



jamk.fi

Digia Secure SDLC

Implementation of Agile SDLC with Security and Privacy

Mikko Jylhä

Master's thesis

October 2018

Technology, Communication and Transport

Master's Degree Programme in Information Technology, Cyber Security

Author(s) Jylhä, Mikko	Type of publication Master's thesis	Date 6.11.2018 Language of publication: English
	Number of pages 49	Permission for web publication: Yes
Title of publication Digia Secure SDLC Implementation of Agile SDLC with Security and Privacy		
Degree programme Master's Degree Programme in Information Technology, Cyber Security		
Supervisor(s) Hautamäki, Jari; Karjalainen, Mika		
Assigned by Digia Finland Oy		
Abstract <p>Due to ISO 9001 standard revision, Digia implemented a major rewrite of its Core Process Model (CPM) during 2017 - 2018. As part of the program, Digia agile SDLC (Security & Privacy Development Life Cycle) was created. The target was to enhance the agile software development process model so that it is adequate for authoring software with high security and privacy needs. As a result, the process produces products or services with acceptable and significantly lower risk levels in regard to security and privacy.</p> <p>The main objective was to create the adaptation of security and privacy features to agile development forming the Digia agile SDLC as part of CPM. Other objectives were to evaluate the implementation against the latest development in the field of secure agile development, to raise development ideas and a synthesis based on findings as well as define and publish the new process publicly to other interested parties (as Digia Open Secure Agile SDLC).</p> <p>The actual process development methodology was based on small workshops where the key principles the author wanted to include in the Digia agile SDLC were fitted to Scrum model. Later a literature review and analysis with the synthesis was done. The main result was and is a live process that is currently being trained in large scale in Digia.</p> <p>Based on the analysis, building a secure Scrum has been an even more discussed topic in the academic world than what was thought. Still, there is no common standard adopted. The Digia approach is based on simplicity and keeping the agile Scrum principles mostly intact. As a conclusion, the literature review would most likely have impacted the end results. This methodology produced original ideas that are similar to the approaches that others.</p>		
Keywords/tags (subjects) Data Security, Security, Privacy, Agile Development, Scrum, GDPR, SDLC, Digia Open Secure Agile SDLC, Development Process		
Miscellaneous		

Tekijä(t) Jylhä, Mikko	Julkaisun laji Opinnäytetyö, ylempi AMK	Päivämäärä marraskuu 2018
	Sivumäärä 49	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x
Työn nimi Digia Secure SDLC Implementation of Agile SDLC with Security and Privacy		
Tutkinto-ohjelma Master´s Degree Programme in Information Technology, Cyber Security		
Työn ohjaaja(t) Jari Hautamäki, Mika Karjalainen		
Toimeksiantaja(t) Digia Finland Oy		
<p>Tiivistelmä</p> <p>ISO 9001 -standardin päivityksestä johtuen Digia toteutti vuosien 2017 - 2018 aikana merkittävän päivityksen sen ydinprosessimalliin (Core Process Model, CPM). Osana tätä kokonaisuutta luotiin Digia agile SDLC (Digia Security & Privacy Development Life Cycle). Tavoitteena oli parantaa ketterää ohjelmistokehitysprosessia siten, että sen avulla saataisiin tuotettua ohjelmistoja, joilla on korkeat tietoturva ja -suojavaatimukset. Lopputuloksena prosessi tuottaa tuotteita tai palveluita, joissa on hyväksyttävissä oleva ja merkittävästi alempi riskitaso.</p> <p>Päätavoite oli luoda sovitus tietoturva- ja suojaominaisuuksista ketterään kehitysmalliin Digia agile SDLC:hen osana CPM:ää. Muut tavoitteet olivat verrata kehitettävän mallin toteutusta muihin turvallisen ketterän kehityksen menetelmiin viime ajoilta, löytää jatkokehitysajatuksia ja tehdä synteesiä arviointiin pohjautuen sekä määrittää ja julkaista työn tulokset myös muille kiinnostuneille.</p> <p>Varsinainen prosessikehitysmenetelmä perustui työpajoihin, joissa Digia agile SDLC:ään halutut keskeisimmät piirteet sovitettiin Scrum-malliin. Tämän jälkeen tehtiin kirjallisuuskatsaus ja -analyysi sisältäen synteessin. Lopputuloksena oli ja on elävä prosessi, jota koulutetaan laajasti Digialla.</p> <p>Analyysistä kävi ilmi, että turvallinen Scrum on ollut laajemmalti keskusteltu aihe akateemisessa maailmassa, kuin oletettiin. Tästä huolimatta aiheesta ei ole yhteisesti hyväksyttyä standardia käytössä. Digian lähestymistapa perustui yksinkertaisuuteen ja Scrumin periaatteiden eheänä säilyttämiseen. Loppuhavaintona oli, että kirjallisuuskatsaus olisi todennäköisesti vaikuttanut lopputulokseen. Menetelmä tuotti kuitenkin uusia ideoita, jotka ovat samoilla linjoilla muiden kehittelemien kanssa.</p>		
<p>Avainsanat (asiasanat) Tietoturva, Turvallisuus, Tietosuoja, Ketterä kehitys, Scrum, GDPR, SDLC, Digia Open Secure Agile SDLC, Kehitysprosessi</p>		
Muut tiedot		

Contents

1	Introduction to Digia Secure SLDC.....	10
1.1	Background of the Project	10
1.2	Objectives and Scope.....	10
2	Theoretical Basis	11
2.1	Research Method	11
2.2	Work Methods in Practice	12
2.3	Scrum in Short	13
2.4	Principles to be Included in SDLC.....	14
2.4.1	Any Secure and Privacy SDLC Should be Documented.....	15
2.4.2	Standards Should be Used	15
2.4.3	Secure SDLC is not Static but Needs Directing	16
2.4.4	Risks Should be Evaluated.....	16
2.4.5	Separate Features, Stories and Defects from Security and Privacy Items 16	
2.4.6	Secure Coding Training should be Available	17
2.4.7	Source Control System with Access Control should be Used	17
2.4.8	Releases Should be Versioned and Controlled.....	17
2.4.9	Separation of Environments Should be in Place	18
2.4.10	Separation of Duties	18
2.4.11	Set of Standard Tools	18
2.4.12	There Should be a Rehearsed Incident Management Process	18
2.4.13	Improving the Security and Privacy Culture.....	19
2.5	Review of Related Current Research.....	19
2.5.1	Secure Scrum: Development of Secure Software with Scrum	20
2.5.2	Security backlog in Scrum Security Practices	21

2.5.3	Busting a Myth: Review of Agile Security Engineering Methods	24
2.5.4	A Systematic Mapping Study on Security in Agile Requirements Engineering.....	26

3	Implementation of the New Digia Secure SDLC.....	27
3.1	Implementation Work in Practice	27
3.2	Changes to Scrum.....	28
3.2.1	The Decision Criteria	29
3.2.2	Process Flow	30
3.2.3	Key Decisions before Entering Secure Agile Process.....	31
3.2.4	Secure Agile Process Initiation	32
3.2.5	Product Owner Readiness Evaluation.....	32
3.2.6	PO Security Training.....	33
3.2.7	Adding a Support Person.....	33
3.2.8	Backlog Security and Privacy Refinement.....	33
3.2.9	Security Requirements in Agile Development.....	35
3.2.10	Privacy Requirements in Agile Development.....	35
3.2.11	Iterative Progress.....	36
3.2.12	Security Audit	37
3.2.13	Release	37
3.3	Primary Support Tools	37
3.4	Key Security and Privacy Design Principles.....	38
3.4.1	Keep It Simple.....	38
3.4.2	Defence in Depth	38
3.4.3	Privacy by Design	38
3.5	Rollout of the Process.....	39

4	Evaluating the Resulting Process	39
4.1	Original Security Principles	39
4.2	Actual Guiding Principles in the Process	41
4.2.1	Additional Reflection to Literature	43
4.2.2	Reflection to Original Design Problems	45
5	Conclusions and Further Ideas	46
	References	47
	Appendices	49

Figures

Figure 1. Secure Scrum phases and the integration to Scrum.....	20
Figure 2. Enhanced Scrum process with additional SB and SM	23
Figure 3. Distribution of agile security research, type per year.....	26
Figure 4. Digia Secure SDLC general process flow.....	31

Tables

Table 1. Security Principles according to Azham, Ghani, & Ithnin.....	22
Table 2. Comparison of security activities	25
Table 3. An exemplary table of decision criteria for using Secure SDLC	29
Table 4. A Summary of post analysis.....	40

Acronyms

BIA

Business Impact Analysis

ISO

International Standardization Organization

CPM

Core Process Model

DoD

Definition of Done

GDPR

General Data Protection Regulation

PCI DSS

Payment Card Industry Data Security Standard

SDLC

Software Development Life Cycle, also: Systems Development Life Cycle

SoD

Separation of Duties

XP

Extreme Programming

Terms

Agile development

Software development done in the spirit and guided by the principles of the agile manifesto.

Agile Team

In scrum the agile team includes a development team, PO and possibly a Scrum Master. Agile team is a cross-functional team that has all the capabilities required for realizing the development effort. This team as a whole should be self-organizing and should have the autonomy to perform its work.

Bug Bounty program

A bug bounty program is a deal offered by many websites and software developers by which individuals can receive recognition and compensation for reporting bugs, especially those pertaining to exploits and vulnerabilities. These programs allow the developers to discover and resolve bugs before the general public is aware of them, preventing incidents of widespread abuse (Bug bounty program 2018).

Business Impact Analysis (BIA)

A Business impact analysis differentiates critical and non-critical organization functions/activities. Critical functions are those whose disruption is regarded as unacceptable. Perceptions of acceptability are affected by the cost of recovery solutions. A function may also be considered critical if dictated by law (BIA 2018).

Core Process Model (CPM)

The Core Process Model is Digia's quality management system that guides its operations. The CPM describes company's key processes and practices. The system ensures that, by following a certain process, tasks are performed correctly and consistently, while achieving the desired level of quality.

Definition of Done (DoD)

When a Product Backlog item or an Increment is described as "Done", everyone must understand what "Done" means. Although this may vary significantly per Scrum Team, members must have a shared understanding of what it means for work to be complete, to ensure transparency. This is the definition of "Done" for the Scrum Team and is used to assess when work is complete on the product Increment (Pohl & Hof, 2015).

Development Team

A cross-functional team of developers and designers that actually produces the product.

The people that work to implement the backlog items. The development team should also be self-organizing, which means, that if the PO is not doing work as one of the developers, he is not entitled to tell them how to change the backlog items into working software, beyond the functional descriptions and quality control constraints written in the backlog (Pohl & Hof, 2015).

Epic (in Scrum)

An Epic can be defined as an amount of work that has one common objective. It could be a feature, or a requirement. In Product Backlog, it is a placeholder for a required feature with few lines of description (Pohl & Hof, 2015).

Microsoft Secure Development Lifecycle

A process model defined by Microsoft promising that "The Security Development Lifecycle (SDL) is a software development process that helps developers build more secure software and address security compliance requirements while reducing development cost" (Techopedia, n.d.).

Product or Project Backlog

The Backlog is an ordered list of everything that is known to be needed in the product. It is the single source of requirements for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering (Pohl & Hof, 2015).

Product Owner (PO)

Product Owner represents the entirety of customer in an agile team. His job is to ascertain that the backlog items are clearly expressed and well-ordered so that the value the team produces is optimized while still preserving the technical and conceptual integrity of the product they are doing.

Ideally, the Product Owner is the sole authority on the order of backlog items and in accepting the development team's implementation of the backlog items i.e. the work to be done. The first part here means that he gets to say what is to be achieved and the second, what level of quality is the desired and acceptable.

Rigid development

Traditional plan-driven or waterfall model software development

Scaled Agile Framework (SAFe)

SAFe® for Lean Enterprises is a knowledge base of proven, integrated principles, practices, and competencies for Lean, Agile, and DevOps. More than the sum of its parts, SAFe is a scalable and configurable framework that helps organizations deliver new products, services, and solutions in the shortest sustainable lead time. It's a system that guides the roles, responsibilities, and activities necessary to achieve a sustained, competitive technological advantage (Leffingwell, Jemilo, Zamora, O'Neill, & Yakuma, 2014; Scaled agile, 2018; Scaled Agile Inc, 2017).

Separation of Duties (SoD)

Separation of duties (SoD; also known as Segregation of Duties) is the concept of having more than one person required to complete a task. In business the separation by sharing of more than one individual in one single task is an internal control intended to prevent fraud and error. The concept is alternatively called segregation of duties or, in the political realm, separation of powers. In democracies, the separation of legislation from administration serves a similar purpose. The concept is

addressed in technical systems and in information technology equivalently and generally addressed as redundancy (Separation of Duties 2018).

Sprint review

In Scrum, each sprint is required to deliver a potentially shippable product increment. This means that at the end of each sprint, the team has produced a coded, tested and usable piece of software. So at the end of each sprint, a sprint review meeting is held. During this meeting, the Scrum team shows what they accomplished during the sprint. Typically, this takes the form of a demo of the new features (Mountain Goat Software, n.d.).

Story (in Scrum)

User stories are one of the primary development artifacts for Scrum and Extreme Programming (XP) project teams. A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it.

1 Introduction to Digia Secure SLDC

Software is everywhere. One's laundry machines, refrigerators, tax records and grocery store data are running on and being processed by software. Where there is software, there are bugs that may lead, and often do lead, to successful attacks on software systems.

Even though Digia does not make software for refrigerators, development at Digia is guided by the changing day-to-day work and needs of its customers. One of Digia's customer promises is to work so that the end results function as agreed, which gave the ultimate motivation for this study: to enhance the company's software development process to produce secure software that takes into account both security and privacy.

1.1 Background of the Project

Due to ISO 9001 standard revision, Digia implemented a major rewrite of its Core Process Model (CPM) during 2017 - 2018. As part of the program, Digia Secure SDLC (Software Development Lifecycle) was created. The target was to enhance software development process model so that it is adequate for authoring software with high security and privacy needs. As a result, the process produces products or services with acceptable and significantly lower risk levels in regard to security and privacy. The target was software development agnostic: both agile and waterfall software development processes had to be addressed.

As it can be seen, the original motivation was to improve software development process in regard of security and privacy. The latter had already got plenty of focus in 2017 due to the forthcoming European GDPR legislation. Secondary objectives were added later as explained below.

1.2 Objectives and Scope

The main objective was to create an adaptation of security and privacy features to software development process, the main focus being on the agile part that forms the Secure Agile SDLC as part of "Digia High Security & Privacy Agile Software

Development (Digia Secure SDLC)". On the other hand, also traditional waterfall model - plan-driven, as it is sometimes referred to - was to be addressed in a similar manner. The later introduced secondary objectives were to evaluate this implementation against the latest development in the field of secure agile development, to raise development ideas and carry out a synthesis based on findings as well as define and possibly publish the process as what is called Digia Open Secure Agile SDLC to other interested parties.

The work is structured as follows. First, the theoretical basis is introduced, which briefly describes agile Scrum, the principles that guided the development of secure SDLC and contains a review of related current research. Then the implementation of the Digia Secure SDLC is described. After that, the Digia Secure SDLC is evaluated against the guiding principles and the current research review. Lastly, conclusions and further ideas are presented.

2 Theoretical Basis

2.1 Research Method

The research method to be used was chosen to be Design Science Research (DSR) method. DSR method was chosen as in general it is based on a pattern of analyzing a problem, defining and developing an artefact to solve the problem and then evaluating the created artefact (Johannesson & Perjons 2014).

Design science can be contrasted to empirical science, such as natural and social science. In empirical science, researchers describe, explain, and predict. In design science, researchers also design and develop artefacts for improving practices. (ibid.)

DSR method is well suited for the kind of research that focuses on measuring the effectiveness of artefacts, which in this case are the security controls proposed and investigated in the papers (Niemelä, Rantonen, & Huotari, 2015). In situations where the controls are few or non-existent, DSR works well.

It should be noted that the purpose of traditional design in systems development is also to create an artefact that fulfils the needs and requirements of some

stakeholders, possibly only for a local practice. Contrary to this, design science research aims at producing and communicating new knowledge that is relevant for a global practice. This difference raises three additional requirements for design science (Johannesson & Perjons, 2014).

First, the purpose of creating new and generalizable knowledge requires that design science projects make use of research strategies and methods. Secondly, the results produced by design science research must be related to an existing knowledge base. Thirdly, the new knowledge should be communicated to both practitioners and researchers (Johannesson & Perjons, 2014). All of these three principles work well with original research problems, which are described in the next subchapter.

2.2 Work Methods in Practice

The agile development method Scrum lacks security features and controls. This fact creates the main research problem and objective of this work: how is agile Scrum changed or enhanced so that it produces secure software?

The main question raises a few additional (secondary) questions (problems): what is needed in a software process in general for it to produce secure end results? Do the same changes or enhancements enable the process to produce end results that have required privacy level and controls as well? How have others approached this problem? How should the author of this thesis contribute this research to the community and provide value to general public?

Design Science Research method was used by first defining and analyzing the research problem (already introduced in chapter 1.2 above). Then a solution (artefact) for the problem was developed according to DSR principles, and then the solution was tested whether it is able to provide a satisfactory result in solving the problem.

The testing could be performed with acceptable accuracy by comparison of related work in academic papers and similar sources, as extensive measuring was not yet possible. In other words, the chosen research strategy focused on existing documents and reliable data. This strategy is briefly described in DSR by Johannesson & Perjons (Johannesson & Perjons, 2014).

The academic documents were sourced mainly from online computer science libraries: Researchgate, ACM Digital Library, Theseus, Springer, IEEE Explore. The searches used typical keywords e.g. Scrum, agile, security, secure software. Most of the academic papers were not available publicly; hence, they were requested personally from the authors with a considerable success rate.

The main reason for choosing the above mentioned online computer science libraries was that they only publish peer-reviewed scientific papers and research work. In practice, this gives the credibility for the papers that were to be compared against the author's own work.

2.3 Scrum in Short

Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value (What is Scrum?, 2018; Schwaber & Sutherland, 2013).

In Scrum, developers are grouped in small teams that are given predefined autonomy to develop software. One of the assumptions is that all developers can implement all tasks at hand (Pohl & Hof, 2015). The latter assumption that Pohl & Hof (2015) make is a strong one; when developing software with security focus, the practice has shown that unfortunately all developers are not able to implement all tasks. However, the thesis does not discuss the possible weak points of Scrum further, but instead continues introducing Scrum in general.

In Scrum, development schedule is split into increments called sprints that have a fixed length typically between 2 and 4 weeks.

The development team develops an increment of the current software version during a sprint. The increment typically consists of an agreed number of features. These features are written in user stories and stored in a prioritized list called Product or Project Backlog. In essence, the user stories are used to document client or other requirements for a project.

Each sprint is planned in a planning session and user stories typically from the top of the backlog are selected and divided into smaller tasks. These tasks are then stored

in the Sprint Backlog. Scrum includes an important role called Product Owner (PO). The PO is considered the single point of communication between a customer and the development team. He or she also prioritizes the backlog, i.e. the order and whether or not some feature is implemented. Standard Scrum does not include any security-specific parts and therefore motivates the research work here.

The key term in Scrum is so-called Definition of Done or DoD. When a backlog item or an increment is described as “Done”, everyone in the team must understand what “Done” means. The team members must have a shared understanding of what it means for work to be complete to ensure transparency. This is the definition of “Done” for the Scrum Team and it is used to assess when the work is complete on the product Increment. Typically Definition of Done is a written statement in Scrum team documentation, and the definition is evaluated and possibly revised at times (Schwaber & Sutherland, 2013; “What is Scrum?,” 2018).

2.4 Principles to be Included in SDLC

The original key security principles to be included in the forthcoming secure and privacy development process were mostly collected and summarized in notes during the year 2017 when the actual process development work started. The principles were based on personal earlier M.Sc. studies at Tampere University of Technology, interviewing other companies in similar situations at e.g. ISC2 (International Information System Security Certification Consortium) seminars during 2017-2018 and years of work in software and security industry. Therefore, the principles formed basically a strong subjective description of the industry’s best practice. It was seen enough to implement the process based on this best practice and the key learnings from Digia GDPR project that took place in fall 2017 and winter 2018. The latter focused on privacy issues in software and formed a key part within “Digia High Security & Privacy Agile Software Development (Digia Secure SDLC)”. It was after the actual first implementation that it was concluded that an academic review would be needed to get improvement ideas and to get a more scientific comparison to other work done outside of Digia.

The following subchapters briefly introduce each key principle that was intended to be covered in some way in the new secure SDLC or its support tools.

2.4.1 Any Secure and Privacy SDLC Should be Documented

Secure development should be based on repeatable process and patterns. The process should be documented formally if possible. The process should be easy to understand, simple and short, concrete, and digitally available to all users.

The process should include steps for security and privacy design and design review. The design should be based on security architecture and the process should require one to be made for each case. Product Owner (PO) or Business Owner (BO) should attend the reviews. If possible and based on overall risk assessment of the nature of the project, a security architect or at least a security specialist should attend the reviews, even if he or she is not actively working within the project.

The practice has shown that the sort of design review process described above results in threat modelling type of work. This implies that the developers need to be trained to perform threat modelling. One of the well-known threat modelling methods is STRIDE by Microsoft (Geib, Casey, & Santos, 2017; Mohamood, 2017). Further explanation of STRIDE is out of scope of this study. However, it can be seen that the responsibility for coordinating such training should be in corporate security.

Pattern libraries and examples should be part of the tools section related to secure SDCL. Policies and related standards should be easily available to development teams.

2.4.2 Standards Should be Used

Open and internationally recognized standards can guide or give support in defining a secure development process. Good standards to evaluate are, for example, ISO 27001 that formally defines requirements for information security management, ISO/IEC 27034 Information technology - Security techniques - Application security, OWASP, PCI DSS and Open SAMM. These standards are just examples and should be evaluated against the enterprise strategy, the field of industry and the applicable context.

2.4.3 Secure SDLC is not Static but Needs Directing

A secure SDLC process is not and should not be static; it needs direction and constant improving, refining and upkeep. The key directing factors for a secure SDLC process are a framework for company general policies, policy-based proper access control even within the development process, proper policy guided backups and connection business continuity process (e.g. ISO 22301), company guidelines and in general, standards.

2.4.4 Risks Should be Evaluated

Risk assessments and risk-based decisions related to security and privacy should be a part of a secure SDLC process. Developers and persons leading or guiding development should have tools and skills to perform risk assessments at least on sprint backlog item level.

Security risk assessment should be an integral part of the process. In plan driven models, security risks should be evaluated several times or regularly during the proposed project structure. In agile models, security risks should be evaluated in each sprint and with every story or similar.

There should be a requirement to assess risks if a certain number of changes is made to e.g. program code. It should be analyzed, for example if the change has an effect on security or privacy and then act accordingly.

2.4.5 Separate Features, Stories and Defects from Security and Privacy Items

All mature software development processes include tracking of features or stories, defects, bugs, and security or privacy issues. These are typically tracked in issue tracking system such as Atlassian Jira.

As a security requirement for a development process when designing features or writing stories, each item should be evaluated, whether it affects security and privacy or not. If an effect on security can be seen, the item should be tagged for later or immediate risk security and privacy risk assessment.

As a security requirement for a development process there should be a clear separation of common bugs from security and vulnerability issues in bug tracking. A categorization based on the severity of the issue should be in place. A policy should be in place that directs how much time is allowed after a vulnerability or security issue is found, and that a risk assessment and decisions on how to act are done. The same policy should give guidance on severity estimation in risk assessments.

2.4.6 Secure Coding Training should be Available

Architectures and programming languages vary, and projects are different from one another. There should be secure programming training available to educate developers and other stakeholders that considers the changing landscape of software industry. Security training should be available as a tool or auxiliary service related to any secure SDCL process.

2.4.7 Source Control System with Access Control should be Used

Source Control System with role-based access controls and proper audit trail should be used. Also, it should be documented and controlled how so-called forks, branches etc. are used. It should be clear, for example, who accepts features as done and who accepts release versions and when.

Every line of code should go through a review performed by another person for defects and vulnerabilities. Pair coding should be promoted if not required, as it promotes reviewing by another person.

Each line of code should be traceable to the original user story in Scrum backlog. The traceability should have details on e.g. time, who implemented the story and who reviewed the implementation, who approved the feature to testing and production.

2.4.8 Releases Should be Versioned and Controlled

Releases should be versioned and documented rules should exist that define which versions are maintained. There should be a policy in place that defines which e.g. major versions of a product or service get security fixes. A similar policy should be

created in overall maintenance of the code. In effect, the requirement is to formally define the lifecycle for software or service.

2.4.9 Separation of Environments Should be in Place

There should be separate environments for development, testing and production. All these environments should have a maturity requirement policy and a predefined set of rules, when features or components are moved from development to testing and testing to production.

2.4.10 Separation of Duties

Generally speaking, the phases of development process should have a clear separation of duties (SoD) principle in place for security control but also to help proving different kinds of liabilities afterwards.

2.4.11 Set of Standard Tools

There should be a company standardized set of tools that are used e.g. in risk assessments, privacy impact assessments required in the software development process, not just the typical bug tracking systems and source control. One of the possible tools that enterprises typically lack could be a kind of compliance roster that tracks different kinds of compliance requirements per product or service. This roster or table could easily give an overall picture which development projects need to adhere, e.g. PCI DSS requirements.

Automated tools should be used as much as possible to minimize possibilities for manual errors and manual work. Security testing tools such as fuzzing tools and possibilities to have the code externally security tested should be available to development projects.

2.4.12 There Should be a Rehearsed Incident Management Process

Things happen in real world, and when things happen, people should be ready to deal with the situation. Incident management process should be defined, assigned and rehearsed regularly. The details of mature incident management system are out of the scope of this document.

2.4.13 Improving the Security and Privacy Culture

Culture is a sum of many factors, however, with small steps, training and awareness the culture can be changed bit by bit. One example of security culture changing acts is to test software in bug bounty competitions. Security and privacy can be a fun part of software development, not just a burden.

2.5 Review of Related Current Research

There have been quite some papers and thesis work done revolving around secure agile software development. Plenty of papers are about improving security features in agile XP method. Such work includes Security Engineering and eXtreme Programming: An Impossible Marriage (Wäyrynen, Bodén, & Boström, 2004) and Extreme Programming Security Practices (Ge, Paige, Polack, & Brooke, 2007). These papers were abandoned as they were slightly obsolete and concentrated on agile XP, not agile Scrum.

As there were so many papers and theses available, it became apparent that the research strategy had focus slightly, and more selection criteria had to be applied. It was decided that the selected papers should describe the use of Scrum in a secure way or describe a similar extension to Scrum. Two more recent papers were selected to represent the plethora of science papers from the last 15 years.

In addition to the selected two Scrum related papers, two articles were found that aim to summarize the current (at the time of their writing) research on secure agile methods. These two are briefly introduced and their results are used in a synthesis in Chapter 4 Evaluating the Resulting Process.

The chosen papers are presented shortly in the following subchapters. Later, in Chapter 4, these papers are compared to the implementation created in this study. One paper was specifically a proposed implementation of the same subject (Pohl & Hof, 2015); therefore, it is the first to be covered in more detail.

2.5.1 Secure Scrum: Development of Secure Software with Scrum

The article by Christoph Pohl and Hans-Joachim Hof (2015) starts by introducing Scrum shortly. Then the motivation for secure Scrum is covered. Pohl & Hof also focus on identifying security relevant parts of a software project as was done in this thesis project.

Following figure (Figure 1) illustrates the key phases of Secure Scrum and their relation to standard Scrum.

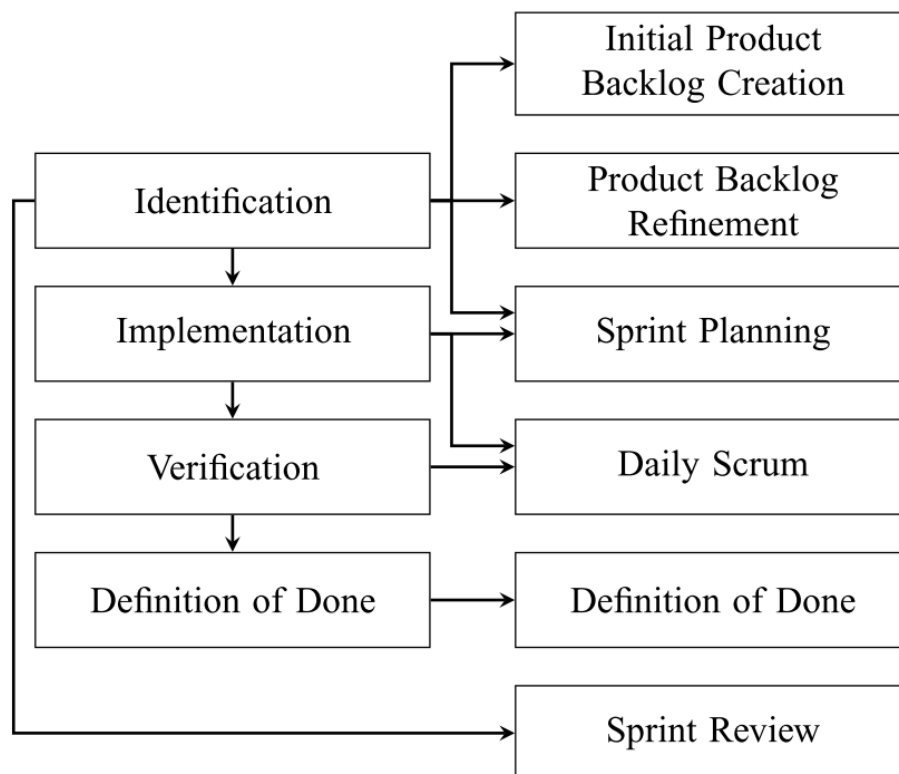


Figure 1. Secure Scrum phases and the integration to Scrum

Secure Scrum aims at achieving an appropriate security level for a given software project. According to the authors, the term “appropriate” was chosen to avoid the costly over engineering of IT security in software projects. The authors state that the definition of an appropriate security level is the crucial point in resource efficient software development. Secure Scrum helps developers to identify appropriate security testing means for security relevant parts of a software project (Pohl & Hof, 2015).

Secure Scrum has four components that are put on top of the normal Scrum: identification component, implementation component, verification component and Definition of Done component. (ibid.)

The implementation component that, according to the paper, is used in Scrum Sprint Planning and Daily Scrum meetings, raises the awareness of the Scrum team for security issues during a sprint (ibid.).

The verification component managed during Daily Scrum meetings should ensure that the team members are able to test the software with the focus on IT Security.

The Definition of Done component enables the developers to change the Definition of Done for security related issues as postulated in standard Scrum (ibid.).

In the first step of Secure Scrum, identification step, stakeholders or PO and Scrum team rank the different user stories according to their loss value (ibid.).

In Secure Scrum, after finalization of the identification component, team members and stakeholders have a common understanding of security risks in the Product Backlog. To document this understanding in the Product Backlog, Secure Scrum uses so called S-Tags. An S-Tag identifies Product Backlog items that have security relevance with a Marker called S-Mark (ibid.).

Secure Scrum changes the verification process so that in a task done during the same sprint and by the same developer, the verification must be a part of the task. This ensures that the verification must be a part of the Definition of Done (ibid.).

According to Pohl & Hof (2015), it is possible that a developer does not have the required knowledge for verification, or the verification needs external resources, extra time for testing, or anything else that hinders an immediate verification. In this case, the verification cannot be a part of the Definition of Done (Pohl & Hof, 2015).

2.5.2 Security backlog in Scrum Security Practices

The article by Ghani, Azham & Ithnin (Azham, Ghani, & Ithnin, 2011) starts by introducing typical agile requirements: flexibility, speed, leanness, learning and responsiveness. Then the article continues through a literature review to present the problems with Scrum and security concerns. The writers point out that there are no

standards to secure Scrum and the typical security requirement oriented practices have evolved before agile methods (ibid.).

The key security issues Azham, Ghani, & Ithnin (2011) point out in Scrum are related to a too short release cycle to address security issues, security requirements changing Scrum, absence of documentation due to Scrum nature, lack of guidelines in regard of security requirements handling, and last but not least, the lack of security awareness in a typical Scrum team. (ibid.)

The authors continue to introduce a key element in guiding secure development, namely software security principles. These are listed in article table and reproduced in Table 1 below. (ibid.)

Table 1. Security Principles according to Azham, Ghani, & Ithnin

	Least privilege	Failing securely	Securing the weakest link	Defence in depth	Separation of privilege	Economy of mechanism	Least common mechanism	Reluctance to trust	Never assuming that your secrets are safe	complete mediation	Psychological acceptability	Promoting privacy	Design Principles
Julia H. Allen et. al., 2009	√	√	√	√	√	√	√	√	√	√	√	√	X
Us-Cert	√	√	√	√	√	√	√	√	√	√	√	√	√
Open Web Application Security Project (OWASP)	X	√	X	√	X	√	X	√	X	X	√	X	√
Gary McGraw and John Viega, 2009	√	√	√	√	√	√	√	√	√	√	√	√	√
NIST	√	√	√	√	√	√	√	√	√	√	√	√	√
SANS	√	√	√	√	√	√	√	√	√	√	√	√	√
IBM	√	√	√	√	√	√	√	√	√	√	√	√	√
<u>Saltzer and Schroeder</u>	√	√	X	X	√	√	√	X	X	√	√	X	√

The principles are described as universal guidelines adopted by every developer and project planner, and they have been adopted by practitioners and experts in the security field in order to mitigate risks in the architecture, design, implementation, testing and maintenance phases. They have been gathered from sources shown in Table 1 (Azham et al., 2011).

Then the authors proceed in describing what could be called a review of traditional waterfall security model and compare it to Microsoft SDL (Azham et al., 2011; Techopedia, n.d.).

Next, the authors continue to the key point of the article, introducing the ideas of proposed Security Backlog (SB) and Security Master (SM) that handle security and their integration into the Scrum model. Then authors explain the overall enhanced Scrum model. Figure 3 illustrates the idea that is reproduced from the article (Azham et al., 2011).

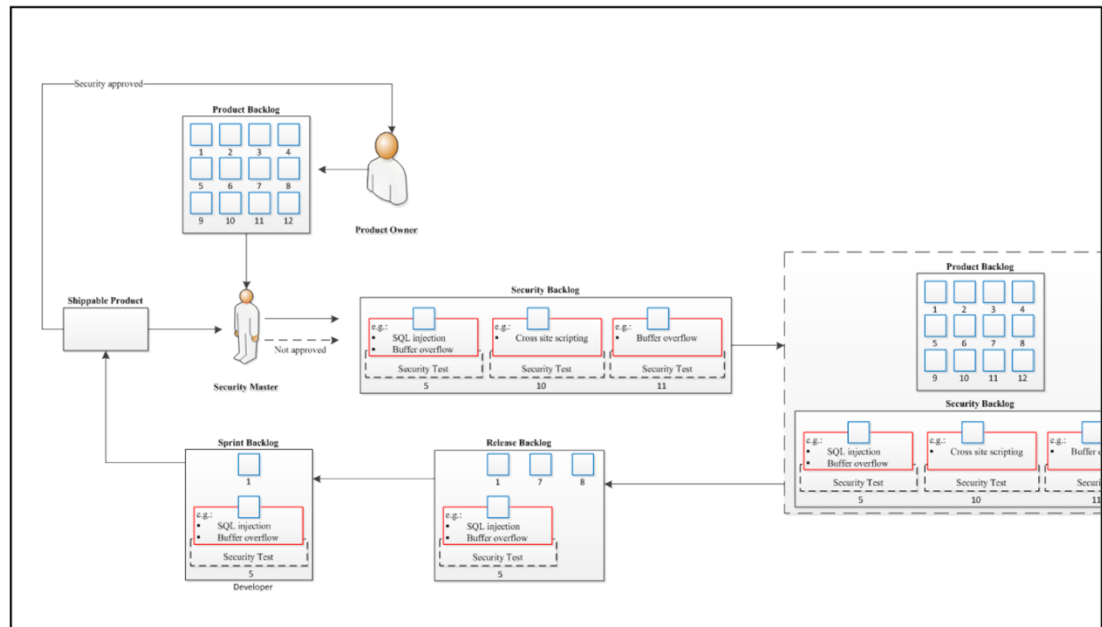


Figure 2. Enhanced Scrum process with additional SB and SM

The requirements are translated into product backlog and go through SB. There, the SM figures out certain features in the product backlog that require security attention and the Security Master marks (dotted in illustrations) the selected features in the security backlog. The SM then creates a document of its activity for use as a reference during the development and testing phases. The marked security concerns are noted in the sprint backlog for the attention of the developers. Lastly, the testing is verified in sprint phases by the security master (Azham et al., 2011).

The last part of the article is given to the analysis of the caused changes including their effect on agility. The results show that agility is improved if the security backlog (SB) is implemented, which the authors see meaning that agility is not negatively affected if the SB is added to the Scrum model. The authors note that in the study the agility of different phases has not been considered. In addition, the purpose of the study has not been not to prove that PB is unsuitable. On the contrary, the

intention had been to discover, after the enhancement of PB, how far Scrum agility was affected overall (ibid.).

2.5.3 Busting a Myth: Review of Agile Security Engineering Methods

The article by Rindell, Hyrynsalmi & Leppänen (2017) describes inherent incompatibility between the traditional non-agile security processes and the new agile methods and the related suspicion which the authors claim still affects the attitude towards agile security. The article presents a literature review of a selected set of agile secure software development methods and a systematic literature method that was used to find the definitive set of secure agile software development methods. A core set of 11 papers had been selected for analysis, and the security activities documented in the methods were extracted (Rindell, Hyrynsalmi, & Leppänen, 2017).

The selected articles cover agile development broadly: from XP to modern Scrum. In the beginning of the article, the authors present a two-stage selection process and then review and analysis process where the security activities described in the articles had been identified. The activities had been evaluated using the principles of Common Criteria, which is the ISO standard to “permit comparability between the results of independent security evaluations. It does so by providing a common set of requirements for the security functions of IT products and systems and for assurance measures applied to them during a security evaluation” (“Evaluation criteria for IT security ISO/IEC 15408:2009,” 2014).

Furthermore, each method had been screened for any empirical evidence provided, as were references to any security, quality or safety standard (Rindell et al., 2017).

An overview of the gathered security activities is reproduced below in Table 2.

Table 2. Comparison of security activities

Phase	Activity	Occurrences	Source(s)
Pre-requirement	Specify operational environment	1	[9]
	Identify global security policy	1	[9]
	Institute security awareness program	3	[9], [24], [2]
	Monitor security practises	1	[9]
	Research and assess security solutions	1	[9]
	Build information labeling scheme	1	[9]
	Project security officer	1	[16]
	Pre-Requirement phase adaptations total	9	
Requirement	Identify user roles and requirements	5	[4], [6], [9], [16], [2]
	Perform security analysis of requirements	5	[4], [6], [9], [16], [2]
	Specify resource-based security properties	2	[6], [16]
	Requirement inspection	1	[4]
	Requirement phase adaptations total	13	
Design	Risk estimation	3	[3], [16], [2]
	Threat modeling	2	[3], [16]
	Document security design assumptions	5	[4], [6], [3], [16], [2]
	Detail misuse cases	7	[11], [4], [5], [6], [24], [3], [16]
	Apply security principles to design	4	[5], [6], [24], [2]
	Specify DB security configuration	0	
	Perform security analysis of system design	3	[4], [6], [2]
	Design UI for security functionality	0	
	Annotate class designs with security properties	1	[16]
	Quality gates	1	[2]
	Design phase adaptations total	26	
Implementation	Coding standards	3	[4], [16], [2]
	Pair programming	2	[24], [16]
	Integrate security analysis into build process	1	[2]
	Implement and elaborate resource policies	0	
	Implement interface contracts	0	
	Address reported security issues	1	[16]
	Implementation phase adaptations total	7	
Testing	Identify and implement security tests	6	[11], [4], [24], [3], [2]
	Perform security function usability testing	1	[4]
	Perform SW security fault injection testing	1	[2]
	Testing phase adaptations total	8	
Release	Manage system security authorization agreement	1	[3]
	Perform source level security reviews	3	[4], [3], [16]
	Verify security attributes of resources	1	[3]
	Manage certification process	1	[3]
	Perform code signing	2	[3], [2]
	Build operational security guide	2	[3], [2]
	Manage security issue disclosure process	3	[11], [3], [16]
	Repository improvement	1	[4]
	Release phase adaptations total	14	
Adjusted total number of security activities	38		

The key finding of the article is that the practice of security engineering appears well adapted to and widely used with agile software development methods (Rindell et al., 2017).

2.5.4 A Systematic Mapping Study on Security in Agile Requirements Engineering

The article by Vegendla, Duc, Gao & Sindre (2018) is a recent study of the publication landscape of approaches that handle security requirements in agile software projects. The paper presents a systematic mapping to outline relevant work and contemporary gaps for future research. In total, the paper covers 21 studies dated from 2005 to 2017 that met the authors' inclusion criteria. The authors found that the approaches typically involve modifying agile methods, introducing new artifacts (e.g., extending the concept of user story to abuser story), or introducing guidelines to handle security issues (Vegendla, Duc, Gao, & Sindre, 2018).

The article presents six key research questions, of which number 3 is the most interesting in connection to this study: How are security requirements handled in each approach? Also, the paper sheds light on the number of available related articles also relevant to this study. Figure 3 below is a reproduction from the article.



Figure 3. Distribution of agile security research, type per year

As can be seen from the reproduced figure above, it appears that before 2014, according to the authors of the paper, there have not been publications of evaluation type of research papers on secure agile development, which is interesting to acknowledge in Digia's secure agile SDCL development.

The article authors also note in the paper summary that in regard to the empirical evaluations, it is evident that there are additional gaps to cover, given that there is

the lack of empirically evaluated studies, which means there is relatively little evidence about the feasibility of the presented approaches (Vegendla et al., 2018).

3 Implementation of the New Digia Secure SDLC

3.1 Implementation Work in Practice

As stated earlier, the actual process development methodology was based on small workshops where the key principles to include in the agile part of secure SDLC were fitted to Scrum. Later, a literature review and analysis with synthesis was conducted to evaluate and test the work done and reveal further opportunities for improvement. The following subchapters explain the actual work and the resulting process in brief. The complete work with details is included in the appendices.

Digia Core Process Model (CPM) development program was divided into parts that were given to different process owners. The general software development process (i.e. the one not having enhanced security features) development was led by another process owner. The general development model focused more on Scaled Agile Framework (SAFe), and as such had a higher level perspective. The secure development process was developed separately and was intended to be used in replacement or enhancement of traditional Scrum. In this matter, the two approaches did not overly overlap, nor contradict each other.

It was evident from the given schedule and resources that Secure SDLC work should focus specifically on agile development, as it was considered to be the most important part, and it was clearly the part lacking most standardization. It was seen that the traditional waterfall model could be dealt either later or by choosing a suitable existing method. In the end, Microsoft SDL (Microsoft Security Development Lifecycle) was chosen to be used in waterfall-style projects. The given time and resources were dedicated to agile and Scrum and enhancing its security and privacy features.

Microsoft SDL is not covered here in detail but briefly put, Microsoft SDL is a software development process based on the spiral model, which has been proposed by Microsoft to help developers to create applications or software while reducing

security issues, resolving security vulnerabilities and even reducing development and maintenance costs. The process is divided into seven phases: training, requirements, design, implementation, verification, release and response (Techopedia, n.d.).

The actual process development was based on small workshops and sessions where the key principles to include in the Digia SDLC were fitted to Scrum methodology. Then there was typically some time to reflect to the proposed ideas before moving forward. The work was done under quite some pressure from other projects and the overall CPM development program schedule.

The end results were documented with examples and training material was prepared for e-learning courses and lectures. All the material was given internally for developers and general review. Additionally, as part of the core process rewrite program, the company quality management reviewed Digia Secure SDLC materials several times to verify e.g. ISO 9001 requirements for the processes to be met as there was an ISO 9001 re-audit just before summer 2018.

It was decided that a literature review and an analysis with a synthesis were to be made to get an academic evaluation of the work done. The sources of academic documents have already been described in Chapters 2.2 and 2.5 above.

3.2 Changes to Scrum

The resulting Digia Secure SDLC, or as it is also called, Digia Secure & Privacy software development process, was not a very heavy one and unnecessary complexity was avoided where possible.

The new process model introduced a few additional steps to agile development and brought along a set of additional (some mandatory) features to any development process. The additional features are in a key role to guide secure development work and revise process in the future. They are explained later in more detail further in the thesis.

It was decided that using the S&P Software Development Process is mandatory only in projects that meet a set of predefined criteria (Digia SDLC decision criteria). However, the application of the process is strongly encouraged in any situation

where security and privacy-related concerns require more attention due to the domain of the application or other requirements set by the customers or end users.

3.2.1 The Decision Criteria

The decision criteria was designed to be in a table form and easy for anyone to use. The idea was to sum up the points of positive answers while evaluating the project in question against the criteria written on each row. The level was then decided after which using S&P Software Development Process would be mandatory. The idea behind this was that the adoption of the process and the results of the criteria evaluations can be monitored and the level then adjusted that sets the process as mandatory later in time.

Table 3 below shows the idea of the criteria table. The initial level and the actual points values of each criterion in the table are sanitized or exemplary due to security considerations of Digia. Of course, if this method were to be applied in another context, the actual level and points should be fitted to that particular context. Nevertheless, the table below gives the general idea, and an exemplary level of points could be e.g. 30, after which using the enhanced process would be mandatory. It should also be noted that the table below gives only a subset of the criteria due to the same reasons.

Table 3. An exemplary table of decision criteria for using Secure SDLC

Area	Criteria	Note	Points
Security			
	Project is classified by government to ST IV	SHOULD	15
	Project is classified by government to ST III	MUST	30
	Project moves data cross-border	SHOULD	X
	End product moves data cross-border	SHOULD	X
	...		
Privacy			
	End product handles genetic data	MUST	X
	End product handles biometric data	MUST	X
	End product handles health data	MUST	X
	End product handles data related to children	...	X
	End product does automatic profiling based on personal data		X
	...		
General risk			

	Considerable reputation risk		X
	Typical reputation risk		X
	Considerable contract risk (sanctions, fines)		X
	Typical contract risk		X
	Considerable end product or service attractiveness to misuse / hacking		X
	...		

3.2.2 Process Flow

The following figure (Figure 1Figure 4) shows the process structure of the finished documentation of the Digia Secure SDLC. Each step is then explained below in the manner as if it were described to a person studying the process. In some points, the key reasoning behind the choices has been added.



Figure 4. Digia Secure SDLC general process flow

Process level inputs and outputs are described formally in the process materials in the Appendices of this document.

3.2.3 Key Decisions before Entering Secure Agile Process

Before entering the secure agile process in the model, some key decisions have to be made. The initial project security requirements have to be analysed and a decision has to be made to use either standard or secure process. Another key decision is whether the project model will be rigid, semi-rigid or agile development. Rigid means here traditional waterfall or plan driven project model. Only the secure agile process is within the scope of this document and it will be discussed below.

3.2.4 Secure Agile Process Initiation

Secure agile process initiation starts by identifying the needed security competences and a suitable agile team for the project. It should be noted that all changes to the team set-up will later have an impact on the team performance. A key person to secure the agile process is a Product Owner (PO). He or she has to be identified early in the process.

At the end of the secure agile process initiation, there should be at least one cross-functional agile team selected, project's or product's initial vision statement, and the first draft of Definition of Done written.

The key element to bring security in Digia secure agile SDLC process was that during the whole process, the work must follow the guiding Security and Privacy Design principles that were introduced as new elements. These form the foundation of the secure practices in a project.

It was seen that further constraints on the development were needed to guarantee the security and privacy claims. These claims were included but were not limited to such as "The solutions guarantees the privacy of people whose information is handled" or "The web-interface is secured against common intruders". Typically, such claims are backed up by showing the measures that have been taken to achieve those results and so, the security-driven project needs to produce documentation to this effect.

3.2.5 Product Owner Readiness Evaluation

In Digia secure agile SDLC, the Product Owner needs to be security and privacy savvy. In the process stage "PO Readiness Evaluation" the organization needs to assure that the person taking the ownership responsibility of the project has skills and knowledge to make all the key decisions during development.

The evaluation can be as simple as seeing that the person has the formal proficiency, s/he possesses a credible education in the field or a certificate from a recognized authority, or it can be had by having the prospective owner carefully interviewed and vetted by the company's security experts.

It was seen that in the end this is likely to be an executive decision, however, it should be based on such an evaluation that the executive in case can be ascertained of the rightness of his/her decision.

3.2.6 PO Security Training

In the evaluation, it can turn out that the considered PO does not have the necessary security knowledge and skills, and still that particular person will get selected to the role from one reason or another. Then the PO should undergo a training providing him/her with the necessary skills. Due to the importance these skills have in the successful performance of the role, the training should contain an assessment that verifies that the person has obtained the required knowledge and skill set.

It was decided that the security trainings will be organized by at least Digia Academy; also, projects are required to discuss external trainings.

3.2.7 Adding a Support Person

Alternatively, or in addition to training, the selected PO, a security specialist or specialists can be added to the Scrum team to support the PO in security matters. This person would then complement the areas where the PO is found lacking skills or knowledge.

3.2.8 Backlog Security and Privacy Refinement

During this stage the known key Scrum epics and stories will be refined with the security angle in mind. Most agile practices maintain that only the readily known stories near the top of the backlog are defined in the fine-grained level, as the ones waiting for further work down the line remain in an only generally defined state.

At this time, the less defined stories should be looked into in passing with the question "Does the story at this state and stage have a security impact". If the answer is no, it should not be defined further as that would create the undesired upfront design work that the agile process seeks to avoid. The story should be looked into again once it has been refined to such level that its security implications can be analyzed.

The known and understood stories, however, should be looked into keenly and evaluated as for their security and privacy impacts.

At the first time the process enters this stage, the product's initial Definition of Done should also be written. The agile team should, with their PO in the lead, see how they should incorporate the prescribed criteria from the tools section of this process.

With this done, the PO should then see if the project would benefit from having the optional security stories from the tools section added to the backlog as well. With these additions and refinements of the backlog, the security should be addressed adequately for the starting iteration.

Security refinement can be returned to from the product iterations or failed final audit and after a release; yet, in an ideal world the PO will include this work to their normal product backlog maintenance.

A Short Recap of Agile Requirements

In agile development, the requirements are generally not requirements, but stories and constraints discussed by the whole team and negotiated between the developers and PO. This is because a requirement would be immutable. In a good case this means clearly and unambiguously defined desire of a customer that a provider commits to deliver as specified. Unfortunately, this is seldom so; typically, requirements can vary from lists of hundreds of non-connected requirement clauses to not having any real requirements at all.

The lack of certainty is the typical condition in software development and acceptance of that situation is what the agile-movement seeks to address and cope with.

Because very little can be known in the beginning of a project, no decisions are made with too little understanding but instead, the vague ideas and needs get expressed as stories of user interactions. These are described and negotiated by and between the agile teams and their product owners.

The classic form of the story is "As a <user role> I can <do something relevant> in order to <gain something that has actual business value>". Constraints can be architectural quality attributes or as they are more commonly called non-functional requirements such as modifiability, configurability or similar; or procedural

requirements, such as a requirement that all stories to be considered done need to have a functional automated test written for them as well as the core functionality in the system.

The procedural requirements are either in the individual stories' acceptance criteria or in the project's common Definition of Done. The Definition of Done is by best practices a shared artefact for all stories of given kind; however, it can be recapitulated at individual story items every single time if needed.

3.2.9 Security Requirements in Agile Development

Security issues can be brought to the agile team in two ways. Firstly, they can be introduced to the backlog as stories especially in such cases where they have actual interaction. Secondly, they can be brought in as constraints and acceptance criteria either on the project or story item level.

In either case, it is the responsibility of the PO to make certain that the security and privacy issues are addressed for each backlog item. The POs are strongly encouraged to use security specialists if they are uncertain, how the security issues should be handled.

The developers, especially the developers specializing in security-enhanced development work, are equally encouraged to present the PO with questions and suggestions that help the PO to make the proper additions and adjustments to the acceptance criteria and backlog in general to make the system under construction secure.

3.2.10 Privacy Requirements in Agile Development

It was seen that the key issue in privacy and software development is recognizing the personal information, the impact that its compromise would have on the data subjects and the controls used against such compromises.

EU data protection legislation (GDPR) requires Privacy by Default and Privacy by Design principles to be adhered in addition to a number of so called Data Subject Rights. Privacy by design was also adopted as a guiding principle in Secure Agile SDCL

process. During Digia GDPR program 2017 - 2018, a new privacy policy was written, which also gives a strong direction for privacy related work.

In general, personal data information flows, storing and processing including the full content must be known and analyzed to achieve any privacy in software. This is called initial personal information (PI) analysis.

After this initial PI data analysis, PI data is then analyzed with the default tool for addressing privacy impact issues. A default Privacy Impact Analysis and Data Protection Impact Analysis (PIA / DPIA) tool for Digia Secure SDLC was constructed during the process development.

After PIA / DPIA processes, Digia Secure SDLC was designed to require the modelling of privacy related security controls that the PI data requires. This includes modelling the flow of Personal Information through the system and the protections that have been planned for this information and then analyzing if all is as it ought to be or if something should be changed.

General requirements such as "Systems handling PI data must have PI register handling declaration (käsitelyseloste)" were given as requirements for the process. It should be noted that this does not contradict with agile Scrum, as the requirement is targeted for the process, not the end product.

3.2.11 Iterative Progress

The Digia Secure SDLC process was defined as agnostic to the methods of agile development as possible. The iterative progress in the development can so be done according to almost any agile framework.

Due to the development process's agnostic approach, this process step was intentionally left vague in description; nevertheless, a set of tools was developed. It should be stressed here that these tools were seen as a key method for incorporating security and privacy elements to the agile software development.

The tools were written in the format of guiding principles and prewritten epics, stories and acceptance criteria constraints, of which the latter can be applied to

either individual stories or to the Definition of Done shared by all stories of the project.

3.2.12 Security Audit

It was decided that before every release the developed solution should be inspected and tested in order to give assurance that it fulfils the promises Digia are making regarding the security and privacy it provides. This includes fulfilling the contract with the client.

Depending on the targeted security level, the release cycle and the practices the team adopts in its iterations, the security audit step can be a full-blown security audit arranged by Digia or an external 3rd party organization, or it can be a demonstration of the required analysis and tests done as a part of a sprint review.

3.2.13 Release

To release a product increment is to make it available for its intended audience. The core act can be deploying the software to servers or just making an installation package available for the customer. The auxiliary actions would be publishing the release notes and the like, however, they are covered in Product process and are not within the scope here.

Due to the nature of the secure development process, it was decided that it is mandatory e.g. to have a separate section in the release notes for the security fixes that the release addresses.

3.3 Primary Support Tools

The tools section of Digia Secure SDLC process was designed to help the POs to adjust their backlogs to produce security and privacy retaining systems. A set of stories and acceptance criteria, mandatory for all projects wishing to adhere to the process, was designed as part of the process.

The primary tools were written as principles, mandatory and recommended stories and acceptance criteria constraints, example stories, related requirements and best

practices. Beyond these there it was seen that decisions support tools and security protocol standards would be necessary additions.

As the principles are seen in a key role, they are reproduced below. The pattern library is included in the Appendices section of this thesis.

3.4 Key Security and Privacy Design Principles

3.4.1 Keep It Simple

Whatever the solution is, it should be kept as simple as possible. Truth of the matter is that the more complexity is added to a given solution or system, the more difficult it is made to properly understand and use it. Thus, it holds that if security of some design needs to be increased, its complexity and the ambiguity related to it needs to be decreased.

3.4.2 Defence in Depth

Defence in depth means that relying to only one layer of defence should be avoided and instead defences ought to be stacked. The classic example of the principle is that the data lying in the company servers should be protected by a firewall; however, in case it fails it would be nice if the data in question also happened to be encrypted. In this example, the encryption is the second tier of defence. The principle is further examined in the pattern examples.

3.4.3 Privacy by Design

Privacy by design is an information management principle that ought to be followed in designing systems produced within the process described here in order to ascertain that most likely a system is being made where the privacy requirements are adequately addressed. The principle can be further subdivided into seven foundational principles that provide more specific guidance for the work.

At Digia, this foundational rule should and shall be considered in conjunction to the European Union's General Data Protection Regulation (GDPR).

This principle is fully reproduced from Digia's intranet in Appendices.

3.5 Rollout of the Process

The rollout of the process started in early 2018. The process materials were reviewed several times as already described and then published as a part of Digia's CPM program on the company intranet.

Presentation material was prepared and several large scale lectures and a number of smaller ones were given at Digia. Then the material was prepared for an electronic course to Digia's e-learning environment and it is available to all employees at Digia. There are plans to have additional in-house lectures during winter 2019.

The next phase is to publish this work to open public and get feedback from other companies and possibly academic world. The current draft name for published version is Digia Open Secure Agile SDLC.

4 Evaluating the Resulting Process

The main research problem was: how to change or enhance agile Scrum so that it produces secure software? It was then proceeded to form one's own view of best practice and suitable controls, and then in several workshops these principles were fitted to Scrum. The original work method assumes, perhaps too strongly, that correct choices were made in selecting the principles. Therefore, it was only reasonable to evaluate the made choices against work done by others. The latter became one of the secondary research problems.

4.1 Original Security Principles

It is reasonable to evaluate if the author managed to incorporate the original principles to the company's secure agile SDLC; yet, first the principles themselves deserve some critique.

The original principles were based on best practice and years of experience. They were mixed in granularity and some of them were not even security principles, instead, they were good practices and, in some cases, needed support tools.

However, this does not imply that they would not help in producing software that considers security and privacy properly. The initial approach to forming the principles could have been more scientific in the first place. However, as progress was made, the need for more clearly defined principles came clear.

Table 4 summarizes the post analysis regarding the principles, their evaluation and possible success rate.

Table 4. A Summary of post analysis

Principle	Evaluation	Success
2.4.1 Any Secure and Privacy SDLC Should be Documented	<ul style="list-style-type: none"> - Natural requirement - S&P design and design review were included - Security architecture is promoted - PO/BO are required and their required skill level is emphasized - Additional security resource need is evaluated within process - The support tools give links to STRIDE documentation, however, its training is lacking - Pattern library is pretty ok and growing - Policies and standards are available in intranet and situation is getting better 	<p>Success</p> <p>Needs improvement, needs simplification and structurization</p>
2.4.2 Standards Should be Used	<ul style="list-style-type: none"> - We are basing current S&P processes to holistic view on security and privacy, not excluding ISO27001 but not fully compatible - This is reflected in our development projects, Scrum is not yet identically deployed, and not all use Scrum: Kanban etc. are also used. - OWASP is included in our pattern library 	Partial success
2.4.3 Secure SDLC is not static, it needs directing	<ul style="list-style-type: none"> - Our secure SDLC is currently constantly reviewed and developed - Policies framework is being updated 	Success
2.4.4 Risks should be evaluated	<ul style="list-style-type: none"> - Process supports this - Simple tools needed to promote this more 	Partial success
2.4.5 Separate features, stories and defects from security and privacy items	<ul style="list-style-type: none"> - Not addressed by the process, but the tools - This is a common policy, but there is still a lot of legacy burden 	Partial success
2.4.6 Secure coding training should be available	<ul style="list-style-type: none"> - Not addressed by the process, but the tools - Training is still not reaching all it should 	<p>Success</p> <p>Needs further work</p>

	- Training is available, but will be emphasized in 2019	
2.4.7 Source Control System with Access Control should be used	- Not addressed by the process, but the tools - This is de facto in general - Old source control software that is not capable in enforcing mandatory policy of code review still in use in some parts	Success Needs further work
2.4.8 Releases Should be Versioned and Controlled	- Not addressed by the process, but the tools - Basically de facto	Success Needs further work
2.4.9 Separation of Environments Should be in Place	- Not addressed by the process, but the policies	Success Needs further work
2.4.10 Separation of Duties	- The process principles should add this - This was not included in the process properly	Not success Needs further work
2.4.11 Set of Standard Tools	- In place and improving	Success Needs further work
2.4.12 There Should be a Rehearsed Incident Management Process	- Not addressed by the secure SDLC process, but the related another process - This was renewed also during 2018	Success
2.4.13 Improving the Security and Privacy Culture	- Not addressed by the process, but the tools - In emphasis 2019	Success Needs further work

Out of 13 original design principles, nine are successes, three partial successes and one could be considered a failure to implement at all. The root cause for the latter points to the rather irregular workshop methods. The tracing of original requirements was not done well enough. In this light, Digia's secure agile SDLC does cover original ideas; however, as can be seen in the evaluation column, quite many, if not most of the original principles are not actual security principles. The next subchapter provides some analysis and synthesis regarding the security principles.

4.2 Actual Guiding Principles in the Process

As part of the secure agile SDCL, only three security and privacy design principles were defined as follows:

1. "Keep it simple"
2. Defence in depth
3. Privacy by design

In addition to these, mandatory or strongly promoted story patterns were defined:

1. MANDATORY: As a Business Owner I want the product to have a version management strategy that clearly defines how many versions get security fixes in order to control costs and manage the expectations of my customers.
2. MANDATORY: As a CSO I want the product issue management to differentiate security issues from regular issues within the issue management system.
3. MANDATORY: As a CSO I want the development, testing (staging), and production environments to be separated.
4. MANDATORY: As a CSO I want Privacy by Design and Privacy by Default implemented in every product / service.
5. MANDATORY: As a Product Owner I want to get an attack surface analysis for the solution, at least 1 per Epic or bigger
6. SHOULD / STRONGLY RECOMMENDED: As a Product Owner I want to have penetration test results for the solution in order to have guarantees for the security levels I am promising, (once / release based on documented risk evaluation / mandatory on government high security projects)
7. SHOULD / STRONGLY RECOMMENDED in some cases MANDATORY: As a Product Owner I want to have fuzz-tests set up and passing for all interfaces of my system, (fuzzing should be done at least once / version of interface of the system based on documented risk evaluation / mandatory on government high security projects)
8. MANDATORY: As a Product Owner I want to see known and listed security issues fixed in a hardening themed iteration(s) / sprint(s) in order to raise the product to the desired level, (once / release cycle)

In addition to these, mandatory or strongly promoted acceptance criteria were defined:

1. DoD + attack surface analysis for the new interface
2. DoD + traceability criteria
3. All code is reviewed by another developer before merging to the master branch, per story
4. Source code implementing an issue item (task, story etc.) is marked (tagged) with issue identifier.
5. All dependencies brought in by dependency management systems are set to a particular reviewed version, per story
6. The ways this story affects the attack surface of the solution have been evaluated, per story

Finally, OWASP TOP10 standard was written in the form of agile stories to complement the principles, the story patterns and the acceptance criteria.

Azham et al. (2011) describe the following describe security principles:

1. Least privilege
2. Failing securely
3. Securing the weakest link
4. Defence in depth
5. Separation of privilege
6. Economy of mechanism
7. Reluctance to trust
8. Never assuming that your secrets are safe
9. Complete mediation
10. Psychological acceptability
11. Promoting privacy
12. Design principles

It can be seen at least the first five principles and principles number 11 – 12 have been covered. Partly covered are the principles number 7 and 8. In this light, it can be seen that proper security and privacy promoting features or controls have been addressed; however, obviously the selected security principles need to be reviewed and the secure agile SDCL reviewed in future work.

If comparing these principles to those found in the literature, it can be easily found that typically privacy issues are not addressed at all together with security. Security and privacy can be seen closely connected, and as such, one might be tempted to give oneself some credit here.

4.2.1 Additional Reflection to Literature

Comparison to Development of Secure Software with Scrum

The article Development of Secure Software with Scrum (Pohl & Hof, 2015) does not explain exactly what is an appropriate level of security, or how it should be defined. Compared to this study, the appropriate security level was left to be defined in a risk analysis and with guidance from e.g. GDPR related requirements.

All in all, Pohl & Hof (2015) have a very similar approach to the one described in Chapter 3, as the method proposes having additional features or enhancements to standard Scrum. The approach in the thesis does not directly or specifically mention Daily Scrum meeting contents or tasks.

Pohl & Hof (2015) propose DoD changes, as does this thesis.

The ranking of user stories to loss values is not something that was chosen to do. The idea seems promising and when working, it could actually contribute to security a great deal. However, it can be seen that a typical Scrum team and possibly a single PO representing the customer does not have the knowledge to perform this kind of analysis in the real world. The idea proposes performing analysis comparable to business impact analysis (BIA), which typically needs much more involvement from business management. This kind of involvement in typical SDLC is rarely if ever seen.

Security tagging in backlog items was chosen to be used as did Secure Scrum, which helps to get an overview of the security related stories and keeps the necessary focus on security and privacy.

Verification was covered; however, such strict principles were not set here. This kind of strict approach could be good, however, it could be hard to implement in a large scale. The agile agnostic angle must be remembered and it should also be remembered that the secure SDLC has to scale to SAFe level if needed.

Pohl & Hof (2015) suggest the use of external consultants whereas the thesis suggests using them as part of the Scrum integrally.

Comparison to Security backlog in Scrum security practices

Typical agile requirements, flexibility, speed, leanness, learning and responsiveness were desired to be maintained in the same way Azham, Ghani & Ithnin (2011) discuss. Actually, a great amount of work was done to keep Scrum in line with the original Agile manifesto (Beck et al., 2001).

The security principles analyzed in the article of Azham, Ghani & Ithnin (2011) have already been analyzed above.

The design choice for traditional plan driven or waterfall models was actually to use Microsoft SDL with them.

The key point with the paper of Azham et al. (2011) was to have a Security Backlog. Such separate backlog was not used but tagging within the main backlog was used. In this sense, the thesis is in line with the ideas of others.

4.2.2 Reflection to Original Design Problems

The main research problem was: how to change or enhance agile Scrum so that it produces secure software? It can be concluded that it was managed to design an agile SDLC that considers security and privacy requirements. The thesis ended up with quite similar solutions as others have. However, where the thesis lacked in academic approach and perhaps the work was based more on practice in the first place, it managed to keep Scrum intact and still have some good features that the fellow researchers had not come up with.

The secondary research problems were:

1. What is needed in a software process in general for it to produce secure end results?
2. Do the same changes or enhancements enable the process to produce end results that have required privacy level and controls as well?
3. How have others approached this problem?
4. How should we contribute this research to the community and provide value to general public?

The process described in the previous chapters and appendices and the referenced science papers hold the main elements already that answer the first question with acceptable reasoning. It is also apparent that as privacy cannot exist without security, to have a software process that produces end results with a predefined privacy level, secure development process just has to incorporate the needed features. It is possible.

The third question is answered in brief in this work and could also be considered as achieved.

Last but not least, the answer to the fourth question is basically to publish this thesis openly through this work and additionally on e.g. Digia web pages as what could be called "Digia Open Secure Agile SDLC". It is seen that it is worth trying to contribute to the public, as one can in return get feedback and comments on how to improve this work.

5 Conclusions and Further Ideas

The main result of this work is a live process that is currently being trained and being adopted in a large scale in a 1,100-employee software company.

Based on our analysis, building secure Scrum has been even hotter topic in the academic world than what we thought. However, it is surprising that no real standardization still exists.

Our approach is based on simplicity and keeping agile Scrum intact and truthful to the original Agile Manifesto (Beck et al., 2001) As a conclusion, the literature review would most likely have impacted the end results if it had been conducted earlier in time. Our approach produced original if similar ideas as others have found useful.

It is evident that we still need to continue developing our secure SDCL process and continue developing tools and guidance for it to be even better in the future.

References

- Azham, Z., Ghani, I., & Ithnin, N. 2011. Security backlog in scrum security practices. 2011 5th Malaysian Conference in Software Engineering, MySEC 2011, 414-417. Accessed October 21, 2018. Retrieved from <https://doi.org/10.1109/MySEC.2011.6140708>
- Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Grenning, M. F. J., ... Thomas, D. 2001. Manifesto for Agile Software Development. Accessed July 6, 2018. Retrieved from <http://agilemanifesto.org>
- BIA 2018, Page on Wikipedia. Business Impact Analysis, Accessed October 20, 2018. Retrieved from https://en.wikipedia.org/wiki/Business_continuity_planning
- Bug bounty program 2018, Page on Wikipedia. Bug bounty program, Accessed October 20, 2018. Retrieved from https://en.wikipedia.org/wiki/Bug_bounty_program
- Evaluation criteria for IT security ISO/IEC 15408:2009. 2014. ISO/IEC. Accessed July 6, 2018. Retrieved from <https://online.sfs.fi/fi/index.html.stx>
- Ge, X., Paige, R. F., Polack, F., & Brooke, P. 2007. Extreme Programming Security Practices. In G. Concas, E. Damiani, M. Scotto, & G. Succi (Eds.), *Agile Processes in Software Engineering and Extreme Programming*, 226–230. Berlin: Springer Berlin Heidelberg.
- Geib, J., Casey, L., & Santos, R. 2017. Microsoft Threat Modeling Tool. Accessed November 14, 2018, Retrieved from <https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool>
- Johannesson, P., & Perjons, E. 2014. An introduction to design science. An Introduction to Design Science. Accessed August 10, 2018, Retrieved from <https://doi.org/10.1007/978-3-319-10632-8>
- Leffingwell, D., Jemilo, D., Zamora, M., O'Neill, C., & Yakuma, A. 2014. Scaled Agile Framework (SAFe). Accessed November 14, 2018. Retrieved from <https://www.scaledagileframework.com>
- Mohamood, H. 2017. Application Threat Modeling using DREAD and STRIDE. Accessed June 5, 2018, Retrieved from <https://haiderm.com/application-threat-modeling-using-dread-and-stride/>
- Mountain Goat Software. n.d. Sprint Review Meeting. Accessed November 23, 2018, Retrieved from <https://www.mountangoatsoftware.com/agile/scrum/meetings/sprint-review-meeting>
- Niemelä, J., Rantonen, M., & Huotari, J. 2015. Statistical Analysis Of Malware Defence Methods, Accessed April 17, 2016. Retrieved from the author.
- Pohl, C., & Hof, H.-J. 2015. Secure Scrum: Development of Secure Software with Scrum. Accessed May 19, 2016. Retrieved from <http://arxiv.org/abs/1507.02992>

- Rindell, K., Hyrynsalmi, S., & Leppänen, V. 2017. Busting a Myth. Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17, 1–10. Accessed June 5, 2018. Retrieved from <https://doi.org/10.1145/3098954.3103170>
- Scaled agile. 2018. Scaled agile framework - safe for lean enterprises. Accessed November 14, 2018. Retrieved from <https://doi.org/10.1017/S1431927615014221>
- Scaled Agile Inc. 2017. SAFe[®] 4.5 Introduction Overview of the Scaled Agile Framework[®] for Lean Enterprises. A Scaled Agile, Inc. White Paper. Accessed November 14, 2018. Retrieved from <https://www.scaledagileframework.com>
- Schwaber, K., & Sutherland, J. 2013. The Scrum Guide. Scrum.Org and Scrum Inc. Accessed July 17, 2018. Retrieved from <https://doi.org/10.1053/j.jrn.2009.08.012>
- Separation of Duties 2018, Page on Wikipedia. Business Impact Analysis, Accessed October 20, 2018. Retrieved from https://en.wikipedia.org/wiki/Separation_of_duties
- Techopedia. n.d. Microsoft Security Development Lifecycle (Microsoft SDL). Accessed November 16, 2018. Retrieved from <https://www.techopedia.com/definition/23582/microsoft-security-development-lifecycle-microsoft-sdl>
- Vegendla, A., Duc, A. N., Gao, S., & Sindre, G. 2018. A Systematic Mapping Study on Requirements Engineering in Software Ecosystems. *Journal of Information Technology Research*, 11(1), 49–69. Accessed November 16, 2018. Retrieved from <https://doi.org/10.4018/JITR.2018010104>
- What is Scrum? 2018. Accessed July 13, 2018, Retrieved from <https://www.scrum.org/resources/what-is-scrum>
- Wäyrynen, J., Bodén, M., & Boström, G. 2004. Security Engineering and eXtreme Programming: An Impossible Marriage? Accessed November 16, 2018. Retrieved from https://doi.org/https://doi.org/10.1007/978-3-540-27777-4_12

Appendices

All appendices are added directly to published pdf of this thesis due to printing issues. For this reason, the formatting differs from the thesis. Templates, other attachments and hyperlinks have been sanitized from the appendices.

Appendix 1. Digia High Security & Privacy Agile Software Development (Digia Secure SDLC) – a direct reproduction from Digia’s intranet version.

Appendix 2. Digia Secure SDLC: Security and Privacy Design Guiding Principles

Appendix 3. Digia Secure SDLC: Pattern Library of Mandatory and Generic Patterns

Appendix 1: Digia High Security & Privacy Agile Software Development (Digia Secure SDLC)

Owner	Jylhä Mikko
Purpose	To enhance an agile software development process model so that it is adequate for authoring software with high security and privacy needs. As a result, the process produces a product or a service that has acceptable and significantly lower risk levels in regard to security and privacy.
Customers	Digia employees, Digia Subcontractors, Digia partners, Digia Customers (limited to certain roles within the process), possibly other interested parties e.g. auditing bodies in connection with Digia.
Start	Continuous process

- 1 Introduction
- 2 Modification and variance guide
- 3 Process overview
 - 3.1 Process inputs and outputs
 - 3.1.1 Templates
- 4 Decision: Rigid, semi-rigid or agile development
 - 4.1 A word of caution
 - 4.2 The choice
- 5 Decision: normal or enhanced security and privacy
 - 5.1 Decision criteria
- 6 Informing CSO office of process adoption
- 7 Software development in rigid project setting and enhanced security & privacy context
- 8 Software development in semi-rigid project setting and enhanced security & privacy context
- 9 Software development in agile project setting enhanced security & privacy context
 - 9.1 Terms Used
 - 9.2 Secure development process steps
 - 9.2.1 Secure Agile Process Initiation
 - 9.2.1.1 Input
 - 9.2.1.2 Output
 - 9.2.2 Evaluation: Product Owner and Release Management personnel Readiness
 - 9.2.2.1 Input
 - 9.2.2.2 Output
 - 9.2.3 PO Security training
 - 9.2.4 Backlog security

Introduction

Secure & Privacy software development process is not a very heavy one. Unnecessary complexity has been avoided where at all possible.

The process model introduces a few additional steps to agile development, and brings along a set of additional (some mandatory) features to any development process.

Using the S&P Software Development Process is mandatory in Digia based on a set of predefined criteria (Digia SDLC decision criteria), but its application is strongly encouraged in any situation where security and privacy related concerns require more attention due to the domain of the application or other requirements by the customers or end users.

This page and its sub-pages exist for development of this core process.

Modification and variance guide

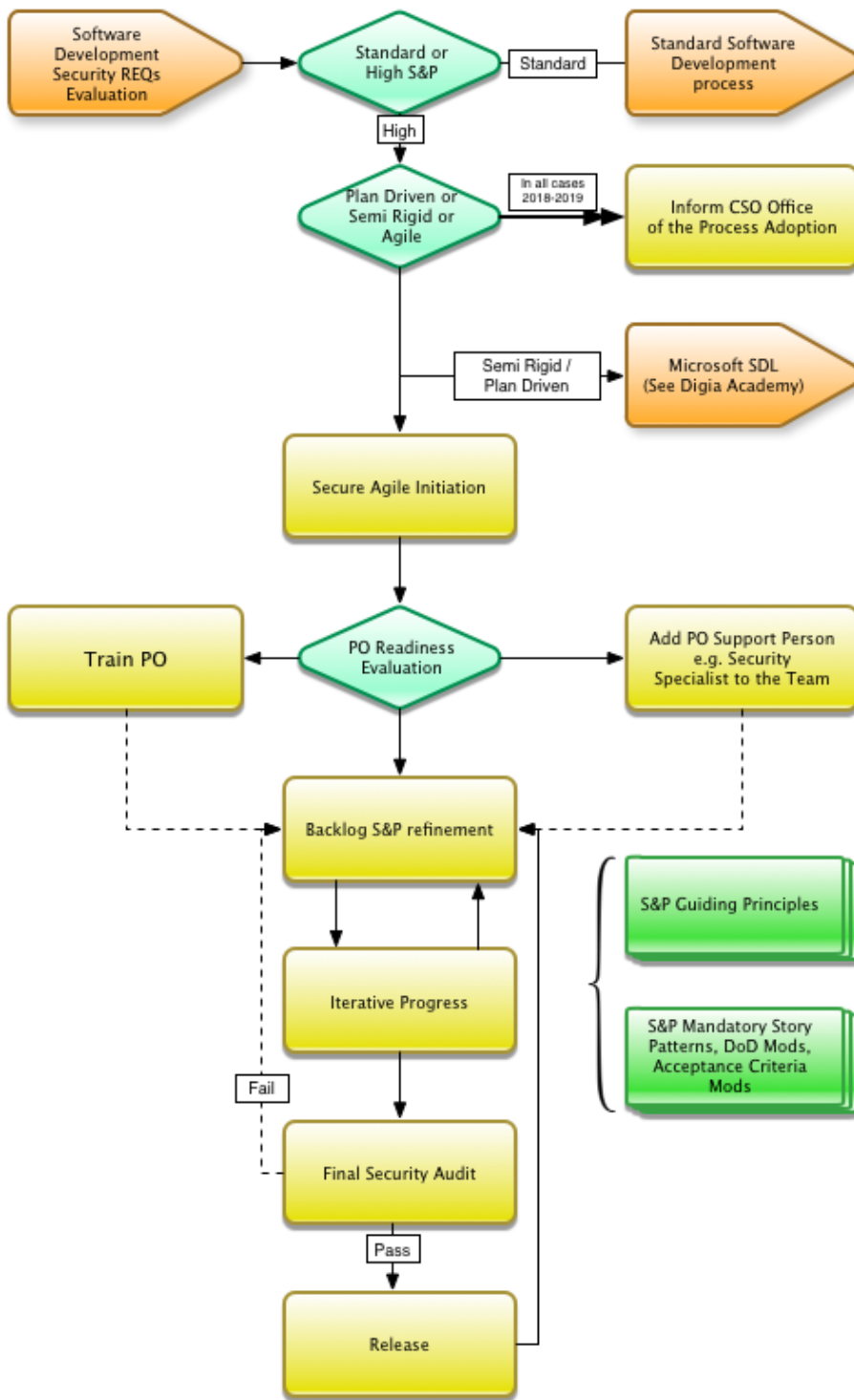
This process or process enhancement exists already in two versions. The agile version and as the other version that is to be used when using a plan driven or semi-rigid model of development.

If there is a need to adapt the process for a given Delivery Area or Delivery Group that can be considered. However, the changes, their effects and the reasoning has to be discussed with the CSO or CSO Office and approved by the CSO office.

The process was designed to be as versatile as possible. The normal variance such as adaptation to a given Scrum, ScrumXP, Lean/ KanBan, SAFe or LeSS implementation should be well within the tolerances of the process.

One should always begin by learning the process either by just going through these pages or by kick-starting one's progress by one of the instruction sets available (e.g. in academy.digia.com) and only after familiarising oneself with the content here. Should you still after that require help in taking this process into use, please contact your BU's security champion and ask for help.

Process overview



- and privacy refinement
- 9.2.4.1 A quick recap of agile requirements
- 9.2.4.2 Security requirements in agile development
- 9.2.4.3 The privacy requirements in agile development
- 9.2.4.4 Input
- 9.2.4.5 Output
- 9.2.5 Iterative progress
- 9.2.6 Final Security Audit
 - 9.2.6.1 Input
 - 9.2.6.2 Output
- 9.2.7 Release
 - 9.2.7.1 Input

Process inputs and outputs

Initial information	Process outcomes
Product vision, at least as a suggestion	A functional software system
Product Security and Privacy considerations	A version controlled repository of the system's source code

Feasibility study or similar	Documentation of the software system's architecture
Staffing needs suggested by feasibility study and product / project scope	Documentation of the final security audit outcome
Preliminary PIA/DPIA stages for the business process identified so far	Development log in the product issue management system (JIRA history)
Or	
Initial information	Process outcomes
Project Requirements <ul style="list-style-type: none"> • Including the Security and Privacy Requirements • Documentation requirements • Process requirements for planning etc. • Preliminary PIA/DPIA stages for the business process identified so far 	A functional software system developed in a secure manner
	Whatever documentation was agreed with the customer or deemed necessary by the plan driven project model
	MS SDL security documents

- 9.2. 7.2 Output
- 9.3 Tools
- 9.4 Process verification
- 9.5 Process quality measurement and metrics
- 9.6 Quality goals for the process
- 10 Mapping to the Software Development Process
- 11 References

Roles and responsibilities

Role	Responsibilities
Product Owner (PO)	Scrum PO responsibilities as explained in terms below including but not limited to security and privacy.
DPO	Monitoring and support
CSO Office	Monitoring and support
Legal	Supports in e.g. Record of Processing
Security & Privacy Consultants	Security Consultants can be used as support for PO.
DA Security & Privacy Champion	1st level of support for DA projects in S&P issues. Each DA has one or more S&P Champions to support projects and secure SDLC of their business.
Release Management Team (RMT)	See terms.
Program or Product Security & Privacy Champion	In large programs or projects with high security requirements, program or product level S&P champion role should be appointed to oversee the program/project and work in co-operation with POs (in case of several).

Also see roles and responsibilities in general [Software Development Process](#). Secure SDLC builds on that process and assumes SAFE principles are followed from there.

Templates

File

Modified ▲

Decision: Rigid, semi-rigid or agile development

A word of caution

Security and Privacy (later: S&P) related concerns are addressed differently in different development processes. In practice this means taking S&P into account in different phases of the development and/or involving different mandatory or voluntary activities while following the process. As a result, **the first the first decision that needs to be made is the one concerning the way that the development is to be carried out.** After making the initial choice (a decision) of the development process being used, one should follow its principles, or make a new decision to shift to another, and NOT mix different models on the fly.

The choice

Traditional Plan Driven a.k.a. Waterfall model accepts a set of well defined requirements for producing security and privacy cognisant solution and based on the industry practices we can say that most of them are quite well known to us. Our initial reference point in plan driven approach is the [Microsoft Secure Development Lifecycle](#). *Note: In the further versions of this process we shall address those requirements in more detail.*

We generally advice all personnel to choose either a purely plan driven or truly agile approach, but acknowledge that there are situations where the customer/project/service is for example in transition towards the agile development practices, but still bound some constraints keeping them from full agility. In these situations we recommend including regular security checkpoints, as much of the security related stories and constraints from our agile guidance as possible and allocating time and work to address the findings. Special care in sales needs to be put on recognising, alleviating and owning the financial risks of such projects. In practice this means that the sales process MUST take into account the cost of security and privacy related development and testing in pricing as well.

The agile development is applicable for the projects where there is a shared desire to share the success and risk between us and the customer. This process guidance expects and assumes that the agile project is carried out in the manner that is in the spirit and letter of [agile manifesto](#). This means that the project MUST be sold in a manner accordingly, preferably as allocated time of an agile development team and the customer shall retain a keen interest to the project all through it.


Though this decision belongs to all software development projects of our company, its consequences to the security issues are such that we cannot exaggerate its importance at the initiation of development process.

Decision: normal or enhanced security and privacy

The key decision regarding if a development project shall be realised with higher attention to the security and privacy is to be based on customer needs and the financial impact derived from them.

The CSO office shall provide and maintain a decision making tool that allows making a weighed assessment of needs producing a development recommendation regarding the process to be followed.

Decision criteria

SDLC Decicion Criteria	
	Use the Digia SDLC decision criteria , to determine the appropriate process.

Informing CSO office of process adoption

All projects that adopt this process shall inform the CSO office that their efforts will be conducted with it. This is done in order to maintain the process quality and provide the efforts all due support from the CSO's organization. (Note: We *do* want to be able to support the projects in adoption of the process.)

The informing MUST be done, by adding the project Name, contact person and when available JIRA and Wiki-page links to the page [List of projects that have started using updated Secure Development](#).

Software development in rigid project setting and enhanced security & privacy context

This variant of the process is for the plan driven approach.

At this revision we will not reinvent the wheel, but choose to use Microsoft Secure Development Lifecycle (MS SDL) as our process of choice when dealing with plan driven project settings. Training material for MS SDL can currently be found at <https://academy.digia.com>.

Software development in semi-rigid project setting and enhanced security & privacy context

This variant of the process is for the waterfall or very much waterfall like phased project management approach.

At this revision we will not reinvent the wheel, but choose to use Microsoft Secure Development Lifecycle (MS SDL) as our process of choice when dealing with plan driven project settings. Training material for MS SDL can currently be found at <https://academy.digia.com>.

Software development in agile project setting enhanced security & privacy context

Agile development contrary to the classic plan driven and similar approaches cannot by definition rely on upfront design work to take care of the proper ramp up of security features in projects. In these projects ¹ the security and privacy issues need to be addressed differently. As stated above, the first thing that needs to be done is the decision on if the work is to be carried in normal way or the manner that puts more organised and heavier emphasis on security and privacy in the development work. If latter, then the process requirements described below MUST be added to the basic (agile) process being followed.

Terms Used

Study and revisit these, as they are crucial in understanding the following process description.

Term	Meaning in this context
Agile development	<ul style="list-style-type: none">• Software development done in the spirit and guided by the principles of the agile manifesto
Agile Team	<ul style="list-style-type: none">• The entire team. In scrum the agile team includes a development team, PO and the possible ScrumMaster.• A cross-functional team that has all the capabilities required for realising the development effort.• This team as a whole is self organising and should have the autonomy to perform its work
Development Team	<ul style="list-style-type: none">• The cross-functional team of developers and designers that actually produces the product.• The people that work to implement the backlog items.• The development team is also self organising which means that if the PO is not doing work as one of the developers he is not entitled to tell them how to change the backlog items into working software, beyond the functional descriptions and quality control constraints written in the backlog.
Microsoft Secure Development Lifecycle	<ul style="list-style-type: none">• A process model defined by Microsoft promising that "The Security Development Lifecycle (SDL) is a software development process that helps developers build more secure software and address security compliance requirements while reducing development cost"

<p>Product Owner (PO)</p>	<ul style="list-style-type: none"> • Product Owner represents the entirety of customer in the Agile team. His job is to ascertain that the backlog items are clearly expressed and well ordered so that the value the team produces is optimised while still preserving the technical and conceptual integrity of the product they are doing. • The Product Owner is the sole authority on the order of backlog items and in accepting the development team's implementation of the backlog items ie. the work to be done. The first part here means that he gets to say what is to be achieved and the second, what level of quality is the desired and acceptable. • SAFe has a lovely description of the Product Owner and even though this is for a singular team, only a part of a product it is still useful for most. <ul style="list-style-type: none"> • The Product Owner (PO) is the member of the Agile Team responsible for defining Stories and prioritizing the Team Backlog to streamline the execution of program priorities, while maintaining conceptual and technical integrity of the Features or components for the team. The PO has a significant role in quality control, and is the only team member empowered to accept stories as done. For most enterprises converting to Agile, this is a new and critical role, typically translating into a full-time job, requiring one PO to support each Agile Team (or, at most, two teams). The role has significant relationships and responsibilities outside the local team, including working with Product Management, who is responsible for the Program Backlog, to prepare for the PI Planning meeting. • Note: At current version of the process in the context of Scaled agile development the term PO in this document is still at places referring to the whole of the product/ Program/ Solution management team that we collectively call Release Management Team in this process guidance.
<p>Release Management Team (RMT)</p>	<ul style="list-style-type: none"> • The SAFe has train staff that deals with issues across teams. In a single Agile Release Train the staff consists of System Architect/ System Engineer, Product Management and Release Train Engineer and they are supported by Business owners. When SAFe is scaled to larger solutions the solution train gains its own train personnel. Release Management Team here refers to any team of people at train level that has authority to approve work and set guide lines on what is to be achieved at system development. • In short, the Release Management Team has the owner's power at train level.
<p>Rigid development</p>	<ul style="list-style-type: none"> • Traditional plan driven / waterfall model software development

Secure development process steps

This process discusses a number of steps that are expected to take place while we are developing software in the agile manner with enhanced security and privacy concerns. Some of these are aspects and elaborations of existing agile project events.

The security refinement can be a separate event, but just as easily it can be only an aspect of the standard run of the mill kind of backlog refinement process. Iterative progress happens. As this is a method agnostic process refinement the actual iterative progress can be a standard sprint from Scrum, a longer stretch containing several sprints or a SAFe's product increment run of several iterations.

The security audit can be included in the normal sprint review or it can be a strict audit run by the security auditors available for our company / external subcontracting. The point is that it MUST be done.

Secure Agile Process Initiation

Due to the very basic software development needs the process has to be initiated according to the project management process. It can be a lean process, utilizing KanBan visualisation, a classic Scrum, XP or any of the other agile approaches. It can be a scaled setup a simple scrum of scrums or a full blown SAFe implementation. These are all decisions that need to be made here.

In secure process in addition to the development decisions the security and privacy related aspects need to be addressed as well. This means that the security competences necessary need to be identified and a suitable agile / development team found for the project. It should be noted that all changes to the team set-up will have an impact on the team performance. Also, the Product Owner needs to be found / identified. If the process framework is SAFe, that means finding the program or solution management, system architects and the release train engineers who can set the train in motion. Roles section describes additional security and privacy related roles, that should be considered strongly.

At the end of Agile Secure Process Initiation we should have

- at least one cross-functional agile team staffed,
- project's / product's initial vision statement, and
- first draft of Definition of Done drawn.

During all of the process, the work MUST follow the guiding [Security and Privacy Design guiding principles](#). These form the foundation of the secure practices in a project.

There are further constraints on development to guarantee security & privacy claims. These claims include but are not limited to such as "the solutions guarantees the privacy of people whose information is handled" or "The web-interface is secured against common intruders". Typically such claims are backed up by showing the measures that have been taken to achieve those results and so, the security driven project needs to produce documentation to this effect.

Input

- Product vision -suggestion
- Some suggestions on what might be a killer app and key features in MVP
- Feasibility Study or similar
- Staffing needs suggested by feasibility study and product / project scope
- Preliminary PIA/DPIA stages for the business process identified so far

Output

- Refined Product Vision
- Initial team or in case of a scaled production train line up
- Agile Playbook
 - Security Constraints for the development
 - Communication lines
 - Requirements concerning premises etc
 - Documentation plan
 - Privacy by design - design documentation
 - Security design documents
 - Security audit documents
 - Security testing documents

Evaluation: Product Owner and Release Management personnel Readiness

In this process the Product Owner needs to be security and privacy savvy and this is important enough issue to merit a process phase.

At this stage the organisation needs to assure that the person taking the ownership responsibility of the effort has skills and knowledge to make the decisions during development. The evaluation can be as simple as seeing that the person has the formal proficiency, he possesses creditable education in the field or a certificate from a recognized authority or it can be had by having the prospective owner carefully interviewed and vetted by our security experts leading them to pronounce him capable for the job.

In the end this is likely to be an executive decision, but it should be based on such evaluation that the executive in case can be fully ascertained of the rightness of their decision. In case it turns out that the executive made a wrong decision based on a haphazard evaluation process, the executive ought to be held accountable and sanctioned severely in order to avoid similar mistakes to be made in future.

Input

- [The PO Competence profile](#)
- The desired PO Security Competence profile, competence needs indicated by the production
- Executive to compare the two and make the decision, could be from CSO Office or delegate

Output

- Decision of the PO's capability and actions with which the adequate security capability for Product Management can be achieved.
- Documentation of the decision including reasoning behind it.

PO Security training

If in the evaluation it turns out that the prospective PO does not possess the necessary security knowledge and evaluation skills and we still want to have that particular person in the role, the PO should undergo a training providing him with the skills necessary. Due to the importance these skills have in the successful performance of the role, the training should contain assessment that verifies that the person has obtained the required knowledge and/ or skill set.

Security trainings will be organized by Digia Academy; also projects are required to discuss external trainings, if suitable trainings are not in schedule in suitable time frame.

Backlog security and privacy refinement

During this stage the known key epics and stories are refined with the security angle in mind. Most agile practices maintain that only the readily known stories near the top of the backlog are defined in the fine grained level as the ones waiting for further work down the line remain in only generally defined state. At this time the less defined stories should be looked into in passing with the question "Does the story at this state and stage have a security impact". If the answer is no, it should not be defined further as that would create the undesired upfront design work that the agile process seeks to avoid. The story should be looked into again once it is refined to such level that its security implications can be analysed.

The known and understood stories however should be looked keenly and evaluated as for their security and privacy impacts.

At the first time the process enters this stage the product's initial Definition of Done should also be drawn up if it wasn't already and the agile team should - with their PO in lead see how they should incorporate prescribed criteria from the tools section of this process. With this done the PO should then see if the project would benefit from having the security stories from the tools section added to the backlog as well. With these additions and refinements of the backlog the security should be addressed adequately for the starting iteration. Security refinement can be returned to from the product iterations and after a release, but in an ideal world the PO will include this work to their normal product backlog maintenance.

A quick recap of agile requirements

In agile development the requirements are generally not requirements, but stories and constraints discussed by the whole team and negotiated between the developers and Product Owner. This is because a requirement would be an ironclad and immutable, clearly and unambiguously defined desire of a customer that a provider commits to deliver as specified. Generally in software development this kind of understanding is difficult if not impossible to achieve and during the decades the industry and profession has engaged in commercial software production we have learned that such assumption outside of classroom context nearly always fails and quite often catastrophically. The lack of certainty being the customary condition in software development and acceptance of that situation is what the agile-movement seeks to address and cope with. Because very little can be known at the beginning of projects, no decisions are made with faulty and lacking understanding, but instead the vague desires get expressed as stories of user interactions are elicited and negotiated - in bona fide manner - by and between the agile teams and their product owners.

The classic form of the story is "As a <user role> I can <do something relevant> in order to <gain something that has actual business value>. Constraints can be architectural quality attributes or as they are more commonly called non-functional requirements such as modifiability, configurability or other -ilities or procedural requirements such as a requirement that all stories to be considered done need to have a functional automated tests written for them as well as the core functionality in the system.

The procedural requirements are either in the individual stories' acceptance criteria or in the project's common definition of done. The definition of done is by best practices a shared artefact for all stories of given kind, but it can be recapitulated at individual story items every single time if such is desired.

Security requirements in agile development

Security issues can be brought to the team in two ways. They can be introduced to the backlog as stories especially in such cases where they have actual interaction or they can be brought in as constraints and acceptance criteria either on the project or story item level.

In either case it is the responsibility of the Product Owner to make certain that the security and privacy issues are adequately addressed for each backlog item. The product owners are strongly encouraged to use Digia Security experts if they are at all uncertain of how the security issues should be handled. The developers - especially the developers specialising in security enhanced development work - are equally encouraged to present the PO with such questions and suggestions that help the PO to make the proper additions and adjustments to the acceptance criteria and backlog in general to make the system under construction actually secure.

The tools section of this process guideline seeks to help POs to adjust their backlogs to produce security and privacy retaining systems. The process also defines a set of stories and acceptance criteria mandatory for all projects wishing to adhere to the process and a further set that is mandatory to the web based software systems which at this writing form the bulk of the business systems we are providing.

The privacy requirements in agile development

The key issue in privacy is recognising the personal information, the impact of its compromise would have on the data subjects and the protections that we have against such compromises. EU legislation (GDPR) requires Privacy by Default and Privacy by Design principles to be adhered in addition to a plethora* of Data Subject Rights. Privacy by design is also a guiding principle in this process for the full versions, please read through [Digia GDPR](#) and [Digia Privacy Policy](#).

1. Personal data information flows, storing and processing including the full content must be known and analyzed.
2. After this initial PI data analysis, PI data is then analysed with the default tool for addressing privacy impact issues. The default tool is the PIA / DPIA analysis. The acronyms stand for Privacy Impact Analysis and Data Protection Impact Analysis respectively. There are ready forms and tools for conducting the analysis, we can somewhat roughly say that the key in both is identifying the personal information (initial analysis), evaluating and analysing possible impacts of exposure (PIA); and
3. then modelling the protection (controls) that the PI data requires i.e. the flow of this information through the system and the protections that we have/ have planned for this information and
4. then analyse if all is as it ought to be or should we change something.
5. Systems handling PI data must have PI register handling declaration (käsittelyseloste); there are ready templates for this in Security Wiki.

For the development purposes this analysis might change the existing stories directly or it might bring in additional stories. Security controls for the data to be protected, come in the form of

backlog items, which are implemented in the system; but this being agile development, the answer might also be to change the flow of data in the business process. This latter can be a product owner's domain or more backlog work in the domain of service design (can we eliminate the hazardous stages from the information flow by changing the business process that this software system is supposedly serving?).

Input

- Existing Product Backlog
- Existing Definition of Done
- Product Vision
- The mandatory and recommended stories from this guidance
- Existing previous or preliminary PIA/DPIA analysis

Output

- Product Backlog enhanced with the mandatory and recommended stories
 - Some of the initial epics, features and stories might have been enhanced with the recommended acceptance criteria security constraints
- The stories that have been possibly altered due to findings of the PIA/DPIA
- The PIA/DPIA analysis that describes the system's planned release.
- The revisited DoD that now includes the DoD level acceptance criteria derived from this process guidance
- Decision making documentation in the product's documentation that records
 - Which stories were added from this process guidance
 - How the DoD was altered at this stage
 - When any of the mandatory and recommended stories and criteria are left out, the reasoning behind it and estimates on security and privacy impacts for the product

Iterative progress

This process is defined as agnostic to the methods of agile development. The iterative progress in the development can thus be done according to most any agile framework. It should be noted that Digia as a company is moving toward preferring [Scrum](#) and [SAFE](#) as our go to approaches and this process relies on the agile framework having an inspect and adapt type of development cycle where the results of development efforts are periodically inspected and the product backlog as well as the ways of working are adapted to correct the course.

Due to the development process agnostic approach, this part is intentionally left vague in description but armed with a set of tools. These tools take the format of guiding principles and prescribed epics, stories and acceptance criteria constraints of which the latter can be applied to either individual stories or to the Definition of Done shared by all stories of the project. At some of the stories the recommendation is to have a story describing rigorous testing of an existing increment followed by one or more hardening iterations which will address found issues that are deemed sufficiently serious to merit the development time required to address them. These kinds of patterns are possible only due to the agile nature of development. If the delivery time is set in stone, only the level of security planned from the start is possible.

Final Security Audit

Before every release the developed solution should be inspected and tested in order to guarantee that it fulfils the promises we are making regarding the safety it provides. This includes fulfilling the contract with the client.

Depending on the targeted security level, release cycle and the practices the team has adopted / steered to adopt in its iterations this can be a full blown security audit arranged by our own 1st party or external 3rd party organisation or it can be a simple demonstration of the required analysis and tests done as a part of the run of the mill sprint review.

Whatever the approach in the actual context is, it has to provide us with the guarantee that we and our product are keeping our promises. It must ascertain that the customers and end users getting involved with our system are getting the level of security we say they are, or better. When we are auditing software for higher levels of security it is required to use independent 3rd party auditors outside the development team or train in order to maintain independency of the audit.

There exists separate guidance on security audits under Security & Privacy Process in CPM.

One example of audit criteria for web applications can be found in the [OWASP Application verification standard](#). If the product is to be developed to acceptable higher assurance level, the target level has to be set at 3. The level is negotiable with the customer; typically clients and the

security world views the world through requirements, and a typical requirement is to have the end product be tested against OWASP without explicit mention on what ASV level.

Input

- Product Increment,
- Source code for the components developed
- Design and development documentation of the components (like comments in backlog items, notes made to project documents etc)
- Team
- Auditors

Output

- Audit report documenting the security of the produced solution.
 - This can be a project wiki page, the content should be prioritized over the form.
- Stamp for the release
 - Implementation hint: Audit stamp could very well be a JIRA-Issue that is resolved with a pass resolution, when the audit goes through. This is still a documentation of audit passing with a responsible auditor recorded in the issue history.

Release

To release a product increment is to make it available for its intended audience. The core act can be deploying the software to servers or just make an installation package available for the customer's little worker gnomes to take and install to what environment they see fit. The auxiliary actions would be publishing the release notes etc.

Some agile productions can release every increment they make. Typically the releases are however done in a sparser schedule, and in this process release to the customer is a noticeable milestone in development efforts.

We define the release as consisting of having an increment available or deployed (preferably in an automated fashion) to the customer(s) of the production and the customers to be informed of the issues relevant to this software release. What new features were included and what bugs were fixed.

Due to the nature of the secure development process it is mandatory to have a separate section for the security fixes that the release addresses. Hopefully most of the other work related to release is automated, but we expect that producing the release notes is a non-trivial undertaking at this stage. In addition to that an informed and diligent release decision is to be made.

Input

- Product Increment
- Final Security Audit stamp / report

Output

- Product release

Tools

The primary tools offered by the process shall be principles, mandatory and recommended stories and acceptance criteria constraints, example stories, related requirements and best practices. Beyond these there will be decisions support tools and security protocol standards.

- [Security and privacy design principles guiding our work](#)
 - The principles guiding our work have been separated to their own section. Any and all work aiming for enhanced security and privacy should adhere to them faithfully.
- [Pattern Library of MANDATORY and generic patterns of SDLC](#)
 - Beyond abstract principles software is developed with user stories and acceptance criteria constraints. Our story and criteria patterns are separated to their own section and in there divided to mandatory and recommended.

Process verification

To ensure that it is possible to verify that the agile process has worked in a manner intended and prescribed by this guidance, any undertaking using the process shall retain in use or as archived the following items:

- Versioned source code with full revision history from the initial version onwards
- Log of code reviews
 - If the project used GitLab or similar system for code reviewing the system automatically retains the code review information unless it is destroyed
- Issue management history
 - If JIRA or similar system is used this happens automatically as long as the JIRA project and its history is not destroyed. The projects using this process model shall ensure that their issue management history is retained by the IT.
 - If the project uses a post-it on a whiteboard kind of backlog the PO /RMT is responsible for retaining an electronic copy of issue history
- Security Audit history
- Project documentation history
 - Agile projects in general maintain such documentation that they deem fit. Project wiki / Agile playbook etc. The projects using this process shall retain a versioned history of their documentation all the way through their life span. They shall especially retain documentation of what their Definition of Done at any given time has been, but also of their agreed upon conventions, coding style, review practices etc.
 - A versioned history of project wiki is deemed sufficient. IT shall ascertain the persistence of such history.

Process quality measurement and metrics

The quality that this process produces for projects is two fold. The first and primary indicator of process quality (by agile principles) should always be the working software. If the software works as intended with the set functional and non-functional requirements it is good software, software of high quality we can truly be proud of. This means that this process's primary quality is its ability to produce *well functioning software which lives up to the promises we make concerning its S&P features*. The secondary quality of the process is its ability to support and facilitate the consideration of S&P aspects of the development process.

The primary quality is quite boolean in nature, there is no partial success for realising S&P features, there is only success or failure. Measuring that quality can thus be achieved by the results of audits (provided that these audits themselves have been carried out in professionally). Reducing that to metric is no simple feat, but the relation of passed and failed audits per releases ought to provide a followable and comparable metric sufficient for the standardised process and useful for the company management.

Mandatory metrics:

- Metric: Final Security Audit results of a given project, calculating pass / fail / releases
 - Same for projects adhering to the process
 - Comparison with results from similar audits for projects using some other process
- Metric: Sprints / Increments failed because of problems with S&P issues

Currently optional metrics:

- Metric: Developer feedback
 - Based on question "How well would you say that the process features provides for security and privacy" with a rating from 1 to 5 where one is poorly and 5 excellent.
- Metric: PO feedback
 - Based on question "How well would you say that the process features provides for security and privacy" with a rating from 1 to 5 where one is poorly and 5 excellent.
- Metric: Customer satisfaction with the S&P results / Increment (increment in the sense that the term is used within Scrum framework)
 - Based on questions:
 - "Do you have complaints about security issues in the provided software system" with a rating from 1 to 5 where 5 is none, 3 is some actually and 1 is quite a lot.
 - "Do you have complaints about privacy issues in the provided software system" with a rating from 1 to 5 where 5 is none, 3 is some actually and 1 is quite a lot.
 - "How satisfied are you to the system as a whole, usable, secure and privacy protecting system" with a rating from 1 to 5 where 1 is not satisfied at all and 5 is quite happy indeed.

Quality goals for the process

The process quality goals shall be based on the quality metrics produced by projects adhering to it. What they shall reflect are the quality attributes produced by the process. The goals shall be set for the minimum and average / mean rates with the plan that the process is rolled in during 2018 and starts to produce measurable results at the early 2019. As the process itself is intended to be agile (i.e. improvable), the produced quality shall be followed and the process tweaked in order to improve the results.

In H2/2018

In H1/2019

In H2/2019

Mapping to the Software Development Process

This process co-exists with the company's general development process and though it is designed to be used in conjunction with any software development process we of course do not recommend abandoning the Digia's default process to favour something else.

This section discusses mapping the process steps from here to default process' steps.

Process element	New Development phase (s)	Notes
"Decision: Enhanced or Normal sec & priv"	Sales, Project Initiation	When the development is done like it usually in our company would, based on customer requests and sold either as an honest project or some sort of a development effort, the basic decision of should the development be done with the basic or enhanced security and privacy is likely part of a sales process as this will have a non-trivial impact on efforts and pricing. At the very latest stage the decision must be made at project or product initiation.
"Decision: Waterfall, semi-rigid or agile development"	Sales, Project Initiation	Equally to the previous decision the way forward must be decided at the onset. If the customer demands a traditional project, then we must plan our actions accordingly and when we are given room for agile, we must make most of that opportunity.
"Agile Secure Process Initiation"	Initiation, Kickoff	Agile secure process initiation maps to the initiation and kick-off stages of the standard process' agile initiation phase. The key differences are that the product-vision must be honed holding the security principles that guide this process in mind and the

		security minded view must be adopted in other initiation activities as well.
"Evaluation: PO Readiness"	Initiation	At the initiation time the PO presence is no doubt one of the key factors that needs to be settled for the agile process. If the project /product is to be developed with customer's PO whose commitment to development needs cannot be ascertained and the process is consequently fitted with Digia Business Analyst, the PO readiness evaluation is to be carried out for that BA as well. This work fits with the staffing function of the initiation process.
"PO Security training"	Initiation, must be completed during kickoff	When the PO's capabilities are lacking the remedial measures need to be completed preferably during initiation but at very latest during kick off.
"PO support personnel addition (BA, Sec Analyst etc.)"	Initiation, must be completed during kickoff	When the PO's capabilities are lacking the remedial measures need to be completed preferably during initiation but at very latest during kick off.
"Backlog security refinement"	Kickoff, Planning	The Backlog security refinement is an undertaking that is first addressed during kickoff when the agile play book and product backlog are further populated and refined from the process guidance. Backlog security refinement is revisited cyclically as long as development is ongoing.
"Iterative progress"	Planning, Development	The iterative progress in this process refers to the normal progress of the agile methodology used. In the Digia process that is Planning, Development cycle that contains the normal test development.
"Final Security Audit"	Development, Pre increment	Final security audit is an activity that has to take place before a release of increment is done. Thus it needs to take place in the development cycle before release or before a hardening iteration where no degradation of security can happen.
Release	Product Increment	The release in this process maps directly to the Product

	Increment of the default process. Key difference is that release cannot be done in the hyperagile fashion fully automated as security constraints require an informed decision before deployment of software.
--	--

References

Latest OWASP Top-10: https://www.owasp.org/index.php/Top_10_2013-Top_10c

[Cavoukian] Privacy by Design The 7 Foundational Principles, Ann Cavoukian https://iab.org/wp-content/IAB-uploads/2011/03/fred_carter.pdf

*a long list of

Security and Privacy Design guiding principles

#1 Keep it simple

Whatever the solution is, it should be kept as simple as possible. Truth of the matter is that more we add complexity to a given solution or system, more difficult we make it to properly understand and use it. Thus it holds that if we need to increase security of some design, we need to decrease complexity and ambiguity related to it. Also it so happens that if we look at software development from the non-security related perspective, KISS is still an excellent engineering principle to follow.

#2 Defence in depth

Defence in depth means that we should avoid relying to only one layer of defence and instead stack defences. The classic example of the principle is that the data lying in our servers should be protected by a firewall, but in case it fails it would be nice if the data in question also happened to be encrypted. In this example the encryption is the second tier of defence. The principle is further examined in the pattern examples

#3 Privacy by design

Privacy by design is an information management principle that we ought to follow in designing systems we produce within the process described here in order to ascertain that we are most likely making a system where the privacy requirements are adequately addressed. The principle can be further subdivided into seven foundational principles that provide more specific guidance for our work.

At Digia this foundational rule should and shall be considered in conjunction to the European Union's General Data Protection Regulation ([GDPR](#)) which is well in harmony with as these principles have been driving the regulation as well.

Also read [Digia GDPR](#).

Proactive not Reactive, Preventative not Remedial

The first foundation rule of the principle is to acknowledge and give explicit recognition to the value of privacy. Thus the organization following the principle will also recognize the value and benefits of proactively adopting strong privacy practices, early and consistently. This rule implies that in a context where the principle is followed there should be a clear, pervasive and demonstrable privacy commitment.

Privacy as Default

As design goes the collection and sharing of data should always be minimal. When it comes to interactions with the system when an individual does nothing that should have no effect, the absolute minimum impact on privacy should happen, further disclosure should be the one that requires a separate action. If an individual does nothing, their privacy still remains intact. No action is required on the part of the individual to protect their privacy it is built into the system, by default. This foundation rule can be subdivided into four rather concrete practices. [Ann Cavoukian](#) gives them in the following form

- **Purpose specification** - the purposes for which personal information is collected, used, retained and disclosed shall be communicated to the individual (data subject) at or before the time the information is collected. Specified purposes should be clear, limited and relevant to the circumstances.
- **Collection Limitation** – the collection of personal information must be fair, lawful and limited to that which is necessary for the specified purposes.
- **Data Minimization** the collection of personally identifiable information should be kept to a strict minimum. The design of programs, information and communications technologies, and systems should begin with non-identifiable interactions and transactions, as the default. Wherever possible, identifiability, observability, and linkability of personal information should be minimized.
- **Use, Retention, and Disclosure Limitation** – the use, retention, and disclosure of personal information shall be limited to the relevant purposes identified to the individual, for which he or she has consented, except where otherwise required by law. Personal information shall be

- #1 Keep it simple
- #2 Defence in depth
- #3 Privacy by design
 - Proactive not Reactive, Preventative not Remedial
 - Privacy as Default
 - Privacy embedded into design
 - Full functionality - Positive sum not zero-sum
 - End to end security - lifecycle protection
 - Visibility and transparency
 - Respect for user privacy
- Respect for the data subject rights, [GDPR guidance on privacy](#)
 - Consent
 - Data Subject Rights

retained only as long as necessary to fulfill the stated purposes, and then securely destroyed.

It is worth of noting that where there is no clarity of the need or use of personal information, the presumptions of privacy and precaution should prevail and the settings regarding privacy should be the most protective ones.

Also read [Digia GDPR](#).

Privacy embedded into design

At all points of data flow in our systems where personal information is handled the privacy impacts should be considered and mitigated. This should be done from as early stage of development as possible. In every access of the personal information the access should be limited to the absolute minimum. We can also say that the privacy shall be considered an essential component of the core functionality being delivered or if such an intangible is beyond our grasp, then a quality attribute of the more readily understandable functions. Cavoukian further elaborates the foundational rules by stating that in a system where the privacy is embedded into design

- A systemic, principled approach to embedding privacy should be adopted. Such an approach should rely upon accepted standards and frameworks, which readily cater to external reviews and audits.
- All fair information practices should be applied with equal rigour at every step of development and operation of the system.
- Wherever possible, detailed privacy impact and risk assessments should be carried out and published, clearly documenting the privacy risks and all measures taken to mitigate those risks, including consideration of alternatives and the selection of metrics.
- The privacy impacts of the resulting technology, operation or information architecture, and their uses, should be demonstrably minimized, and not easily degraded through use, misconfiguration or error.

So in short, by this rule privacy should be a key consideration in all parts of our system design.

Full functionality - Positive sum not zero-sum

An interesting part of the principle is the requirement not to balance privacy against other legitimate interests and objectives, but to instead *seek win-win situations where nothing is sacrificed in order to provide the privacy but the other requirements are satisfied as well as the privacy*. As an example instead of having a false dichotomy such as privacy versus security, the principle privacy by design seeks to illustrate that both should be there. This means that embedding privacy to the system should be done so that nothing of the functionality is sacrificed and when the objectives are negotiated the zero-sum game is to be avoided.

End to end security - lifecycle protection

The privacy by design is an information management principle. A system where personal information is managed and its operator takes a responsibility for this information including a responsibility to protect it. This means that a system which contains personal data should be designed to protect that data as long as that system and or data exist. This means that adequate security measures are included (ie what the most of the enhanced process is all about) and that the applied security measures dictated by sensitivity of the personal information are present including e.g. such things as logging the access to and the destruction of the data containing personal information.

Visibility and transparency

These two are key attributes of practices ascertaining accountability and trust. In our work they translate into design objectives stating that the system we are creating should make all usage and access of personal information readily auditable and understandable to all stakeholders. This typically translates as a companion requirement to user stories or use cases related to access of personal data.

Respect for user privacy

Respect for the users' privacy quite naturally lies at the heart of the principle. If we have it we naturally follow the constraints dictated by Privacy by Design and if we lack it we will be at best paying lip service to this principle. One of the easiest applications of this sub-principle is the empowering of the data subjects themselves to play an active role in the management of their own data. This can begin from applying the elementary fair information practices of consent (the subject need to give explicit, informed and uncoerced consent to collection, use and / or disclosure of their data), accuracy (personal information shall be as accurate, complete, and up-to-date as is necessary to fulfill the specified purposes), access (Individuals shall be provided access to their

personal information and informed of its uses and disclosures. Individuals shall be able to challenge the accuracy and completeness of the information and have it amended as appropriate) and compliance (Organizations must establish complaint and redress mechanisms, and communicate information about them to the public, including how to access the next level of appeal).

Going beyond these practices the sub-principle is reflected in the user experience of the system, meaning that in order to comply with the sub-principle the user interfaces should be human centered, user-centric and user-friendly in order to - as Cavoukian states - allow users to reliably make informed privacy decisions.

Respect for the data subject rights, GDPR guidance on privacy

For us the General Data Protection Regulation (GDPR) is obligatory guidance regarding privacy related matters. In general this part of union legislation exists to secure that data operators respect the citizens' rights regarding their personal information. This process deals with the legislation as it is the software that we produce that needs to enable and empower the data operators in putting these principles to practice and acting in accordance with the legislation. As GDPR is strong and demanding legislation where failure to oblige is met with severe sanctions and repercussions our customers need to be able to irrefutably show that they have earnestly made all that is required from them, Thus all that is demanded of them needs to be auditable and in the process, archived.

Since the GDPR is non-elective legislation containing guidance on addressing privacy from the design level onwards in software systems, our development process needs to not only ascertain that we work by these norms but also that we document that in a manner that the authorities and in extremis legal courts may judge us as having practised all due care and diligence in our work.

The GDPR is fully documented at [Digia GDPR](#), [Digia Privacy Policy](#) and related documentation in [Security and Privacy Wiki](#).

Consent

The first requirement the legislation makes is that unless there is an explicit reason in the law allowing PI data processing, the *data operators must ask for informed consent on collecting and dealing with personal information*. The conditions for making this consent informed are such that it is not acceptable for the operators to hide behind long and complex terms and conditions full of legalese, but the request must be made in an intelligible and easily acceptable form, with the purpose for data processing attached. The consent needs to be actively and intentionally given, so pre-filled forms do not meet the requirements. Furthermore the consent must be unambiguous and clearly distinguishable from all other matters, not hidden for example amongst application's user interface preferences.

The consent needs to be considered a thing with a life cycle, so that it is as easy to withdraw as it is to give.

Data Subject Rights

GDPR makes a number of data subject right explicit and unambiguous. These rights are at the heart of the legislation and respecting them is a key principle when we try to write software.

- *Right to be protected when breach occurs*
 - When a breach of confidentiality is noticed and this breach is likely to "result in a risk for the rights and freedoms of individuals" (which in practice means whenever personal data is in question) the due authorities need to be informed of this within 72 hours of the incident and the impacted people themselves without undue delay.
- *Right to Access*
 - Data subjects have the right to obtain from the data controller confirmation on if personal data is being processed, where and what purpose. The controller needs to be able to also provide a copy of personal data, free of charge, in electronic format. This can be addressed as a dedicated user story as we show in the story pattern part of this process's tools section.
- *Right to be Forgotten*
 - The data subjects have an absolute right to be forgotten which means that any and all data concerning them must be removed and destroyed. This might have rather extensive impacts as in many cases there are issues like referential integrity where the simple removal might not seem like a valid avenue.
- Right to Data Portability
 - The data subjects have a right in order to make their own data available and utilisable to a new controller to receive it in a commonly used and machine readable format and transfer it to that other controller of their choice.
- Privacy by Design

- GDPR includes as article 23 a requirement that translates into a requirement to treat Privacy by Design as a guiding principle. It is thus not only a part, but a central mandatory part of the principles that guide our work. Privacy by Design is extensively discussed above and where applicable we will offer examples in translating it into stories and constraints. However since it is a perspective and principle rather than a set of practices it is best treated as such, a principle that all of our personnel involved in software development knows and applies to their work on daily basis.
- *Data Protection Officers (and their tools)*
 - tbw

<http://www.eugdpr.org/>

Pattern Library of MANDATORY and generic patterns of SDLC

Stories and acceptance criteria, how the patterns work

Many agile coaches and evangelists recommend the product owners and teams to make effort in *expressing as many of the requirements as they can in a story format*. Sometimes this is impossible, but most often carefully considering the possible user roles involved in the system will enable the POs and teams to express requirements in a story format. In such endeavours one should remember that a system can have many kinds of users, not just end users, some of which might even be transitional. For example an auditor might interface the system not directly but via some other system, which in turn needs to interface or be interfaced with the system currently being under development. For this example auditor's needs we might for example need to log user interactions with the system and then make certain that the logs are available to the auditor via what ever solution that fits his needs the best.

Story patterns (general)

Story patterns are written in generic form. They are followed by the MANDATORY story patterns and acceptance criteria.

2-factor authentication

Two factor authentication is a good example of a user story as its variants describe the user's interaction with the system. Two factor authentication is often described by security experts as a form of authentication combining something that the user knows with something that the user has. This can take many forms such as:

As a user logging in to the system, I want to initiate the procedure by inserting my keycard to reader attached to my terminal and then sign against the user data indicated by that key in order to have a smooth flow on the signing in process while being ascertained that malicious users would have a very hard time getting to the system as me.

or

As one of the end users of the system I want to have my signing in double checked by a pin challenge against a one time password sent to MY cellphone so that I can use feature's requiring strong authentication in the system.

or

As a logged in user of the system I want to be further challenged with a pin code sent to a cell number registered to the system when I initiate the action X so that should I have forgotten to log out of the system my account could not be maliciously used to do X by a third party.

As an observant reader may note the first two of these are general access control to the system and the third one is for a particular action that requires elevation of rights. The rationale or business reason is described from the perspective of the particular user performing operations with the system. These same stories could also be seen from perspective of a system operator and thus a single implementation could actually address two stories.

e.g.

As an operator of a medical facility I want to ascertain that the person accessing patient records as patient is indeed the patient himself and thus I want to check that he knows the password he has currently set in the system and is also in possession of the unique patient tokens given to all of our patients. This is most important as I am by law required to make certain that the very intimate patient records will in no circumstances get into the hands of third parties.

As a PO/ Operator / Responsible I want to establish a border of trust between A and B in order to further secure B.

- Stories and acceptance criteria, how the patterns work
 - Story patterns (general)
 - 2-factor authentication
 - As a PO/ Operator / Responsible I want to establish a border of trust between A and B in order to further secure B.
 - Acceptance criteria patterns.
 - DoD + attack surface analysis for the new interface
 - DoD + traceability criteria
- Mandatory Story and Acceptance Criteria patterns
 - For all projects
 - Stories & Epics
 - MA
 - ND
 - AT
 - OR
 - Y:
 - As
 - a
 - Bus
 - ine
 - ss
 - Ow
 - ner
 - I
 - wa
 - nt
 - the
 - pro
 - duc
 - t to
 - hav
 - e a
 - ver
 - sio
 - n
 - ma
 - nag
 - em
 - ent
 - stra
 - teg

This story aims to add more security by separating A from B. Border of trust means that all input from A needs to be treated as untrusted and thus a subject to sanitation and validation operations. This story needs to be written out so that it reflects the context. It could be separating the components with networking means such as a Firewall to provide protection at network level or it could be setting up data clearing features at the border.

Acceptance criteria patterns.

In the introduction of pattern examples we discussed a hypothetical auditor of the system and later in the acceptance criteria side we will now discuss traceability as an acceptance criteria. The criteria has been given in a form that is used in addition to DoD that is being used.

DoD + attack surface analysis for the new interface

This kind of an acceptance criteria addition can and likely should be made e.g. for stories where implementation clearly adds to outwards facing interfaces. In this example it should be noted that the outwards facing interfaces can occur at any architectural layer of the solution and the security principle of defence in depth guides us to address them at any such point where they occur.

DoD + traceability criteria

Traceability is one of the classic -ilities, a non-functional requirement or software quality attribute if one follows the Carnegie-Mellon SEI's architecture based approach. If traceability is added to the DoD for a given story it can usually be assumed to mean that the implementation of a story must make certain that as a side effect of enacting the story system also notes who made changes to some assets in system and when. Generally it is not a good idea to leave anything in acceptance criteria as assumptions, but rather write these matters explicitly to the story when it is elaborated. The traceability for one story might be satisfied with a user identifier and timestamp, elsewhere it might require further information like the originating IP. Elaboration of such details is very much the responsibility of product owners and this should be done when the stories rise towards the top of the backlog in a Just In Time fashion. The developers are very much encouraged to pick all such criteria carefully apart during the refinement and planning sessions and probably should generally decline development of stories with only vague acceptance criteria written in. Regardless of best efforts most agile teams will find themselves in a situation where their sprint backlog will have stories with vague acceptance criteria and here the acknowledged approach is to have PO flesh the criteria out as soon as the situation is found out.

It pays to note that the traceability might not be a separate acceptance criteria but a part of the core DoD as well. In such a case all stories should be implemented in a traceable fashion and be inspected against this criteria when considered for acceptance by PO.

Mandatory Story and Acceptance Criteria patterns

This library of story and acceptance criteria patterns is a fleshing out part of the process and will prescribe security and privacy story fundamentals for projects adhering to the process. Having implemented the in the mandatory segment and respected the constraints in development work the Digia product and project owners should be secure in their assumption that their products meet the criteria without failing.

For all projects

Some patterns MUST be adopted by all agile projects producing software in enhanced security and privacy approach

Stories & Epics

MANDATORY: As a Business Owner I want the product to have a version management strategy that clearly defines how many versions get security fixes in order to control costs and manage the expectations of my customers.

y
that
clearly
defines
how
many
versions
get
security
fixes
in
order
to
control
costs
and
manage
the
expectations
of
my
customers.
• MANDATORY: As a Business Owner I want the product to have a version management strategy that clearly defines how many versions get security fixes in order to control costs and manage the expectations of my customers.

A secure product by convention is supported by security fixes that address the vulnerabilities found in it. To assume responsibility of providing such updates to ever increasing range of product versions is an ascertained way to destroy all semblance of profitability from the product or product line. Thus it is most important that from the outset the product adopts a strategy of how long legacy tail it will provide security patches for and establishes this in the product communications.

In short, the business MUST decide and publish how many versions of the end product will be updated and how many of them will get security updates.

MANDATORY: As a CSO I want the product issue management to differentiate security issues from regular issues within the issue management system.

The levels used to categorise the severity of the security issues shall be *critical, severe, and minor*. Individual issue's severity is derived as follows:

- CVSS base score (*or in-house scoring, built later, based on CVSS v3*) is used to estimate the severity to above categories.
- Use <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator> calculator to get the base score
- Digia classification from CVSSv3: *critical 8-10, severe 5-7, and minor 0-4*
- PO deducts the resulting category based on the score and information from the development team. The category is not be changed from the ones above.
- Finally, categorisation can be overruled by Digia security & privacy

Critical vulnerabilities typically lead into the breaking of sprint (if the development is done in Scrum) or other such reshuffling of priorities that lead into immediate and significant efforts to address the issue as soon as possible. A good example of such issues would be a 0 day vulnerability that has a published exploit. Security issues should not be categorised to critical level lightly. Security issues that fall into category critical, must be reported to CSO office; as they could cause further incident management related tasks, that are not known by the development team.

Severe category issues won't automatically lead into the breaking of sprint (in Scrum), but should cause a reconsideration of the scope to see if they can be addressed immediately. If the team can not accept increase in scope and there is nothing from the to do pile of the sprint backlog that the PO would be ready to drop out from this sprint, the vulnerability should fall to the top section of the Product Backlog from where it can be taken to the next sprint.

The rest of the vulnerabilities - (80% - 95%) - typically fall to the minor category where they become product backlog items. These should be divided into Must, should and nice to have and prioritised accordingly by the PO / product management.

MANDATORY: As a CSO I want the development, testing (staging), and production environments to be separated.

The development efforts developed according this process ought to have a separated environments for development, testing / staging and production. This separation of concerns seeks to keep such mistakes as having development essential information leakages happening in the production environments and having a safe testing place for the new features.

A second consideration is that one such environment has to be isolated from others including environments of other projects also at network level (Firewall).

MANDATORY: As a CSO I want Privacy by Design and Privacy by Default implemented in every product / service.

Both of the above principles are described in [Security and Privacy Design principles that guide our work](#).

MANDATORY: As a Product Owner I want to get an attack surface analysis for the solution, at least 1 per Epic or bigger

As attack surface analysis is a non-trivial task, it is prescribed in its entirety to be performed only once per Epic or bigger iterations. It is worth noting that this assumes that the related definition of done criteria for each story (DoD requiring that for each story its impact to attack surface is known for) is present and respected in the DoD of the team/ train.

• MA
ND
AT
OR
Y:
As
a
CS
O I
wa
nt
Priv
acy
by
Des
ign
and
Priv
acy
by
Def
ault
imp
lem
ent
ed
in
eve
ry
pro
duc
t /
ser
vice.
• MA
ND
AT
OR
Y:
As
a
Pro
duc
t
Ow
ner
I
wa
nt
to
get
an
atta
ck
surf
ace
ana
lysi
s
for
the
sol
utio
n,
at
lea
st
1
per
Epi
c
or
big
ger

• SHOULD / STRONGLY RECOMMENDED: As a Product Owner I want to have penetration test results for the solution in order to have guarantees for the security levels I am promising, (once / release based on documented risk evaluation / mandatory on government high security projects)

SHOULD / STRONGLY RECOMMENDED: As a Product Owner I want to have penetration test results for the solution in order to have guarantees for the security levels I am promising, (once / release based on documented risk evaluation / mandatory on government high security projects)

This is one of the key stories that the process prescribes to the projects and one of the most expensive components that make up the whole of the secure delivery. Penetration testing is to be done by security professionals, where this means that the professional has deep understanding and knowledge about the field as well as competence and skill in finding vulnerabilities and exploiting them.

The testing can be set up as either a *black box* or *white box* testing and at this writing of the guidance we recommend white box version due to effectiveness and cost issues. Should the POs find the stakeholder commitment and funds to organize black box testing as well, we laud the effort and accomplishment, but at present stage of the company can not require it. We however provide example on how to set up black box testing at Digia.

SHOULD / STRONGLY RECOMMENDED in some cases
MANDATORY: As a Product Owner I want to have fuzz-tests set up and passing for all interfaces of my system, (fuzzing should be done at least once / version of interface of the system based on documented risk evaluation / mandatory on government high security projects)

This story is part of the automated or semi automated tests of the system and should be set-up from the first version of interfaces that we do.

In short, fuzzing means calling an interface with invalid data and monitoring the system for crashes, memory leaks and like. Fuzzing has been very useful in revealing significant vulnerabilities and is thus required testing for all the enhanced security systems we create.

MANDATORY: As a Product Owner I want to see known and listed security issues fixed in a hardening themed iteration(s) / sprint(s) in order to raise the product to the desired level, (once / release cycle)

Hardening refers to a practice of revisiting the existing implementation with a list of found known security vulnerabilities in the system and fixing the system so that these vulnerabilities are removed. Typically these vulnerabilities would come in the format of backlog items that the team needs to refine so that at they can say what it takes to refactor the system to a faultless state regarding the particular vulnerability. This will lead into them being work estimated iterable backlog items.

The amount of significant vulnerabilities will dictate the amount of hardening iterations necessary. It is recommendable to dedicate those iterations to hardening as implementing new user stories / features could easily introduce new deficiencies or add complexity to the features that the team is seeking to improve in their security aspects.

To be honest this is a *meta-story* that should be broken down to actual stories as the hardening closes in. In the best case scenario the work is done so well that that there is no need for hardening, but in real life that is rarely the case. Thus feeding this story to backlog is intended to remind the PO of the realities of life in software development.

Typically, in the requirement driven world, the customer contract and related SLAs may dictate the priority and the time frame, in which we end up fixing known or discovered vulnerabilities. In such cases, typically the urgency is dictated by the vulnerability score (see above).

Definition of Done criteria (All Mandatory)

The POs managing development should seriously consider adding the following criteria to their Definition of Done and have good reasons to cite when they choose to ignore them.

The criteria are followed by level of item they're to be associated with. The levels assume the common division to Epics which are large entities likely taking several sprints of development time to complete whereas the story refers to a singular user story that the end user would perform in pretty much one go and the development team would implement in one iteration/ sprint.

All code is reviewed by another developer before merging to the master branch, per story

Though this is a good practice in general, it is doubly so in this context. Code review should be carried out in a way that exposes all code to sufficient oversight by the development or scrum team ascertaining that all code added to the system is known and fully understood by the team. This generally means that code review will take significant effort as at least to peers are needed to review all additions. (This criteria is related to separation of duties control.)

Source code implementing an issue item (task, story etc.) is marked (tagged) with issue identifier.

This can be achieved in a couple of ways and it is up to the agile team to agree on how they want to make this happen. If the coding conventions followed by the team are very much in favour of this, the relation between source code units and blocks can be maintained in code comments. This should work rather well in a set up where the programming paradigm is very much module oriented and it is followed to the hilt e.g functional programming where the results are achieved with small atomic functions.

A second strategy for achieving this is to establish and follow a clear branching strategy in the version control system, where branch-naming contains e.g. the JIRA identifier of the issue being worked on. In the git-environment this strategy produces a version history where the issue / source code relation remains recorded.

No matter which one of these strategies gets chosen by the agile team, it is crucial that the team commits fully to it and will not accept work violating this as done.

All dependencies brought in by dependency management systems are set to a particular reviewed version, per story

The security in source control should not be limited only to code produced in house, but also to code brought in by dependency management systems such as Maven (Mvn), Node Package Manager (Npm) or Docker base image chain. To ascertain that the whole of the produced software is safe the dependencies must be set accordingly to precisely given versions the content of which is known to us.

The ways this story affects the attack surface of the solution have been evaluated, per story

This means that each and every story should as part of its review be evaluated from this perspective as well. The exact form the criteria takes should be decided and fleshed out for each product/ project separately so that it suits it precisely and can be satisfied with the resources available. Generally speaking the criteria can be satisfied as simply as having a security savvy developer pronounce the evaluation done and telling *how* it passed at the time of code review and if necessary it can be as drastic as having a full blown security expert led analysis on code and deployment carried out for each story. Quite obviously the exact form this takes depends heavily on what kind of a product we're making and how high and detailedly provable level of security we want. When we are aiming for intermediate or highly enhanced security we want the evaluation's results to be recorded preferably in the solution spanning attack surface document.

The rationale of having this as an acceptance criteria instead of a separate user story is in keeping the workload minimal. When the evaluation is done for each story. For further information on attack surface analysis and the relation to this evaluation criteria, we recommend consulting expert articles such as https://www.owasp.org/index.php/Attack_Surface_Analysis_Cheat_Sheet.

How to satisfy in sprint review
It is time for the sprint review and the team has tackled a few user stories. The product under development deals with somewhat sensitive customer data and so the production is done with the enhanced mode of security and privacy.
As a sales rep I want to update the customer information from any of the terminals I can log in to the system from in order to have a flexible access to this key task of mine.

/
ma
nda
tory
on
gov
ern
me
nt
high
h
sec
urit
y
proj
ect
s)
• SH
OU
LD
/
ST
RO
NG
LY
RE
CO
MM
EN
DE
D
in
so
me
cas
es
MA
ND
AT
OR
Y:
As
a
Pro
duc
t
Ow
ner
I
wa
nt
to
hav
e
fuz
z-
test
s
set
up
and
pas
sin
g
for
all
inte
rfac
es
of
my
syst
em,
(
fuz

The team has realised this story to the system by making a new rest end point to their customer data for the patching operation and by adding the UIs for their browser and mobile clients.

First Pat, the product owner gets a demonstration of the customer updates done with browser. This is easiest as it allows all the people taking part to the review to see what is going on and try different scenarios. Some of the stakeholders ask the developers to walk them through further scenarios and state some of their improvement wishes.

Some of the key operations are rechecked from mobile client (Pat having good foresight has organised a camera based overhead projector so everyone can see that as well).

Pat being a thorough kind of a fellow who retains a keen interest to the quality attributes of the code base and architecture asks the team of how their code review went and if there were any key findings there. Next he asks about how the implementing this story has affected the attack surface.

As this is a security intensive project the developers have chosen tools that help them. All API end points are formally specified with swagger and the set-up is such that wrong format in calling will result in server exception. The developers based on this can now tell Pat of the new end point and how it is protected. What parameters it accepts or in other words what new paths it creates to the system.

They demonstrate what happens when it gets called without a proper bearer token and what happens when a real user calls it. They can also tell of the protections they have laid to protect this path. In this case the Swagger's validation regarding the parameters and how the user's role and authorisations are checked based on the token information provided.

If the project is a more documentation oriented the developers just show Pat where in the project wiki's attack surface page they have added this information.

zing should be done at least once / version of interface of the system based on documented risk evaluation / mandatory on government high security projects) MA ND AT OR Y: As a Product Owner I want to see known and listed security

For projects producing applications in web context, OWASP Top-10 checklist?

Applications operating in connection with the networked and especially applications with internet connection have particular vulnerabilities. The most relevant of these are gathered to the OWASP top-10 checklist. The process prescribes that these vulnerabilities are specifically addressed when we are developing web applications in the enhanced security mode.

The exact form of addressing is highly dependant on the application produced, its contexts and to a degree the team making the development work. Thus this section is organised by the vulnerability and each such chapter offers example story and acceptance criteria approaches that we have so far been able to produce. It is not mandatory for all web applications to implement each story or criteria from this section, but it is mandatory to address each of these vulnerabilities to the level prescribed by the security ratings of the software under construction and the endeavour producing it.

All development teams are strongly and warmly encouraged to share such story and criteria patterns that they have come up with but have not yet been added to these examples. The most expedient way would be to add those to the prospects section of this page.

A1:2017 - Injection

Injection can become a problem in any web application where the user input is used in conjunction with either system level commands or database management systems. Due to the frequency of such situation this remains solidly at the head of the top-10.

To address the issue the the injection has to be addressed by the Product Owner either as a story or as an acceptance criteria added to particular stories or at the general level with Definition of Done.

As a typical example of data input we can consider an application where the user files in personal information. This kind of cases can be found e.g in practically all web stores. The user would be typing in information such as names, postal and electronic mail addresses and telephone information. Key in tackling this kind of vulnerabilities by OWASP is not allowing untrusted and unsanitised data to be passed to interpreters such as DBMS or shell or passing them only to parametrised interface where the data can not have a command like meaning.

As a Product Owner one can take a look at the recommended [defences](#) and formulate their stories or constraints accordingly. Here we present examples of formulating these to the agile artefacts.

As a system operator I want user input passed to DBMS by binding them to parametrized queries in order to ascertain that no database commands will be passed from untrusted users to our database

This is a story that guides the developers towards either of the first to recommended defence strategies. A prepared statement or a stored procedure. The choice of exact approach is left to the team and will be to a degree dictated by their strengths and weaknesses. If the PO is opinionated in the approach and wants to guide the architectural choices, this can be handled either by writing general constraints for the product or by remembering to write such acceptance criteria to each of the individual user stories.

As a customer I want to update my customer information by adding a delivery address so that I need but state my customer Id and I have any order I'll make prepared with a delivery address which will make my shopping smooth and nice.

Acceptance Criteria:

- *The persisted customer entity after performing the operation will have a delivery address given as input added to it*
- *The story meets the product's general Definition of Done*
- *In the implementation all of the inputs are passed to database via prepared statements and their typed parameters in order to avoid SQL injection*
- ...

Here the constraint on implementation is explicit. This is likely done by a knowledgeable owner who knows the feasibility of this solution in the case. The constraint is given a rationale so the development team knows the intent and can discuss the alternative approaches if the idea offered seems undesirable from their perspective (e.g They are implementing the system in a language / database connection setup where prepared statements are not available) The PO should be prepared to justify the constraints in these cases as dictating choices to the development team is rarely a good idea in agile development setups.

As said the same could be achieved by a general DoD level constraint. Thus for a project done for e.g. in Java or .Net could have it as part of their DoD like this example shows.

Definition of Done

The user story can be performed in the system

An automated test set has been made for the user story

All previously existing automated test also pass in the branch of this user story

The code has been committed, reviewed and based on the review refactored and is now accepted by the team

All interactions with the database containing user inputs are made using parametrised prepared statements

This kind of a Definition of Done makes certain the constraint is followed in across the scope of the system and makes the job of the PO that much more simple. We recommend this approach - putting the security constraint in the DoD - as the primary candidate for tackling this vulnerability while implementing web based systems with enhanced security.

It should be noted that the above examples only cover one strategy in the defence against SQL injections. If we are following the principle of defence in depth we should implement more strategies than just one. The Owasp site recommends the third strategy, input validation as the go to secondary strategy to implement alongside all other variants. That strategy can again be handled as either a story, story's acceptance criteria or as part of Definition of Done.

y
issu
es
fixe
d
in
a
har
den
ing
the
med
iter
atio
n
(s)
/
sprint
(s)
in
order
to
raise
the
produc
t to
the
desired
level,
(on
ce
/
release
cycle)
• Defi
niti
on
of
Do
ne
crit
eria
(All
Ma
nda
tory)
• All
code
is
revi
ew
ed
by
ano
ther
deve
loper
before
mer
ging
to
the

As an operator of a deployment of this system I want to have all database operations intercepted by input validator in order to make certain that no malicious commands get in to our DBMS.

An observant reader can easily note that this is quite technical story and depending a framework or programming paradigm it can have quite ready interpretation (e.g. Spring's interceptors or a function in the chain of operations on some functional set up such as Node.js) It is good to remember that though roles differing from the simple end user and administrator roles prevalent in the field, other roles such as the operator here can help us adapt the user story format to the needs such as security requirements and thus keep the format of backlog items uniform.

If the PO in question was to put the requirement in criteria constraint form the earlier example we had could take following form

As a customer I want to update my customer information by adding a delivery address so that I need but state my customer Id and I have any order I'll make prepared with a delivery address which will make my shopping smooth and nice.

Acceptance Criteria:

- *The persisted customer entity after performing the operation will have a delivery address given as input added to it*
- *The story meets the product's general Definition of Done*
- *In the implementation all inputs need to be validated to ascertain that they do not contain any SQL commands*
- *In the implementation all of the inputs are passed to database via prepared statements and their typed parameters in order to avoid SQL injection*
- ...

If the PO would be expecting to have several user stories interacting with the database it would again make sense to include the defence strategy as a constraint to DoD, which would mean altering our previous DoD

Definition of Done

The user story can be performed in the system

An automated test set has been made for the user story

All previously existing automated test also pass in the branch of this user story

The code has been committed, reviewed and based on the review refactored and is now accepted by the team

All interactions with the database containing user inputs are made using parametrised prepared statements

All user inputs are validated to ascertain that they do not contain SQL commands

It is worthwhile to note that the new DoD constraint is rather precise. It might seem like a good idea to leave the constraint generic stating only that *All user inputs are validated*. This is a bad idea as it as an ambiguous demand can generate unnecessary extra work to implementation. Consider such examples as postal codes and email addresses. An industrious development team can easily generate quite a heap of extra work when validating that not only is the postal code valid, but that the street address it appears with indeed belongs to that code area. Also, when it comes to emails there have been cases where at the end of the failed sprint it turned out that not only was the developer making certain that the email string was formatted correctly, but that it also existed in the domain indicated. His problem had been in implementing the required mail server logic that queries the domain for this information. Moral of the story being, be careful with what you ask as you indeed might get it.

Though the A1 is typically discussed in the context of SQL Injection there are implementations where the web application will perform system commands and the user input will form parameters to these. In these cases the malicious party will have wholly new possibilities at their hands and the PO needs to address these threats as well. The parametrised interfaces that are available for interaction with database management systems are typically not present, which means that first two of the recommended defence strategies are out of play. It can be expected that unless we are creating a web interface for the operating system these cases are somewhat few and far between so, addressing the injection can perhaps be accepted to be a kind of threat that we defend against individually by only one strategy and add depth from points of defence against A7.

ma
ster
bra
nch

,
per
story

- Source code implementing an issue item (task, story etc.) is marked (tagged) with issue identifier.
- All dependencies brought in by dependency management system are set to a particular reviewed version, per story.
- The ways

The most recommendable way to address this problem is addressing it in the acceptance criteria of a story containing interaction with operating system. However all of the ways of describing a security requirement we saw in conjunction with SQL injection are available here as well.

As a story

As an operator of this system I want to have all interactions bringing input in conjunction with system commands intercepted by input validation that checks that only intended parameters and no malicious commands can be entered in order to protect this system from injection attacks.

As an acceptance criteria of a story

As a customer I want to have my readily rendered work of wonder transferred to destination of my choice via SCP in order to disseminate my brilliance
Acceptance Criteria:

- *The SCP command is correctly created from user inputs and the rendered file is transferred to destination*
- *The story meets the product's general Definition of Done*
- *In the implementation all inputs need to be validated to ascertain that they can not be used to perform arbitrary system commands in our environment*
- ...

As part of a DoD

Definition of Done

The user story can be performed in the system

An automated test set has been made for the user story

All previously existing automated test also pass in the branch of this user story

The code has been committed, reviewed and based on the review refactored and is now accepted by the team

All interactions with the database containing user inputs are made using parametrised prepared statements

All user inputs are validated to ascertain that they do not contain SQL commands

All user inputs are validated to ascertain that they do not contain bash commands

A2:2017 - Broken Authentication and Session Management

There vulnerability as such covers several aspects, but as all of these are fairly straightforward to deal with, we can cover them with a few user stories.

This diversity of the vulnerability means that pretty much all of the stories must be included in order to properly address the problem as proposed by OWASP.

As an operator I want the deployed system to use multi-factor authentication in order to avoid brute force attacks and the attacks based on stolen credentials.

This story should be rather self-explanatory. 2 factor authentication is one of our example stories, so please consult that one for further details, but do note that there are other ways of doing strong authentication as well.

As an operator I do not want to have the deployment set with any default passwords in order to avoid them to be accidentally left in.

This could be enhanced so that not only default passwords, but also default login names should be removed if they were in the system. As a negative user story this can be taken in as a constraint for the development work, which then guides the developers to keep from hard coding the admin login and its password in, but if taken in at later time of development the story can be formatted into "*As an operator or PO I want to have all the default passwords removed from the system...*" kind of format which is then an estimateable task.

this story affects the attack surface of the solution have been evaluated, per story

- For projects producing applications in web context, OWASP Top-10 checklist7
 - A1:2017 - Injection
 - A2: 2017 - Broken Authentication and Session Management
 - A3: 2017 - Sensitive Data Exposure
 - Key management
 - Secure storage
 - A4:2017 - XML External entities (XXE)
 - Avoidance via architectural constraint
 - Mitigating patterns

As an operator I want to have a password quality checker that makes certain that no user will use weak password e.g. such as can be found in the 10 000 worst passwords list in order to protect my users from themselves.

Again, a rather self explanatory story. This can be formulated also so that the "e.g." out making it a one that checks clearly and unambiguously against that set. However formulating it like this leaves more room for the agile team as a whole to come up with the best and most expedient kind of solution for the undertaking they're in.

As an end user of the system I want the system to use such password policy that it allows and encourages me to use strong but easily memorable passwords in order not to make me write down the secrets that I was supposed to keep in my head.

This story serves two purposes. First of all It creates a password policy feature for the system. Password policy is a good thing to have and since situations change, it probably should be updatable as well. If the updatability is seen as an issue by the PO / RMT then the story can be accompanied with the policy updating stories, that further flesh it out. The second feature that this story provides is the formulating of the policy with end user in mind. This follows from our design principles and is a good asset to provide our customers who might still be labouring under the misguided password policy guides from yesterday. With this provision they in turn can serve the actual end users better .

As an operator or admin I want to get an alert when a credentials stuffing or brute force attack against the login is ongoing in order to respond to it.

As an operator I want to have all failed login attempts logged in order to maintain awareness of the security of my entryway

(of course as this is a high security system, it probably has a traceability story such as "As an operator I want to have all successful logins logged in order to maintain access log of the system" in effect as well which means that all login attempts should be logged)

It probably should be noted that the above two stories should be included in to the system's backlog not only based on this vulnerability, but also on the basis of the A10:2017 as they are very much at the core of sufficient / insufficient logging and monitoring as explained there.

As a security conscious operator I want the application to give same response in the wrong user name and wrong password cases of authentication in order to harden the system against user enumeration attacks.

As a security conscious operator I want the application to give same response in every case of the password recovery request weather the user is found or not in order to harden the system against user enumeration attacks.

User enumeration is a kind of attack where a malicious user digs out and verifies user account identities from the system. If the message varies between the user account not found and wrong password it is possible to automatically fork accounts out. If this information is kept from unauthenticated / unauthorized people, the system is that much less penetratable by malicious actors from the internet.

As a security conscious operator I want the application to limit the the amount of consecutive login attempts from same origin in order to harden the system against user enumeration attacks.

OR

As a security conscious operator I want the application to delay consecutive login attempts in increasing time in order to harden the system against user enumeration attacks.

The user enumeration attack is typically automated process and thus it can be resisted by limiting login attempts from same origin. This can be done in either way proposed by our example stories.

The following stories of this vulnerability all deal with session hijacking. Of course these stories could be written as constraints to the stories providing the functionality for some other type of user as seen in these examples.

en
XM
L
can
not
be
avo
ided

- A5:2017 - Broken Access Control
 - First example
 - Second example
 - Third example, objects as resources
 - Further mandatory stories
- A6:2017 - Security Misconfiguration
- A7:2017 - Cross- Site Scripting (XSS)
- A8:2017 - Insecure Deserialization
- A9:2017 - Using Components with Known Vulnerabilities
- A10:2017 - Insufficient Logging & Monitoring.
- Recommended Story and Acceptance Criteria patterns
- Prospect Story and Acceptance Criteria patterns

As a security conscious operator I want the web application to use high entropy individual session tokens for each login session in order to protect my users and system from session hijacking.

If this story was written as an acceptance criteria to an end user story instead of a separate one, the whole could look something like this. The criteria in question is written in cursive.

As any end user of the system I want to login to the system in order to get authentication to its restricted functions.

Acceptance Criteria:

- *Basic DoD*
- ..
- The session tokens used are individual to each of the sessions
- The session tokens are produced with high entropy

The next story sets a session handling constraint for the system.

As a security conscious operator I want the web application to relay session tokens in headers or otherwise protected from leakage in order to protect my users and system from session hijacking.

This too is an example of a companion story. If every aspect of the system would have been expressed as a story or a constraint the story this would be a companion would likely be something like.

As an end user of the system, I want to authenticate myself in later interactions with the system with a session token in order to protect my credentials and to have that much smoother user experience with the system.

This story is an explicit statement of a business need that will result with user being provided the session token. Quite often this need is handled implicitly in conjunction to the login-story instead of an explicit one like this. If the PO used the story structure to create the constraint these two would likely be amalgamated in the sprint planning so that they would be achieved with a single sprint undertaking. If instead of a companion story the PO / RMT wanted to use acceptance criteria format, our previous login story would gain further aspects to its acceptance criteria.

As any end user of the system I want to login to the system in order to get authentication to its restricted functions.

Acceptance Criteria:

- *Basic DoD*
- ..
- *The session tokens used are individual to each of the sessions*
- *The session tokens are produced with high entropy*
- The session tokens are relayed in headers or otherwise protected from leakage

As a security conscious operator I want the session tokens to be securely stored in order to protect my users and system from session hijacking.

As with any protected data so too the session tokens need to be stored in a safe manner in both ends when the implementation is such that the tokens need to be stored. This is a story that deals more with how than what, but as there us a security to be protected with it, we feel that it is acceptable companion story to the login story and fits well with the stories that the login and session management stories would typically break down to. Even if this is a something of a constraint it works better as a story than an acceptance criteria of another one.

As a security conscious operator I want the session tokens to be invalidated after set idle time in order to protect my users and system from session hijacking.

As a security conscious operator I want the session tokens to be invalidated after log out in order to protect my users and system from session hijacking.

As a security conscious operator I want the session tokens to be invalidated after system defined time-out in order to protect my users and system from session hijacking.

The final three stories deal with a session invalidation. These stories work as a good starting point, but the PO / RMT should be prepared to provide further elaboration on them. For example the developers are quite likely to ask how is the "idle time" is defined in the system, a configuration item or administrator maintainable attribute of the system. The message in these three are that the session needs to be invalidateable, and it needs to be that in these three occasions. Two of the stories could be tweaked in to our old login story as acceptance criteria with the log out left as a separate story. The login would then look like something like this.

As any end user of the system I want to login to the system in order to get authentication to its restricted functions.

Acceptance Criteria:

- Basic DoD
- ..
- The session tokens used are individual to each of the sessions
- The session tokens are produced with high entropy
- The session tokens are relayed in headers or otherwise protected from leakage
- The session token is only valid for the period defined as maximum duration in the system configuration
- The session is to be invalidated if the user is idle longer than x minutes

A3:2017 - Sensitive Data Exposure

The sensitive data exposure is a bit of a catch all vulnerability regarding information leaking out of the system. Out of the description we can derive some key vulnerabilities that can be addressed. Dealing with this would in normal cases start from analysing which data our system handles is sensitive enough that it needs to be specially protected. When dealing with a system developed with enhanced security we advice to err towards the side of caution so when in doubt, consider all data to be sensitive. It is advisable to remember that all software at Digia is supposedly designed with privacy first principles which leads in to all data relating to individual persons to be considered extra sensitive.

If the costs of treating all data as sensitive is too much to bear, then it can be balanced by including the periodical story that precedes hardening sprints.

As a product owner I want to have data stored and moved within our system to be listed on its sensitivity and how it is protected in order to avoid sensitive data exposure risks.

This like many other story format approaches on security requirements can be handled as acceptance criteria divided to the functional user stories.

The new data items introduced by the story either to storage or movement have been catalogued by their sensitivity and protection applied to them

As usual this kind of a constraint can be introduced either at story or definition of done level.

When the issues are actually addressed they deal with either moving or storing of data. Simple stories dealing with them would be such as:

As a Product Owner I want all data transmitted between components A and B over non-secured connections to be encrypted with X algorithm in order to ascertain that no sensitive data is exposed in transmission.

Sounds simple, but when written explicitly out creates a verifiable requirement and document that the connections between component A and B are properly secured. When the PO is creating this story it will rather likely start in the form of "...with a secure and trustworthy algorithm..." instead of naming one (of course if the PO has notable security expertise, then the X can already be spelled out, but let us assume that at least half the time the PO still lacks this level of knowledge) and the story gains the level of detailing as the team as a whole refines the story by studying the algorithms and solutions available.

A typical practical application could be a situation where A is client and B is a server application. Realisation of the story might be a certificate ecosystem and secured communication protocol (e.g. https)

Key management

As encryption is just as strong as the keys used in handling it avoiding this vulnerability requires that the proper key management... to be written

Either PKI with at least 4028 key length or Diff-Hellman with...

Secure storage

When sensitive data needs to be stored it needs to be protected as well. This means that the data can not be stored in clear text, but instead in encrypted form. If it is stored in database the database management system the system needs to provide reasonable protections as well. Transitioning from clear text storage to encrypted could be written as following kinds of stories.

As a product owner I want that all information of type X to be stored in a secured database in order to protect that data

As a product owner I want that all X files to be stored as encrypted in order to protect the data in them

A4:2017 - XML External entities (XXE)

The threat from external XML entities arises from the usage of complex XML formats in information exchange where the XML input is automatically processed. As the name suggests the vulnerability follows from the attackers ability to define xml-entities via DTD-definitions or schemas. For a PO / Release Manager wringing their hands over these problems there are few possibilities to deal with the issues.

Avoidance via architectural constraint

The scrum makes an allowance for the PO to define architectural constraints for the product. SAFe has System Architect and the ART architectural constraints for the similar purposes, guiding the development efforts to right direction and keeping them from harms way. These mechanisms form the justification for the PO/ RMT to constrain the development effort from using a particular technology such as XML based information exchange. As the process is relying on the self-directing team the PO / RMT does well to argue their case instead of just dictating it. The first principle in our secure development process is to **keep the system and solutions simple**. XML is not a simple format for data exchange especially when compared to formats such as JSON. XMLs aim was never to be simple format, but instead to be extensible, semantically validateable, human readable data format capable of describing semi-structured data in various ways. Anyone who has had the misfortune of dealing with SOAP-based web services while knowing better things can attest to the inherent complexity of that world.

Thus the simplest way to avoid this vulnerability is to avoid XML altogether and do that by architectural constraint used as part or in addition to the rest of the DoD.

The constraint would typically be the architecture of the interface defining it as JSON. This could be defined in the user story format if the PO wished to employ this

As a PO I want the system's external interfaces to exclude the use of XML and instead communicate in JSON or other more reliable format in order to avoid XXE issues.

It could also appear as a formal criteria of endeavour's Definition of Done

The implementation realises the story

All code is peer reviewed by at least 2 other developers

...

No external interface produced by story processes XML

These are not the most elegant of solutions, but they are a possibility. We can't emphasize enough that agreeing on an architecture with a team and recording this design rationale to that architecture is the preferred way.

Mitigating patterns when XML can not be avoided

For one reason or another it might be impossible for the application to avoid using SOAP or other XML formats entirely. In those cases the risks must be mitigated based on technology. Again architectural constraints are the primary tool wielded by release management.

The additional tools are as follows

As an operator of this security intensive system I want the system to refrain from processing any external XML entities or DTDs in order to protect the system from XXE issues.

By realising this story the development team can fortify the system against the vulnerability and the result will be proven. Techniques for carrying out this task can be found at the [Owas p's XXE prevention cheat sheet](#) .

As an operator or PO of this danger zone facing system I want the system filter and / or sanitise the input in order to protect the system from hostile XML-data, in headers or nodes.

This is a call to implement a normal precautionary measures, but one that is geared and aimed toward this kind of threat. Together these two stories should be sufficient to protect the system against this threat if all known routes XML can enter with are known and accessible to these techniques. When in doubt, the PO can call an enabler / spike or study on the matter with a story such as

As a PO I want this team to develop capability to understand and deal with the XXE threat in order to build our system to be protected against the attack.

There are additional things beyond the stories here that the team can do, but they mostly deal with configuring various tools and protocols. Thus they are best dealt by having the software development / software operation experts to flex their study muscles and find the best possible solutions for this team's problems.

A5:2017 - Broken Access Control

This refers to broken or missing access control of the web application's functionality or resources its servers expose as REST / HATEOAS systems tend to do. A functionality in a web application is typically a view that is shown in response to URL. In that context the attack that follows from the vulnerability is namely an elevation of access rights which can occur in the example situation from users with basic access rights bypassing the UI and calling the URL thus gaining access to a private though unprotected view of the system. When we are considering the objects exposed as REST / HATEOAS resources we can no longer say bypass, but only that the users access the objects which lacked the access controls.

We start the addressing of the problem from a function point of view as quite often the user stories translate themselves into such views and the single page applications somewhat muddle the access issue in some people's minds.

Addressing the issue requires usually the application to have a bit more precisely defined and well thought access management system than just seeing if the user is known to system or not.

First example

The simplest step that we can derive from the user needs is the story

As a system administrator I want each of the http-requests to be accessible only to those users who have been granted access to that functionality in order to have a properly refined access control to my system

This story seems to cover the access management. It should produce a situation where a system maintains knowledge of who exactly is allowed to given functionality and whenever a http-request is made, it is checked if this particular user is entitled or should she receive the 403 forbidden response. This would seem to suggest a companion story

As a system administrator I want to maintain the list of authorised users for each functionality of the system because I want to keep good control of who can do what

This story could be subdivided into checking the list of functionalities, list of users authorised for a given functionality, authorising and de-authorising a user from such functionality, but for the sake of this example we believe the above to illustrate the point.

These stories show a one way of adding functionality level access controls to the system and in a set-up where the user base is relatively small and stable it would probably provide the desired benefits. However in the set-ups where the amount of users is larger or the levels of access they need vary, a more robust solution is required.

Second example

The example here shows how to set-up a role based access rights management to an application which uses http(s) requests as functions. The example makes an assumption that each request responds to one function and more involved operations are achieved by chaining these.

As an operator I want to limit users' access to each function of the system by configurable set of user roles in order to have a sufficient precision and power to control the users as situations require.

The story above should lead the team to enhance the user management by adding a possibility to add users one to n user roles from a set that is not hard coded. The team would likely ask how the role set is to be maintained and if the set is not expected to be too variable a set in configuration file might be an instinctual response. In systems where such configurations can be easily edited this might be a reasonable option and e.g. in Node.js it is at least a good enough initial version. In most Java web application set-ups however the configuration is not significantly better than hard coding the roles into an actual source code file.

To make the maintenance of this role set a fully realised part of the system a companion story should also be implemented.

As a system administrator I want to add and remove user roles in the deployment of a system in order to manage the usage rights of my users

This story should see the system's user role management upgraded to a level where it has actual value to the system administrator kind of end users of the product. We can finalise the user rights management by adding one more story of this kind to the backlog. (If there was nothing in place this would of course require stories like: As an administrator I want to open up a listing of all system users etc.)

As a system administrator I want to be able to assign any of the users any of the roles from the system set in order to control their access rights by the least reasonable privilege principle

With the role management in place we can now turn our attention to the access control of individual functions. In this example the controls are added to the http end points in server each distinguishable by URL. Let us assume that some of them had been made before the PO understood that the access needs to be managed at that level and the system had only the access / no access controls in place. In this case the first story to do is

As a system administrator I want each of the http-requests to be accessible only to those users who have a user role to whom that functionality is authorized to in order to have a properly refined access control to my system

After that story whenever a user makes a request his user roles are checked against the list of roles that are allowed access and if it is not found he is likely given a HTTP 403 forbidden error response. This story sees the creation of the mechanism and if there was a reasonable amount of such end points in place also ascertains that the solution that the development team comes up with is repeatable to all of them.

Depending on the situation the next story suggesting itself might be anyone of the end user stories or the furthering of the role based user management feature set with the following.

As a system administrator I want to be able to see which roles are authorized for a given functionality, in order to know how the functions are protected.

As a system administrator I want to be able to add roles to a list of authorized roles for given functionality in order to manage authorizations for a given functionality.

As a system administrator I want to be able to remove roles from a list of authorized roles of a given functionality in order to manage authorizations for a given functionality.

In order to maintain the level of access control created the following user stories creating new http-requests should from here on include the access controls in acceptance criteria. A very careful PO might formulate this as follows

As a user-role x,y and z I want to create new bars in my foo via POST <https://foo/bars> URL in order to make profit

Acceptance criteria:

- *The DoD is met*
- *Only roles x,y and z can create bars, others are intercepted*
- *If a system administrator removed x from the allowed user roles via standard tools, only y and z could create bars*
- *If a system administrator added Tom to user roles via standard tools, then x, y, z and Tom could create bars*

PO with a good relationship with her team and more relaxed manner would probably formulate this in a shorter but semantically similar form

As a user-role x,y and z I want to create new bars in my foo via POST <https://foo/bars> URL in order to make profit

acceptance criteria:

- *The DoD is met*
- *POST <https://foo/bars> end point is covered by our access control system*

Third example, objects as resources

As the access controls are applied to object level we can actually gain improved functionality in our system. Let us consider a further example.

The system offers a web-interface that all of its clients can utilise. This interface is designed and shaped according to the best practices of the industry so that every single item is available based on URL and the URLs are composed of base path and resource identifiers.

In this system we have users like follows:

<https://api.systemurl.tld/v1/users>

Is a locator for the users collection of the system

<https://api.systemurl.tld/v1/users/1/>

is a locator for the individual user of the system.

The API uses http verbs as actions, so GET retrieves an entity or collection of them, POST creates a new entity, PATCH changes a particular resource partially and DELETE removes an entity from a system. As a sanity check, no collections can be removed from this system. The function level controls mentioned in A7 are already in place for POST, PATCH and DELETE as it was obvious at the time that system altering functions need to be controlled, but as the user entity has gained more detailing exposing them as they are is becoming a liability. The simplest of remedies we could apply to this situation is to apply the same access control we have on three other actions to the GET actions as well. This immediately allows us control on what kind of a user can see which kind of resource entities in our system. This is a solution that will correct the situation for us regarding the vulnerability and is incidentally the correct implementation in the light of the rules that have been set-up in our previous examples.

We could however take things further in our example system as for the anonymous usage we might still wish to list usernames of the users in it (provided we did not consider user enumeration a problem or were not giving out their login but only screen names) , but without any further information like legal names or other personally identifying information.

For the logged in users, we might give visibility to the personal name as well as username and the contact information of the users to whom they have a mutually agreed upon connection. For the user admins, we want to give more information etc. Listing all these rules at all the time can be tedious so we can refine our backlog and DoD with a story and criteria that can produce the entity level control.

As an operator I want to have the entities given out in responses transformed so that only the fields that are allowed for the user role requesting are left in, so that I can have fine grained control on data visibility of my system.

This defines the api implementation in such a manner that every time objects are requested, their contents are filtered based on the requester's role.

Of course this could benefit from the companion story

As an operator's administrator I want to be able to set up object field permissions to the system roles so that I can limit entity data exposure only to those users that actually have a relevant need.

Now we get the administration functionality to our system. A **word of caution**, with this level of control with the function level controls of the system can lead into trouble if the permissions for objects are misaligned with the permissions for the functions. The PO will do well to devise the function and object level user stories in most careful manner.

When the mechanism is in place what is left is remedying DoD with a new acceptance criteria

- ...
- *Entities of the system are served out faceted so that only fields that the requester is entitled to are given*

This Example set of rules would take care of the domain entities of a system created as REST/ HATEOAS system.

Further mandatory stories

Exposing resources to web creates a significant vulnerability to all such end points. To tackle some of those, we wish to add further security considerations to our set.

As a reasonable Operator or responsible PO I want to have all API end points individually rate limited in order to protect the system against malicious access from the internet.

This story gives out two things. First it makes certain that automated attacks get hindered by the rate limiting function. Second, it gives resource based control for the operators and developers of the system to accommodate the different usage scenarios of different resources.

A6:2017 - Security Misconfiguration

Security misconfiguration typically refers to a rather wide assortment of half done or negligent practices that cause the application and its encompassing server environment to expose vulnerabilities towards the networks. These vulnerabilities follow from running and exposing unnecessary services, ports etc, Of leaking out overly much information in the exception handling or of leaving the frameworks' security settings to non-protective development time settings. As the problems and their causes are this diverse, so must the solutions to them differ accordingly.

As a product owner, developer or quality assurance specialist I want the development, testing, QA, staging and production environments to have been put together with equal set-ups and setting-sets so that I don't get surprises when the software moves around.

Explanation tbw

As a product owner I want my product to have a repeatable installation process that guarantees the proper hardening of the installation in order to assure myself and my stakeholders of the environment's security.

or

As a product owner I want my product to have a repeatable installation hardening process that guarantees the proper hardening of the installation in order to assure myself and my stakeholders of the environment's security.

Explanation tbw

As a product owner I want the platform of my application to be installed in the absolutely minimised form so that it doesn't include any unnecessary features, samples, services or code.

Explanation tbw

As a product owner I want my solution to have such a deployment architecture, that different components are securely separated from each others by division to containers/ segments / cloud security groups.

Explanation tbw

As a product owner I want to have an automated settings and configurations checking process with which a deployment and its deployment can be checked, verified and validated in order to protect my system from OWASP top-10 A6 - Security misconfiguration vulnerabilities.

Explanation tbw

A7:2017 - Cross- Site Scripting (XSS)

Cross-site Scripting is deplorably common ailment. The attacks of this kind work by having attack material from outside sources embedded into the rendition of a web page typically by slipping the end user a link which makes a http-request into which the attack is included.

In order to avoid the vulnerability *all data* that is going to be rendered needs to be classified either as *trusted* or *untrusted*. The trusted data is such that it can not originate from anywhere but the servers controlled by the operator of the application. Everything else is untrusted. Thus in the work following the enhanced security all stories that produce user interactions via URLs need to an additional criteria in for their DoD.

All data rendered in this view is known for its source and channel and based on these classified either as safe or unsafe.

To use a crude example a URL-parameter is a good example of untrusted data. In enhanced security solution we should seek to avoid untrusted data as much as we can. The cases for using it are usually related to performance as even chatty APIs where lot of requests are made via AJAX/ underlying REST requests can prove burdensome. However these costs should be carefully considered when we aim for secure solutions.

There is a temptation to give a prescription that the acceptance criteria should also contain *n o untrusted data is to be utilised on page*. However as we can not know all the situations which will be met in our projects, the prescribed format is given in significantly lighter form.

The use of untrusted data on the browser interface is minimised to absolute minimum in the story's implementation

This means that in order to deal with the untrusted data we still need to do something more. [https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet) provides good rules and techniques for doing it, but we need to be able to express these as agile requirements ie. stories or constraints. The basic idea from the referred guidance is to make sure that the untrusted data is rendered to the user as data and never ever executed as code. This idea fortunately lends itself fairly well into an acceptance criteria.

All untrusted data is to be dealt with and escaped to be rendered as data and not executed as code.

This is a bit of a cop out however as even though it records the constraint to DoD that should produce the desired outcome, it leaves a lot of work for the development team. So if at all possible the need that is expressed with this constraint should not be left only there, but in co-operation with the developers, be refined into further stories that produce the means of satisfying the need and meeting the constraint.

Let us use as an example a single page application which for one reason or another has to use untrusted URLs showing them in iframe-elements in some stories and in the anchors' href-attributes in the context of other stories. We know that validating URLs is a good recipe for protecting our users. In this context we might wish to add a story like

As a developer or product owner I want to have an URL-validator with which I can validate any untrusted URL we are forced to use on browser side so that the end users are not exposed to XSS and that I don't have to rewrite the validation for every bloody time we need it.

Other such similar stories might include:

I want to have a html-encoder that changes any given string replacing element making characters with html-entities.

Which protects element-content from changing execution context or slipped in executable code.

A8:2017 - Insecure Deserialization

As the name suggests this is a vulnerability that follows from accepting serialized objects from sources where a hostile party can tamper with them so that either their data content or even behaviour might change from the intended. Worst case examples are such where the attacker may for example achieve remote code execution on the servers. The somewhat less severe cases see the contents of objects transformed so as to provide e.g. elevation of privileges.

For the concerned PO the primary solution for the problem is to **constrain** the development into such approaches that do not require the application to accept serialized objects from any but the most trusted of sources, only when that is not possible should other solutions to be considered. In those situation the following shall be used.

Of these one should note on the deserialisation all that which is said of the input handling (injection) and of logging (A10:2017). Serialized objects are input same as anything else, except that they are even more prone to be utilized as attack vectors by malicious agents of the internet. So avoiding their exploitability as sources of injection is very important. A10 gives us a good understanding of how failures that occur in validity checks and sanitation processes ought to be logged and monitored in order to maintain vigilance regarding impending and starting attacks. Again, all of that holds true when it comes to dealing with serialized and deserialisable objects.

As a PO / Operator of the application, I want to have all deserialisable objects to be integrity checked to ascertain that they have not been tampered in order to add to the system's protections against attacks.

The examples that the OWASP gives for this kind of hardening against the vulnerability are calculated checksums, but security capable developers can relatively easily come up with stronger approaches. Security specialists consulted in writing this guidance gave signed (PKI-interface) and encrypted payloads as their go to approach for solving the problem as that approach rather easily encapsulates the deserialised objects not only untamperable, but also invisible to unauthorized eyes.

Further security may be sought by the following stories

As a PO / Operator of the system I want the deserialised objects to be run only in separate low privilege containers / context to avoid exposure of the more vulnerable components of my system.

Explanation TBW

As an operator, PO, security officer of the system, I want no network traffic from containers that deserialise objects in order to further curb the dangers imposed to the system by playing with deserialised objects (OWASP Top-10 A8:2017).

or

As an operator, PO, security officer of the system, I want to strictly limit network traffic from containers that deserialise objects in order to further curb the dangers imposed to the system by playing with deserialised objects (OWASP Top-10 A8:2017).

Explanation TBW

As an operator, PO or developer of a security sensitive system I want to enforce strict type constraints before object creation in order to limit my exposure to code injection when deserializing objects

This is a mitigating feature that can help, but the security community is aware of bypasses to the technique.

The above are all the specific stories with which the insecure deserialisation can be mitigated. As said A1:2017 and A10:2017 should be carefully cross referenced with this one when the deserialisation is utilised in a security project, but we recommend to avoid object deserialisation from untrusted (or finitely trusted) sources as much as humanly possible.

A9:2017 - Using Components with Known Vulnerabilities

As it is fairly common that we use software components by the open source community or third party vendors the configuration management grows crucial in the secure process.

To achieve this the first step for a clever product owner is to have the following story in the backlog

As a Product Owner I want all the third party components and their versions to be listed to the product documentation in maintainable form in order to install dependency management for my product.

If this story is timed at the very beginning of the development efforts, it will initialise the product documentation in the desired format. If a canny product owner times it not at the very beginning but at some of the early sprints it will provide an easy learning experience for the team on what it means to delay this documentation work to be done as a bigger batch instead of continuous and manageable work. In case of greener teams this can be a valuable addition and in case of more experienced teams it could allow them a spot to shine in as they very well could have done that set of documentation as a part of they work without special requests from the PO.

This story is however a one time effort and if the matter is left at that the level of documentation on configuration will deprecate and grow stale. Thus the product's DoD should be augmented so that the configuration documentation of the components is maintained regularly at reasonable cost.

- *If a story introduces a new third party component to the system it is brought in with a specific version*
- *The version with which such a component is brought into the configuration in such a fashion that its version can not change without intentional action by the system managers*
- *All components that the story introduces to the system are documented to the configuration with their canonical unambiguous name and the version at which they are brought in*

These criteria make certain that future stories include the maintenance of configuration management documentation. The agile manifesto states that we value working software more than extensive documentation, but when we are creating secure software the documentation produced here becomes crucial. The documentation can grow rather extensively e.g. in the case of Node Package Manager modules or Java implementations with extensive and transitional dependencies. Thus choosing such components whose vulnerabilities are traceable and manageable quickly grows in priority. The agile team thus needs to start pondering this aspect when picking components to utilise in their efforts.

To address the need of checking this vulnerability the configuration documentation needs to be enhanced with vulnerability information, what vulnerabilities exist and how the components are tracked.

As a Product Owner I want the configuration management enhanced with the vulnerability check data of each component, marking the time of check in order to ascertain that the vulnerabilities in dependencies are addressed at the earliest.

and

As a product Owner I want the vulnerabilities of third party components checked and the configuration document updated in in order to ascertain that we are not using vulnerable components. (Repeated periodically, at least once before hardening of planned release)

With the last story repeated periodically in the backlog. With these stories present we now have a set-up in our product management for making certain that no known vulnerabilities mar our product.

A10:2017 - Insufficient Logging & Monitoring.

In enhanced security context the traceability of actions is in a key role and thus all actions should by default be logged well and with good user context as can be seen in the traceability acceptance criteria examples. The logging in this context is just the first half of the issue and is understood as the data collection part for the solution of intruder prevention. The reasoning behind this is that most of the successful attacks begin by probing for vulnerabilities in the target system. If the probing is allowed to proceed without intervention the likelihood of the attack's success increases by the full 100%.

For the industrious PO / RMT to tackle this issue, they should consider following approaches and stories.

At the first backlog security refinement the the logging as a feature is added to the product backlog either as a feature or an epic as suits the product or project under development. The logging feature should be a centralised asset in the product and likely in a larger context as well. The full details of the logging to various contexts ought to be fleshed out as user stories like As a developer I want to have a logging feature which I can call in order to log significant events from the execution of my program and so forth.

Specifically, all **login**, **access control** and **input handling** (i.e. input validating) stories ought to have sufficient logging and the first approach to achieve this is to enhance their acceptance criteria with logging features in a following manner

As a user I want the sensitive actions available to me, only be available after my claim for access rights is verified by login procedures in order to guard me and mine against malicious users.

Acceptance criteria:

The user is recognised against their credentials and after that able to access those features of the system to which he is authorized.

...

In a situation where the login attempt fails, the failure is logged to the log management of the system with context identifying the user, time and what went wrong.

This is an easiest kind of way to remember that the user interaction needs to satisfy the logging need as well as the basic user interaction. It could be equally well be written as a companion story. It might also be the easiest one for the PO to handle backlog's security refinement as then it would be clearly in. In such a case the item could look like this

As a PO or an operator seeking to guarantee the security I want all failed login attempts to be logged to the system's log management feature in order to keep the forensic data about possible threats.

This kind of a story is a clear feature in its own right and thus easier for the team to understand and PO to see done. It is also possible that when a story is initially written with the acceptance criteria, that criteria is broken away to a story like this in order to make the story more manageable.

As said this is needed for all stories that deal with accessing a privileged resources or deal with input from untrusted sources. If the assets guarded by access rights management in question are especially valuable (i.e. most assets in the context of the secure development process produced software), the logging solution needs to secure audit trail maintenance, this would likely be best expressed by acceptance criteria.

As a user I want the sensitive actions available to me, only be available after my claim for access rights is verified by login procedures in order to guard me and mine against malicious users.

Acceptance criteria:

The user is recognised against their credentials and after that able to access those features of the system to which he is authorized.

In a situation where the login attempt fails, the failure is logged to the log management of the system with context identifying the user, time and what went wrong.

...

The logging is done to such place and in such a way that the untampered log shall be retained in the system.

The example implementation of such a log could be a separate service that offers only an append action and no transform capabilities for this system and takes care of the audit trail for this system. A different approach that can be implemented by the system itself would be an append only database table.

The other half of this issue after the logging is taken care of is the monitoring of the logs provided. Monitoring should be rigged to sniff out attacks and probes that precede a full scale attacks. When such an attack is spotted alerts should be made immediately. Monitoring is obviously a feature or even a separate product that is made up of several user stories.

The feature itself shall be added as an epic or feature to the backlog of any web application that is developed. The stories that make up monitoring can be fleshed in later with the goal that they should provide a timely intervention to any ongoing or looming threat. The monitoring can be done as a feature of the product or as utilisation of an external product which ever is the better way for this endeavour.

Recommended Story and Acceptance Criteria patterns

Prospect Story and Acceptance Criteria patterns