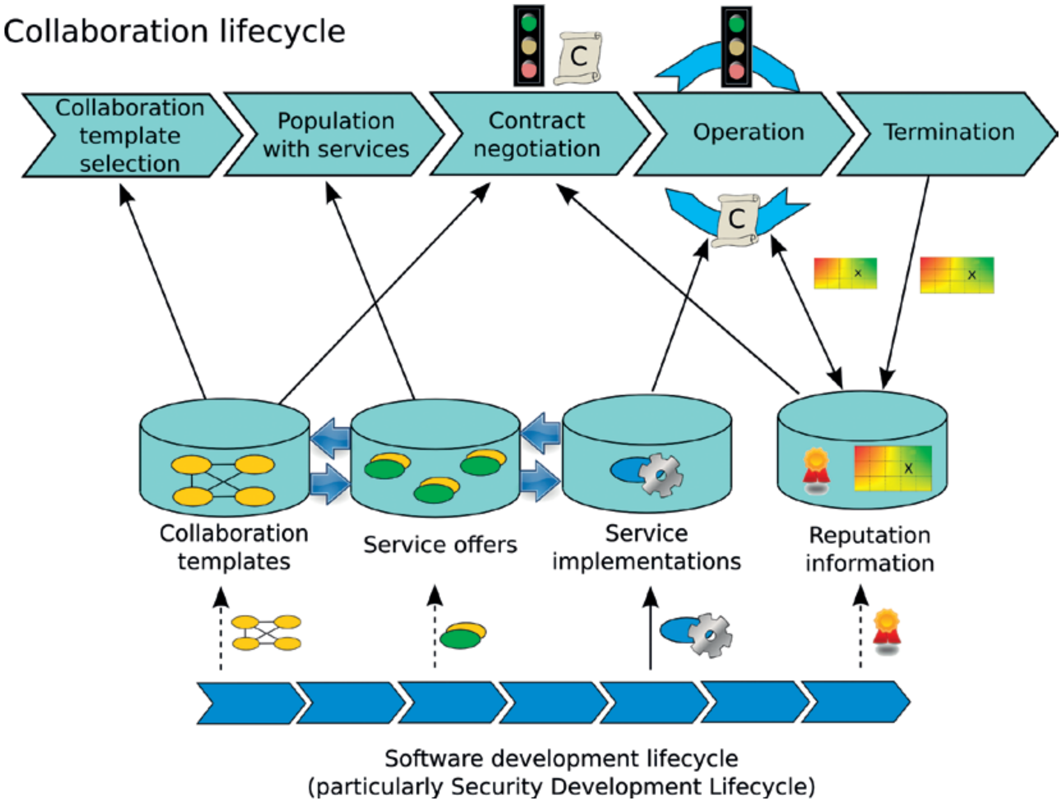


Figure 3: The collaboration lifecycle builds on artifacts produced during the software development lifecycle, most notably the service implementations.Trust decisions are made before committing to a contract and during operational time, and experiences from operation and termination are recorded as reputation information.

need no interpretation and could be automatically judged. Policies reflect the preferences of the organization, and not even all contract violations may be equally worth reacting to: maybe one long-standing partner has a tendency to be a bit unpredictable in its response times but deliver high-quality service otherwise, while in another situation a failure in timeliness would mean a swift end to the collaboration, for example.

Providing every service provider with an equal default policy is also an attractive idea, and the minimal configuration approach is likely to be close to this. One should keep in mind the problems of monoculture too, however: differences in decision-making behavior make the entire ecosystem less attractive a target for attackers, for the same reason that it is beneficial that not everyone on the Internet has the exact same operating system version when a vulnerability is discovered and exploited in it. This also translates to a need for trust management systems to be able to interoperate also when not everyone uses the same system, let alone with the same configuration.

Alternatively, configuration could be eased by learning algorithms: we could hope to provide a set of sample situations for the system and, like a learning spam filter for email, it would then be able to categorize later situations as good, bad or something in between.

The final requirement for policies is that their evaluation should not take too much time and computational resources. As trust decisions must be done in real time, slow decision-making reflects on the user experience. We find that in the system prototype, input information can and in general should be preprocessed so that the actual decisions can be done efficiently.

We have evaluated the performance costs at the level of computational complexity. The two aspects

[3] See Kaur, Ruohomaa, Kutvonen, 2012.

of interest are the number of context filters and the amount of processing needed at the monitoring points. First, the context filters are the technical representations of situation types involving business interests and risks. The complexity increases linearly as the number of filters increase, but it is not likely that business managers would want to deal with an excessively rich world of situations. Second, the processing overhead at monitoring points needs to be kept low, although the monitoring points create the triggers for producing experience reports. We have concluded that the experience gathering for trust management does not need to happen in real time, so the overhead from the point of service use is limited to identifying points where to make the appropriate log entries for the purposes of offline experience processing. Other researchers have studied the operational time costs introduced by monitoring from the point of view of Service Level Agreements and found the results promising [4].

Input data

A state of the art review we have conducted indicates that computational reputation systems for enterprises do not really exist yet; what has traditionally been used is project references, such as in construction business, or past partners and implemented cases in software business [5]. This kind of referrals serve marketing purposes, but also act as an anchor for spreading word-of-mouth kind of reputation: the prospective customer can in theory go see the result or at least ask the partner about their experiences, if they are willing to share them. The feedback collection system of Rakentamisen Laatu is a domain-specific service collecting and aggregating manually entered feedback about construction projects; the feedback information is visible to registered organizational users only.

We have explored different possible business models for establishing a reputation system specifically for computational trust decisions. We have found that a bulk, mostly automated reference collection service could be offered at a low cost to customers such as small to medium enterprises as a web service. On the other hand, more complex analysis personalized to the needs of the specific enterprise demands more resources, and while it provides more control to the user, it is also more likely to only be feasible to offer to large organizations due to the inevitably higher costs [6]. In addition, mash-up approaches for reputation information may not be able to extract the more specific information from its silos: infrastructure service providers collecting reputation information build their business around it, and may not be willing to publish it to external service providers [7].

Credibility is an important factor in shared external information: where good reputation translates to more business opportunities, it is worth enough money to attract both downright reputation fraud and also less deliberately harmful attempts at boosting the service provider's own reputation, such as attempts to buy or otherwise solicit positive feedback outside the expected method of delivering good service. Credibility concerns apply both to the actor sharing their own feedback and to the system delivering aggregated feedback of its users: for example a restaurant rating service has been sued for selling its users means to conceal their negative feedback [8]. From this point of view, organizations that are more closely tied to and represent their own user base, such as associations of service providers within a domain, may be more natural infrastructure service providers for setting up domain-specific enterprise reputation systems than entirely independent companies.

[4] See e.g. Raimondi, Skene, Emmerich, 2008.

[5] See also Ruohomaa, Kutvonen, 2013.

[6] For an overview of the services, see Ruohomaa, Kutvonen, 2013. The business models are analyzed in Ruohomaa, Luoma, Kutvonen, 2013.

[7] For example the online auction site eBay's user agreement explicitly forbids users from exporting their reputation to other sites.

[8] See Ruohomaa, Kutvonen, 2013.

References

Sini Ruohomaa. The effect of reputation on trust decisions in inter-enterprise collaborations. PhD thesis, University of Helsinki, Department of Computer Science, May 2012.

Puneet Kaur, Sini Ruohomaa, and Lea Kutvonen. Enabling user involvement in trust decision making for inter-enterprise collaborations. International Journal On Advances In Intelligent Systems, 5(3&4):533-552, December 2012.

Sini Ruohomaa, Lea Kutvonen. Rolling out trust management to cloud-based service ecosystems. Technical report, University of Helsinki, 2013.

Sini Ruohomaa, Eetu Luoma, Lea Kutvonen: Trust broker service, 2013. Submitted manuscript.

Lea Kutvonen, Toni Ruokolainen, Sini Ruohomaa, Janne Metso: ''Service-oriented middleware for managing inter-enterprise collaborations.'' Global Implications of Modern Enterprise Information Systems: Technologies and Applications (2008): 209-241.

Raimondi, F., Skene, J. ja Emmerich, W.: Efficient online monitoring of web-service SLAs. Teoksessa Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, sivut 170–180. ACM, 2008.

Appendix: Generic Security User Stories

Tuuli Siiskonen, F-Secure
Camillo Särs, F-Secure
Antti Vähä-Sipilä, F-Secure

This appendix lists the Generic Security User Story templates. Please refer to the main article on the background and usage.

Account lifecycle

As	**a person accountable for privacy (data protection)**
I want	that user accounts have a well-defined lifecycle with all the necessary account states
so that	we can manage users' personal data properly.
Description	User accounts need to have a well-defined lifecycle. For example, accounts need to be created; old accounts' data may need to be purged (personal data handling requirements); accounts may need to be flagged or locked because of technical issues or misuse; new accounts may need to go through specific validation steps to be fully enabled (for example, require valid email address to be able to use all features).
Acceptance criteria	>> A state machine description of account states exists, and is documented for maintenance. >> Test cases exist that take an account through all the possible states of an account according to the state machine description. >> Negative test cases exist that try out activities in various states of an account that should not be possible in those states, and verify that the activities fail.
Refinement questions	>> How and by whom are user accounts created? >> How and when are user accounts destroyed? >> What sort of "special states" can user accounts be in? >> Have you thought about failing user interactions (e.g., registration failures) and in which state they will leave the user account?

Access control policy

As	**an information security manager**
I want	that it is clearly defined which user accounts are authorised for which activities
so that	the effectiveness and correctness of access controls can be checked.
Description	It should be clear what different user accounts can do. For example, if there are different user roles, administrator role, power user role, normal user role, and so on, should have clear definitions as to what they will be able to do. The existence of a policy is important for the information security manager because it allows checking whether the system actually conforms to the policy and effectively protects the information.
Acceptance criteria	>> Access control policy (clear definition) exists in design and is provided for maintenance.
Refinement questions	>> What sorts of users or user roles we have, or should have? >> Can users be in more than one role? At all, or at the same time? >> Can a policy conflict between two or more roles for a single user? >> Is there a permission model in the system that can be mapped to these user types or roles, and if so, what is this mapping?

Access control implementation

As	**a user**
I want	that access controls effectively prevent all unauthorised actions or operations
so that	I am protected.
Description	Access controls (whether on user level, or component level) must prevent all unauthorised actions, and a key to this is to have a clear and understandable access control model. Clarity of code and maintainability are critical quality aspects. It is also important to use the "least privilege" principle, meaning that users and components are only given the access they really require in the policy, and not anything "just in case" or "because it did not work otherwise".
Acceptance criteria	>> Negative tests that try to do accesses that are prohibited by access control policy (i.e., should not succeed). >> Exploratory (manual) tests by a professional that try to bypass access controls (in ways which have been left undefined by policy, for example). >> If there are "test accounts" or access control has a "test mode", there must be a test case that verifies that these test accounts and modes have been disabled in released code. >> Tests exist where several users in different roles use the system concurrently, and check for concurrency problems.
Refinement questions	>> Are all access control decisions centralised in one (or few) components instead of being spread all over? >> Do all of your components require appropriate permissions from other components that call them, and do your components only have those permissions that they really require? >> Can you easily explain how your access control works?

Architectural threat analysis

As	**a product owner**
I want	that each functional feature is subjected to an architectural security and privacy threat analysis
so that	all security work that needs to be done is identified and visible on the backlog.
Description	A data flow based threat analysis (a.k.a. threat modelling) should be done on architectural level. This should identify data flows, data stores, and security boundaries in your system. Each data flow and store should be subjected to a threat analysis exercise (for example, utilising Microsoft's STRIDE method). This analysis should produce security features that need to be added, and identify new testing needs. In essence, threat analysis produces security-related new backlog items. This is also needed to produce evidence to customers that security has been thought about.
Acceptance criteria	>> Evidence of architectural threat analysis having been done. This evidence can include new backlog items and test cases as well as analysis notes.
	>> Evidence of threat analysis must be tangible enough so it can be presented to a customer or an independent security assessor.
Refinement questions	>> Do we have a good enough understanding of the architecture so that we could draw a Data Flow Diagram without leaving too many vague areas? >> Which threat analysis method is best suited for our purpose? Do we have a recommended method? (Example: STRIDE with a Data Flow Diagram.) >> Do we have a good grasp of current software security weakness landscape, or should we seek help? >> Should we make architectural risk analysis a standard practice for major new functionality, for example, through our Definition of Done?

Interface robustness

As	**an information security manager**
I want	that every interface is robust against malformed or invalid inputs and malicious data injection
so that	attackers cannot bypass our security.
Description	Interfaces need to be robust against inputs that do not conform to the specification. This kind of inputs include malformed or broken data, inputs that are out-of-sequence or wrong in a given state of system, interactions that are incomplete, or contain various injections (for example, database or scripting injections). The expected result for any of these kinds of incorrect inputs is an error that is handled gracefully, or silent rejection of input. The target is not to add complexity to cover all eventualities but instead to avoid undefined or unmanaged error situations such as crashes and core dumps, hangs, performance degradation, denial of service, or data corruption.
Acceptance criteria	>> Test cases have been implemented that test interfaces with malformed or invalid inputs, and malicious data injection.

Refinement questions	>> Which interfaces handle data received from outside the system (directly or indirectly)? (This information can be obtained from an architectural threat analysis.) >> Which interfaces do not need to be robustness or injection tested? Why? >> Where does each data flow terminate (on each protocol level; in a typical web application, the layers could be, e.g., TCP, TLS, HTTP, and application data)? >> What sort of injections should we test against for each of these data flows (malformed data, a.k.a. fuzz testing, or injections like Cross-Site Scripting, SQL injection, command injection)? >> How should be ensure that robustness tests are also built for any future interfaces we might create?

Third party security assessment

As	**an auditor**
I want	that the security of the product has been assessed by an independent third party
so that	I can rely on the results of the assessment.
Description	Third party (external) security assessments are needed to validate the security activities that we have done internally. These are required both as a customer requirement of independent verification, and because we expect the third party to be up-to-date on any cutting edge security issues, and because for audit purposes, independence from actual developers is required. The use of third party assessments should aim to be validation only, which means that security testing or threat analysis should not be outsourced to them. Third parties should be contracted only after security threat analysis and security testing has been conducted internally, and before the product is launched or given to a customer.
Acceptance criteria	>> A written third party assessment report. >> Bug reports or product backlog items that target the third party findings have been created.
Refinement questions	>> What actually creates a need for a third party assessment? (E.g., contractual requirement, business risk acceptance decision?) Given this business requirement, how large our investment (budget) will be? >> When are we in a good enough shape to get an assessment (enough features but still enough time to fix findings, and all known open security issues fixed)? >> Have we utilised all internally available competencies for security testing before engaging a third party? (Third parties should not find problems we ought to have found ourselves.) >> Is the third party competent in our specific technology area?

Availability

As	**a user**
I want	the application to be available when needed
so that	I get the expected value out of it.
Description	Availability is about the application being available for use when required. The application development needs to consider both random faults and intentional attacks. Reasonable effort needs to be taken to ensure availability in both cases. Fault sources include, for example, hardware failures, power and communication outages, and bugs. Intentional attacks typically target capacity (network, memory, storage, CPU time) or logical flaws (deadlocks, livelocks). Intentional attacks actively try to exploit worst-case scenarios.
Acceptance criteria	>> Tests exist that introduce synthetic failures that simulate availability problems.
Refinement questions	>> What are our availability requirements? >> What types of failures, attacks or attackers could cause availability problems? >> Does our architecture have points that are especially susceptible to denial of service attacks?

Business Continuity

As	**a service owner**
I want	that recovery from outages is quick and cost-effective
so that	disasters do not ruin the business.
Description	Engineering for business continuity addresses how quickly and cost-effectively we can recover from a serious outage. Engineering problems revolve around issues such as avoiding single points of failure (geographically, network-wise, logically), ensuring that your application can be deployed automatically, and managing issues related to data transfer and storage capacity.
Acceptance criteria	>> Recovery plan for the application exists, and has been tested. >> Application deployment has been automated.
Refinement questions	>> What is our maximum tolerable outage time? (A time after which it really doesn't matter anymore.) >> How fast are we aiming to recover? (Meaning faster than the maximum tolerable time.) >> What needs to be restored, recreated or reconfigured when re-deploying the application (data and dependencies)? >> When data is restored, how old can it be? >> How can we ensure the integrity and consistency of restored data? >> Is there a need for geographically distributed redundancy and high availability?

External security requirements

As	**sales**
I want	to be able to show that external security requirements have been addressed
so that	we can sell our product.
Description	Depending on the application, its intended use, the type of industry, and the customer, there may be several sources for external security and privacy requirements. These can be, for example, standards, company policies, customer requirements, or regulatory requirements. It is one thing to know what external requirements are needed, but for compliance reasons, it is also necessary to be able to show (i.e., build an argument) that they have in fact been addressed.
Acceptance criteria	>> Tests exist for all external security requirements that can be functionally tested. >> There is an agreed way in which we document the work we do for external security requirements that cannot be tested.
Refinement questions	>> Do we understand our external security requirements for confidentiality, integrity and availability? >> Which of these external requirements can be tested? >> Which of these external requirements cannot be tested, but could be documented? >> How will we document the work for external security requirements?

Accessible privacy policy

As	**a user**
I want	that our products communicate the up-to-date company privacy policy in an accessible way
so that	I can understand how my personal data is used.
Description	The privacy policy describes the private (personal) information that is collected from the user, and how it is used and protected, and the contact point for more information. It also fulfils the other legal requirements for informing the user. Companies often have a standard privacy policy that may be used. The accessibility of privacy policy means that the language of the privacy policy should be simple enough to be understood by the user, and that it is communicated before the user commits to the use of the service, and at any time afterwards at a request of the user.
Acceptance criteria	>> The privacy policy is accessible from the product in a suitable manner. >> The privacy policy communicated by the product can be updated.
Refinement questions	>> What is the target audience of our product and how should we communicate the privacy policy to this target audience? (Age, accessibility, device interface limitations, ability, language.) >> Does the standard privacy policy meet your needs?

Notification of changes to T&C and privacy policy

As	**a company legal counsel**
I want	that when the terms and conditions or the privacy policy change, the end user is notified
so that	it is clear under which terms and conditions we provide the service.
Description	When there are (material) changes to terms & conditions or privacy policy of our products, the users need to be notified. How prominent a notification needs to be defined based on the changes, but it must be prominent enough so that we can argue that the users did have a reasonable notice of any changes and would have had the opportunity to react.
Acceptance criteria	>> Functionality exists that when the terms and conditions or the privacy policy change, the end user is notified.
Refinement questions	>> Do we need to address this in the product user interface, or do we have other ways to contact the users (e.g., e-mail)? >> Do we need to support potential mandatory opt-in regulations in the future? (Meaning that in order to continue the use of the product, the users would need to positively acknowledge acceptance to new terms.)

Audit log of security activities

As	**a person accountable for security**
I want	an audit log of all security activities
so that	we can detect possible system compromises or abuse and conduct investigations.
Description	Administrator and support persons (such as customer support) are only expected to do administrative tasks when there is a genuine reason to do so and the act is allowed by a policy. For example, we must be able to detect administrator changes to system configurations. We also need to be able to log security decisions made by the system, such as access control decisions, and major system state changes like start-up and shutdown events. At the same time, this audit log must be tamperproof (integrity protected) against those persons whose activities are logged. For systems handling sensitive data, this type of logging is sometimes a legal or contractual requirement.
Acceptance criteria	>> Test that all security activities (access control decisions, etc.) create audit log entries. >> Test that all administrator activities in the system create audit log entries. >> Test that an administrator in the system cannot modify or delete audit log or its entries.
Refinement questions	>> What types of administrators exist in your system? (Power users, system administrators, etc.) >> Are there any specific customer requirements for the audit log (for example, due to industry sector or country where the customer is operating)? >> What events need to go to the audit log (versus a normal log)? The difference is that an audit log is tamperproof against the persons whose activities are logged. >> What external connections need to be logged in an audit log?

Audit log of personal data access

As	**a representative of a personal data controller**
I want	an audit log of all access and changes to personal data
so that	we have evidence of activities in order to fulfil legal obligations.
Description	"Personal data" (especially in the U.S. known as PII, Personally Identifiable Information) is any data about a person that can be tied to a person. Whether data is personal sometimes depends on the context. The company is legally required to guard access to and integrity of personal data, and in order to enforce the privacy policy and have evidence, all accesses and changes to such data need to be logged in a way that can be audited after the fact. A typical case would be that a user ends up on evening news and admins would out of curiosity view this person's data. This audit log must be tamperproof (integrity protected) against those persons whose activities are logged.
Acceptance criteria	>> Test that all personal data accesses in the system create audit log entries. >> Test that the users / administrators whose activities are logged cannot forge or delete audit log entries with their permissions in the system.
Refinement questions	>> What types of personal data do we process? (Involve the company legal and information security functions if unclear whether a specific data is personal data in your context.) >> What user or administrator activities trigger the processing of personal data? >> Which interfaces does personal data pass through? (Sending such data, even if encrypted, is data access too.)

Log event separation

As	**a service manager**
I want	log events to be separated by severity and type
so that	log analysis tools can be used effectively.
Description	In many cases, amount of data in logs is very large. Log analysis can become difficult because of sheer size, or because log analysis software is licensed per volume of log events; or that data can be filtered for optimising log storage and use. Because of this, log events would need to be categorised already when they are created. Typically the axis would be severity (how important a log item is; e.g., debug, notice, warning, error, and fatal error) and type/source (e.g., which process/part of the system created it for which purpose, etc.)
Acceptance criteria	>> Test that all log items that have been created have a severity and type/source tag.
Refinement questions	>> What severity levels do we have for log events? >> What is our list of log event type/sources? (List should be uniform across the whole system.)

Protection of log events

As	**a service manager**
I want	log events to truthfully reflect what is / was happening in the system, during system use and after a system failure
so that	events can be investigated.
Description	Log items may contain a lot of information either directly or through correlation that may be sensitive in several ways, so the data needs to be protected against unauthorised access. Data needs to be a reliable source of information, so they must not be lost or corrupted. Data also needs to be available even if the logged system itself would completely fail. All of these considerations need to apply to log data both when it being created, transferred and when it is stored.
Acceptance criteria	>> Architecturally, log events are stored within a different security domain than the software whose functionality is being logged. (Meaning outside the control of the system which is being logged.) >> Test that log files cannot be read or modified by unauthorised users; for audit logs, test tamperproofing against the persons whose activities are being logged. >> Test that auditable operations cannot be performed if the audit log cannot be written. >> Test that log events do not contain sensitive data unnecessarily.
Refinement questions	>> Where are our logs stored? >> Is the storage sufficiently separated from whatever we log in terms of hardware failures and successful attacks? >> How do we transfer the log data to wherever it is stored? >> Who should be able to read and modify our logs? In the case of an audit log, is it tamperproof against those persons whose activities are logged? >> Are log entries appropriately timestamped?

Standardisation of log events

As	**a service manager**
I want	that log events are logged in a standard way
so that	we can effectively analyse them.
Description	Log events need to be processed with existing log analysis software, without having to determine bespoke matching or parsing systems. This means that logging should use the logging facilities offered by the platform (syslog on Linux, standard logging APIs provided by your framework, etc.)
Acceptance criteria	>> System design states that logging uses the standard logging facilities of your platform and specifies the standard format.
Refinement questions	>> What are the standard logging facilities for our platform or company?

Patching & upgrading

As	**a service manager**
I want	that any software component that is part of the product can be quickly and cost-efficiently security patched
so that	we minimise the time we are vulnerable to an attack.
Description	From the moment we become aware of a security vulnerability in our product, it will be a race against time. A failure to provide a security fix quickly enough may lead to the company having to take drastic measures, such as shutting down the service. This timespan typically ranges from hours to weeks, but cannot be known in advance. It is important to note that the security vulnerability may be in third-party code, but we still need to be able to provide a fix.
Acceptance criteria	>> For all (third-party) code that we do not have a formal maintenance contract in place, we must have a formal patch back-porting process in place. >> We have automated test cases that try to patch each component in the running system.
Refinement questions	>> How can we deploy patches? >> Are there components that are exceptionally hard to patch (e.g., third-party binaries, firmware or ROM code, components requiring formal certification or digital signature)?

Change management

As	**a service manager**
I want	that changes between application versions are clearly documented
so that	unknown changes do not compromise our security.
Description	Developers need to be able to document what changes between application versions, because these may have an effect on security (e.g., changes in behaviour or configuration). Typically, this would mean accurate and complete release notes.
Acceptance criteria	>> There is an agreed way to create release notes.
Refinement questions	>> How can our development project tell what changes between its releases? >> Do all developers document changes on a sufficient level?

Component inventory and origin

As	**an information security manager**
I want	that all software components used are known and trusted
so that	we can effectively maintain our software.
Description	All the software components that are used in the product should be listed and the list needs to be maintained. The list needs also to have the current and historical version information (so that we know what we need to patch), and the patchlevel (so we know how we have changed the components). The company may maintain a centralised list of 3rd party components, and that list needs to be kept up-to-date as well. Components that need to be listed include third party and open source components, and those that are made by us, and include, for example, shared libraries, platforms, and support applications. We also need to agree that we can trust all the code, considering its quality and origin.
Acceptance criteria	>> No unauthorised third party code is included in the product. >> It has been defined who can authorise third party code to be used, and what checks need to be in place before that can happen. >> The list of components with version and patchlevel information exists in a known place. >> We know who the person is who tracks the vulnerabilities of any third party code we use, and this person is aware of the fact that we use this third party code.
Refinement questions	>> Do we have a clear idea of the dependencies (like libraries) we need to run our product? How about their dependencies? >> Do we know where our code originates from? How can we ensure the integrity of code we bring in from outside?

Monitoring

As	**a service manager**
I want	that the functioning of the product can be monitored
so that	we can react to abnormal activity.
Description	In order to be able to tell the internal state or health of a system, you may need to expose some metrics or state information of your system so that this information can be monitored. The simplest form could be that you have a system that responds "ok" within a predetermined response time, if the system is ok. Monitoring interfaces are near real-time.
Acceptance criteria	>> A clear list of product activities that need to (and can) be monitored exists, made available in product administrator documentation. >> There is an automated test that determines whether we get the monitoring data.
Refinement questions	>> What are the reasons for monitoring the functioning and activity of your product while it is in use? >> Given these reasons, what would we need to monitor? >> How can this need be communicated to whoever is supposed to do the monitoring? >> Do we need to implement some standard monitoring interface?

Enabling intrusion detection

As	**an information security manager**
I want	that the system can be monitored for intrusions
so that	we can detect intrusions and respond quickly.
Description	Attackers typically try to modify or replace critical system files to place backdoors or cover their tracks. They also may want to modify files that are served to clients in order to attack users. Network attacks, on the other hand, cause data traffic which is "not supposed" to be there, for example, port scans. There are various intrusion detection tools that can alert administrators when this sort of activity happens, but those tools need to be configured and told what (not) to look for. Developers should have a clear understanding of things that really belong to their system and this should be documented so intrusion detection can be configured.
Acceptance criteria	>> Any network traffic that should not exist can be detected (e.g., a final "deny all" rule of a host firewall triggers). >> Unauthorised modification of any important file can be detected (such files are, e.g., configuration files, log files, key files, binaries and WARs, and code that is sent to clients for execution such as ?JavaScript on web pages).
Refinement questions	>> Which parts of the system are such that unauthorised changes or modifications need to be detected? (E.g., binaries, scripts.) >> Can you accurately distinguish acceptable network traffic from unacceptable traffic? >> Do you know exactly what files and network traffic belongs to your system?

Malware protection

As	**an information security manager**
I want	that the system has real-time protection against malware
so that	the likelihood of infection is reduced.
Description	Systems that are likely to need anti-malware software are front-end servers (primary exposure to attackers from outside), administrator nodes and content servers (especially those who host content from outside).
Acceptance criteria	>> All systems that are deemed to be at risk run real-time anti-malware software.
Refinement questions	>> Which systems are in the need of anti-malware software, and what other ways there could be for protecting against malware? >> Could running anti-malware software cause problems, for example, if a log file would accidentally contain a malware fingerprint?

Content takedown

As	**a customer services manager**
I want	that we are able to quickly remove and reinstate specific content in a controlled fashion
so that	we comply with takedown laws, regulations and contracts.
Description	Content takedown requests result from civil (e.g., copyright violations) or criminal (e.g., child pornography) legislation and in some cases may involve a gag order (we may not be able to divulge the fact that we have a takedown request). Content takedown requests (e.g., the DMCA in the US or a court order) may force us to remove a specific piece of content quickly. The same regulations may cause us a need to reinstate the same content quickly. We should be able to do this without causing collateral damage, for example, we might not want to close a user's account just because of one alleged copyright violation which is unproven. Content takedown usually needs to follow a very specific externally mandated process and leave a sufficient audit trail.
Acceptance criteria	>> Our support/administration tools support these externally mandated content takedown processes. >> Use of takedown functionality leaves an audit trail.
Refinement questions	>> Which content takedown legislation and contracts might apply to us? >> Which content is potential candidate for a takedown? >> Does our information architecture support the takedown of specific content items?

Legal interception and access

As	**a company officer**
I want	that we can provide legal access according to applicable laws in a cost-effective way
so that	we comply with legal interception and access laws and regulations.
Description	A court may order us to hand over data to be used as evidence in a criminal case. In some countries, if we wish to operate in that market area, we might be ordered to provide some level of access to data or data flows. Company officers may in some cases be held personally accountable for failure to comply. It is important to note that irrespective of technical measures, any access of this kind must be reviewed and approved by a person who has been specifically authorised to do so, and leave an audit trail.
Acceptance criteria	>> The specification and features of this functionality has been specifically approved by company legal counsel. >> Our privacy policy is in line with the implemented feature. >> Use of legal interception and access leaves an audit trail.
Refinement questions	>> Which legal interception and access laws and regulations might apply to us? >> Which content or data flows are potential candidates for legal interception or access? >> How can we protect against abuse of these features?

Backups

As	**a user**
I want	that the system and necessary data can be restored after an incident
so that	my data is not lost.
Description	We need to be able to restore (or recreate) the system and the data at request. This can be a result of faults, failures, malicious acts or user errors. Systems and their configuration could be recreated from deployment packages, and in some cases, some data could be deemed not to require backing up. Data must be stored in a way which does not preclude backing up, and it must also be possible to restore it in a way that it stays consistent.
Acceptance criteria	>> A complete inventory of all things that need to be backed up exists, with the backup frequency and retention times.
Refinement questions	>> What needs to be backed up? >> What does not need to be backed up? >> How often do backups need to be done and for how long do the backups need to be stored? >> When data is restored, how old can it be? >> How can we ensure the integrity of restored data?

Cryptographic key management

As	**an information security manager**
I want	that the lifecycle of cryptographic keys is secure
so that	we can have any trust in the security features of the system.
Description	Cryptographic keys are used in various roles, for example, in security protocols, authentication control and session control. Cryptographic key management can fail in multiple phases of their lifecycle: creation, provisioning, distribution, storage, use, replacement, and destruction. Software developers must address this area properly, as an incorrect implementation cannot usually be fixed in production. Cryptographic engineering requires expert knowledge.
Acceptance criteria	>> Test that all secrets are protected correctly both in memory and in storage. >> Randomness tests for session tokens and similar random identifiers exist. >> Verify that key and random token creation use sufficiently strong randomness sources.
Refinement questions	>> Where does your product use cryptography (encryption, random numbers such as session tokens, integrity checks, hashing)? >> How does your product use cryptographic keys? >> How are cryptographic keys protected from compromise? >> How are random numbers seeded?

Platform hardening

As	**an information security manager**
I want	that the platform minimises attackers' possibilities
so that	the likelihood of a breach is minimised.
Description	Best practices guides typically recommend disabling or removing all unnecessary functionality (default servlets, admin consoles, daemons, processes, SDKs, example files), running processes with least privilege, using operating system provided key stores, isolating unrelated components (using different users, chroot, sandboxing, etc.), deny by default firewall policies, changing all default passwords, and removing or disabling debugging functionalities. Each platform has its own peculiarities and should have its own hardening standards.
Acceptance criteria	>> A hardening guideline document exists. >> The platform has been hardened according to a hardening guideline. >> Tests exist that verify that the hardening has been done. >> Tests exist that check for any configuration changes that we have introduced only for testing purposes.
Refinement questions	>> What is / what are the applicable hardening guideline(s) for our product, considering the whole software stack from the underlying hardware up to our application? >> Are there operating system provided services (for example, key storage services) that could be leveraged? >> Are there any configuration items (e.g., open SSH port) that are only needed for testing, and should be removed in production deployment?

Network architecture

As	**a service manager**
I want	that the network architecture is secure and is leveraged to provide security
so that	attacks are easier to contain.
Description	Incorrect network configuration can severely compromise your application, even if your application would have a good security model. For example, if a connection to a database is insecure even though your application assumes it is secure. On the other hand, a good network configuration will significantly enhance your security at a low cost. For example, you can isolate database machines so that they can only communicate with your application.
Acceptance criteria	>> A Data Flow Diagram of the system exists and it shows network trust boundaries. (A Data Flow Diagram is usually produced in architectural threat analysis.) >> Network scanning tests exist that validate the network deployment.
Refinement questions	>> What implicit assumptions are we making about the network? (For example, do we assume that some connections are "trusted")? >> Does our application support network segmentation? Which type of segmentation would offer additional security? >> How and where do we deploy firewalls?

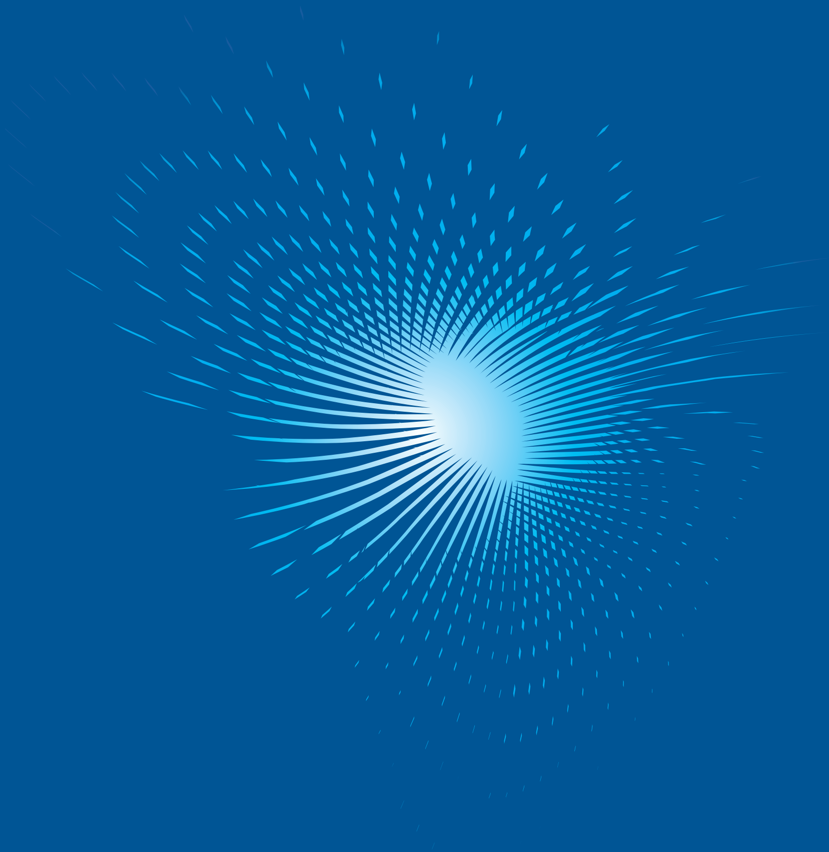
Administration access

As	**an information security manager**
I want	administrator access to be protected by strong authentication and access controls
so that	we can have any trust in our user account model.
Description	Insufficient security for administrator access may cause complete system compromise. Because of this, access to administrator functionality must be restricted and administrators must be authenticated with strong authentication. Passwords alone are not regarded as strong authentication.
Acceptance criteria	>> List of administrator interfaces and administrator roles is documented. >> Administrator interfaces have been subjected to exploratory security testing.
Refinement questions	>> What roles on the system are "administrative" roles? >> Who are our administrators and from where do they need to administer the system? >> How do we control access to administrative interfaces?

Separate development and production environments

As	**a service manager**
I want	to have separate development and production environments
so that	production environment and data are not put at risk.
Description	Different environments should not interfere with each other: development should not affect testing, and testing should not affect production. Also, production data should not be exposed through testing environments. However, from security features perspective, the environments should be identical so that test results from the test environment are really valid also in the production environment.
Acceptance criteria	>> Different environments and instructions to set them up have been documented. >> Setting up environments has been automated.
Grooming questions	>> What is our development process starting from single developer's workstation and ending to the production environment installation? >> What are the purposes of the environments? What are the significant security differences between the environments? >> What are the rules of moving releases from one environment to another? >> Do any of our non-production environments have any dependencies to any other production system (e.g., authentication systems, e-mail, CRM, etc.)

Cloud Software Finland



How can you manage security and privacy risks and requirements in agile software projects? What is fuzz testing? Need insight on security metrics or trust management?

During the four-year Finnish Cloud Software Program, people from multiple participating companies and universities developed interesting new approaches to software security, privacy, metrics, and trust. The results have been disseminated through numerous academic articles, which are usually not very approachable to the laymen. Instead of letting the articles get buried in obscure conference proceedings, expensive journals and project archives, the authors decided to publish them as an easily approachable collection for everyone who is interested in these topics.

ISBN: 978-952-62-0341-6