How to adapt information security in agile development?

LOCALTAPIOLA

How to adapt information security in agile development?

Foreword

How do we create software that does its job and is also secure? Not the way we used to.

Old software development paradigms were useful in their time, but they did not consider the situation where production software is always connected and exposed to not just friendly use but also adversarial attacks. They did not consider extensive use of third-party software. We need new paradigms and methods to produce software that are both innovative and secure. Without security as a primary goal, software and hence our business will not be trustworthy. Without trust, no business.

Fortunately, there is something that stimulates innovation and at the same time enhances security: **speed of action**. When we shorten the software development cycles, we can experiment more and learn faster. Iteration results in more innovation. With faster and ultimately continuous software integration and deployment, we can fix security vulnerabilities much faster. This results in better security, i.e. a lower risk of data breach.

A second thing that improves both innovation and security is **information sharing**. Software code has not one but two tasks: To tell a computer what to do, and to communicate ideas between developers. By sharing ideas and code openly, new ideas will emerge sooner. Flaws will also be detected sooner. When they are fixed, security improves. Teams that openly and immediately share information, insights and ideas with each other outperform all other teams.

Today no software is developed in isolation. Every piece of code takes inspiration from some previous piece of code. Every application uses **third-party** frameworks and libraries. The bill of material of a software application can get extremely complicated. Our principles and rules for how to choose a library and when to update it are as important as the code we write.

Software is the expression of human intent. What goes on in our brains dictates what the software code will look like. **Coding culture and design principles** have dramatic effect on the outcome. The best software development teams (and collections of teams) spend time defining their culture, emphasizing priorities such as security, robustness and elegance of code. As a result, their every line of code, from the moment it is first written, lives up to a higher standard than if it were haphazardly thrown into the CI/CD framework.

Software engineering has become an exceedingly collaborative practice. In the greatest applications, software collaborates well with other software, and engineers collaborate well with other engineers. It is the **hacking mindset**: figure out things, share your findings, build something cool. We are open to input from others. Quality and security is everyone's duty.

Mårten Mickos
CEO, HackerOne

LocalTapiola - The lifelong security company

Our mission is to help our customers secure their lives and businesses. We tailor the security, financial and health products and services included in lifelong security to suit our customers.

LocalTapiola will accelerate its renewal into a lifelong security company. In particular, renewal means a shift from traditional reactive action to proactive, individual and preventive promotion of our customers' lifelong security.

We will continue to develop our customer experience, with the aim of distinguishing ourselves by being genuinely caring, personal and rewarding. Customers get lifelong security solutions that are tailored to their individual needs and protect throughout life. In the strategy period, we aim for a significant increase in the number of customers whose lifelong security level is optimal in all aspects: security, health, finances (3T customer relationship).

In addition to easy access and damage and emergency services, we will introduce proactive services to help customers reduce damage, improve financial success and promote health. For LocalTapiola, increasing services creates a competitive advantage in the insurance business as well as opportunities to influence customer paths and profitability.

The core of our strategy is to provide our customers with tailored lifelong security solutions and related services. Our success is based on customer-oriented service leadership, a regional company structure, a professional and passionate culture of lifelong security, and the utilisation of knowledge to improve our customers' lifelong security and develop our business.

Introduction

The document you are reading showcases a high-level approach for including security in agile development, with handpicked examples of supplementary documentation mostly consisting of generic application security guidance. The goal of this publication is to offer a view into a living, real-world implementation.

The sections and chapters below are a sample of created instructions, with minor modifications to remove organization-specific details.

Information Security in Agile development and why do we write about it?

Software developers can be highly skilled and still forget about security. Why is that? Even though we can read about security breaches almost every day, in projects many times security is still something that is neither budgeted, in scope, nor reasonably resourced. Understanding is often lacking, as is guidance - security is seen as something that is done just before going into production. The final hurdle to be crossed. LocalTapiola works towards a better world where security is holistically considered throughout the project and its life cycle, in the budget and right from the beginning. It requires security knowledge in the organization and sometimes hard work to implement. Benefits on the other

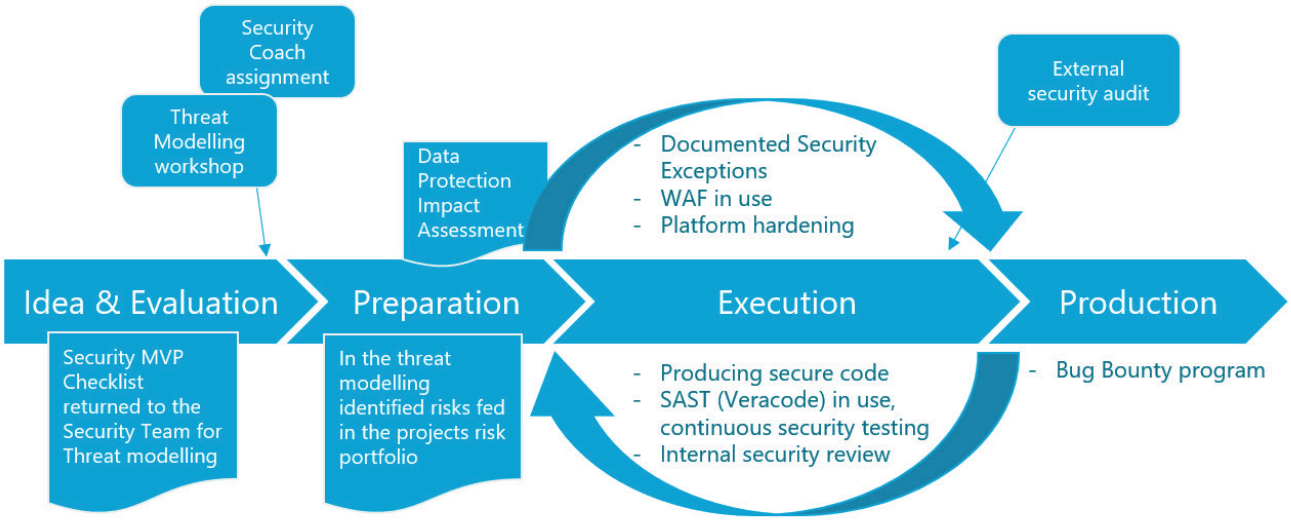
hand are better cost forecast, and achieving production installation schedules, instead of missing them due to security issues. The aim is to have a good user experience with transparent security, no hacks nor quick fixes and above all secure and reliable systems that both customers as well as LocalTapiolas personnel can trust.

This leaflet is not the whole truth. By no means does it cover all activities we do around security. Its purpose is to give you some insight into how we do it at LocalTapiola and the things we have found important to specifically raise.

Have you thought about security in the projects you are working with?

Related processes

This section introduces some non-technical ways to achieve a better security outcome in the project. The purpose is not to cover any specific project methodology but rather to point out some best practices that can be used in a variety of ways and implemented into different development models.



SecDevOps, an iterative way of doing

SecDevOps includes security inside in the development and operations processes. In a SecDevOps approach, secure operating models are integrated into the DevOps process. SecDevOps promotes flexible cooperation between development (Dev), information security (Sec) and operations (Ops) teams. The primary goal of SecDevOps is to narrow and ultimately close down the potential gap between software development and information security, while ensuring prompt delivery and implementation of the code in a secure manner.

Security and project roles

It is important to understand that security is not only the security departments or security teams responsibility. Security should take place throughout the application/ software development life cycle in the project. Many roles in the life cycle should be aware of the basic security and privacy requirements in the industry, especially if it is a regulated one. For example, how to handle customer data (privacy), when a strong identification should have been used? What information and which events can be logged and which must be logged. The list is long. To support project security needs, it is good to have a specific role - a Security Champion is one of the project roles that must be considered.

A Security Champion is defined in OWASP (https://www.owasp.org/index.php/Security_Champions) as "active members of a team that may help to make decisions about when to engage the Security Team". You can also call them Security Coaches as they should work like Agile Coaches to help the development teams and the projects to hit security goals with a good spirit. Who then can become a Security Champion in the organization? The Security Champion/ Coach should have an interest in information security but not necessary be a security expert. The role requires the ability to run and maintain security related stories or requirements in the sprints, identify possible problems, support the team and coordinate security tasks with in-depth security experts if needed.

Preparing for the implementation - threat analysis

Before commencing a threat modelling workshop, a short checklist should be filled out by the stakeholders. The checklist is used to gather some basic information about the application, like

- Who will be using the application?
- What is the application business purpose?
- What data is processed in theapplication?
- Where will it be deployed?

The information gathered in the checklist is used by the security team to better understand what topics must be covered during the threat modelling workshop. Each organization should define its own checklist, as its contents are highly dependent on context, industry and business criticality. The checklist also assists the different owner roles and stakeholders to understand what may or may not affect security decisions in the project.

Threat modelling

Threat modelling can be conducted either using a formal, process-oriented methodology or more informal, discussion-based approach. Understand what is needed for your development target, and choose accordingly. However you decide to threat model your target, it is recommended to break it down to manageable chunks and start from the actual intended business process.

A more formal process, focusing on complete diagrams and documentation are suitable for environments where changes or updates are costly, regulatory documentation requirements exist or safety is paramount. Depending on your situation, early stage architectural choices or a completely new system may benefit from this kind of a process. For less seasoned professionals, it can also be easier to follow a predefined set of steps.

Downsides can include large amounts of data flow diagrams and documentation, which are more or less identical, with the same risks, threats and technology choices. Formal approach can also be heavy on the project teams, disincentivizing participation and demand-based scheduling. The amount of documentation increases the required resources for each session.

Informal and more casual approach works well in an environment where technology choices are homogeneous, architecture defines security controls, and risks & threats are well understood. For seasoned professionals a friendly chat over an architecture diagram often reveals the obvious pitfalls, especially if the target is limited in size. It is recommended to produce a memo from the session, identifying the findings and potential weaknesses for the development/project team to take into consideration during development. Less process and diagram-focused discussion session moves the burden on the subject matter experts, making attendance more light-weight for the development/project team members.

Downsides can include missed weaknesses and vulnerabilities, heterogeneous coverage and difficult-to-understand logic when a seasoned subject matter experts use their experience to cherry pick issues. If junior security people, or others interested in learning, are present it is useful to open up the thinking and thought processes.

Managing information security and privacy requirements in a project

Threat modelling is responsible for producing a set of security non-functional requirements (NFRs) for the project. For the sake of efficiency, a predefined set of NFRs should be provided to the project from which the project can pick and choose any and all relevant items. The predefined list is usually maintained by the security team. The NFRs aim to cover as many different security aspects as possible. However, if threat modelling uncovers anything that may not be already covered by the predefined NFRs, the project should add customized NFRs per their own need.
The NFRs contain items and topics about

- Processing and managing PII
- Injections and user input
- Authentication, authorization and access control
- Secure architecture, hardening and configuration
- Logging
- Various technical security issues

Execution - measuring security progress and keeping track of activities

Once threat modelling is done and the project has its NFRs, it is time to maintain the security stance throughout the project, all the way into running the application in pro-

duction. To do this, some simple metrics and checkpoints should be involved to make sure security is not forgotten or overlooked. These metrics should be chosen in such a way that it is easy enough for the project or the security team to gather the data manually or partially automate it. The actual metrics must depend on the organization or business, but there are some generic items that can be used for almost all application development. Metrics - especially checkpoints - should be put on a timeline or attached to certain phases of the project model.

By having checkpoints and metrics available for the project from the start, security is not a running target.

Potentially usable metrics and checkpoints may include

- The initial pre-threat modelling checklist has been completed and the results of the threat analysis are available to the project
- Data protection impact assesment (DPIA) has been done
- The role of the Security Champion (or similar) has been assigned and the Security Champion is active in the project
- Security best practices have been identified and chosen (there should be a predefined library of best practices that is maintained across all projects)
- Security NFRs are progressing in each sprint or phase (this must be a continuous metric)
- Data flows have been documented
- The architecture has been documented including security boundaries
- Security decisions are documented - accepted risk is noted as well as any mitigations
- Security exceptions are approved and documented
- Static security scanning is done regularly (SAST and SCA)

Security testing covers many things including penetration testing. Penetration testing can be used for additional metrics and checkpoints. However, penetration testing as a single security checkpoint - at the end of a project - is not recommended as it does not encourage a built-in holistic security-by-design approach in the development life cycle.

Production - handover of security to operations

Every project targets production. While that means the project is over it does not mean that security is over. The responsibility of handing over security from development to operations is not always straightforward - and as the organization grows bigger and things become more complicated, so does the handover and responsibility for security.

Bug bounty programs

Bug bounty programs are increasingly popular and also maturing well. Running a successful bug bounty program is not straightforward and requires good underlying processes - and patience. As new services and applications are rolled out, they are taken into the bug bounty program by default.

The LocalTapiola bug bounty program has been very successful in finding complicated flaws and has proven very valuable in the post-GDPR world in finding flaws before they become issues. The program has rewarded nearly 300 reports and paid out over 100 000€ in bounties to date. Findings from off the shelf applications and software has been duly reported to authorities for responsible communication and disclosure (when applicable).

Bug bounty is a program into which hackers can participate for a chance at a bounty reward. Bug bounty programs are organized by companies from all industries. The program can be open, private or time-bound. The goal of a bug bounty is to have the good guys find bugs in the target application or system before the bad guys do - and get paid for it. The more valuable the target, the bigger are also the rewards.

Web Application Firewall

While we aim to have perfectly flawless software in production that is never the case in reality. In addition, the complexities, dependencies and even peculiarities of the underlying systems and integrations means that the word "quick" in quick fix can mean days or weeks. Hence we need another tool to be able to mitigate issues when no other options persist. While a WAF is no silver bullet and by no means an excuse for bad code, it does provide a means for quick (as in minutes or hours, not days or weeks) fixes.

Each application requires a separate policy - the set of rules that are applied to the web application firewall. The rulesets may contain everything from disallowed HTTP response codes to context-sensitive regex pattern matching for specific parameters. Maintaining these rulesets is tedious work. The WAF is many times used in conjunction with the bug bounty program. When a flaw is discovered, a quick fix is done with the WAF (while a long-term fix is planned on the backlog). For off-the-shelf software the challenge is that there may not even be a fix or that the manufacturer does not "acknowledge" the issue in the first place. In these cases the only way to mitigate is to plan and design rules.

WAF stands for Web Application Firewall. There is a clear difference between a "normal" firewall and a WAF. The WAF inspects the actual contents of http/https traffic for attack patterns, whereas a normal firewall is only concerned about ports being open or closed.

The WAF looks at every parameter, every request, every path, every cookie and every header - everything. The WAF does not automatically know how the application is supposed to work (hence the need for rulesets). Out-of-the-box it makes assumptions. Usually these assumptions are 90% correct, but the remaining 10% might make the application unusable. For this reason, the developers must partake in defining the ruleset. It is important to understand

- the most critical functionality of the application
- anything that is custom and outside of what is considered "normal"
- specific cookies and headers that are used
- parameters and values outside of normal US-ASCII character set
- ...and a whole lot more

Technical guidelines

The instructions are relatively middle of the road, suitable for wider consumption. Some of the topics you encounter in the daily life of software security depend heavily on the used frameworks and architecture components. For those, it makes the most sense to create technology specific guidance, which is kept up-to-date. After all, if there are specific tools and architecture components with which to solve the problems, why not give detailed instructions, instead of forcing everyone to find their own solutions to the same problems. One approach to creating your own instructions is to look at your environment, current processes, used technology stack and ensure all layers have been covered.

Creating secure coding practices

There are different approaches with which to tackle this topic. Here we introduce two commonly used viewpoints - top n lists and positive practices. Taking focus to eliminate a handpicked list of vulnerabilities typical for your own organization is a common reason for using a top n list. Creating a set of positive practices, which need to be followed, approaches the same problem from another direction. Both viewpoints have their pros and cons. This chapter attempts to give you guidance on how to create your own - and describe what can go wrong even with good intentions. By understanding the difference between the two, you can start finding the right way to implement secure development methods in your organization. One size rarely fits all.

Top n list

By creating a list of vulnerabilities, even if based on real world data, there is a risk that this criteria is not updated over time as environment and technology evolves. An outdated list looks in the past, instead of focusing on software developed for tomorrow. Unless communicated properly, it can become the lowest level of security - top n lists can be good awareness tools for promoting application security, but rarely work as such for properly securing an application. Depending on the development target, parts of the list can be irrelevant, watering down the message. The descriptions should contain information on how to avoid the issues, not just describing issues and their impact.

Vulnerability focused approach is good for trying to eradicate complete vulnerability classes, especially if you cannot tackle them with other controls. This should be complemented with technology specific best practices, and ensuring these do not become the only security criteria - rather a litmus test for software quality. The approach should be a combination of relevant historical data from identified vulnerabilities (internal reviews, assessments, bug bounty or other external reports), and handpicked technology specific pitfalls. If you have suitable intelligence available on current trends or industry views, use this to enrichen the list.

Spend effort to make sure this does not become your security guide, and it is regularly updated.

Positive practices

In theory, focusing on the good practices and software development methods yields positive outcomes. The problem with high level concepts is that interpreting their meaning to

the development target in question requires seniority and understanding of secure development. Often the people who can best interpret them are the same who are already practicing secure development practices, and have no need for generic guidance. For juniors, learning is often easiest with concrete examples. Having said that, by focusing on practices which cover a lot of ground, when implemented properly, can yield positive outcomes. For example, strict and well-defined input validation easily tackles a large part of the attack surface.

The practices must have a good match with the technology, development environment and languages. Try to categorize your practices to make it easier for the reader to approach the topic - for example language, target environment (web, embedded, mobile, mainframe, etc.), technology, and product. Finding the right balance between giving high level guidance versus detailed implementation specific practices or secure patterns depends on your ecosystem and partner network. Well-designed high-level practices and patterns age well whereas technology specific instructions may require frequent updating. Hopefully your partners have the best subject matter expertise, and are comfortable creating and updating their own developer guides to complement existing documentation.

Be sure to regularly review the effectiveness of the practices.

Secure mobile development

Secure and privacy aware mobile development requires one to understand the intricacies of chosen platforms, in addition to having a holistic view of the overall security and privacy needs. This chapter focuses on platform specific areas, where care should be taken. Instead of viewing this guidance as a comprehensive security requirements documentation, the reader is advised to familiarize themselves with other additional security materials and platform best practices.

Purpose of this chapter is to focus on areas previously recognized as having a potential impact in mobile development. As your use cases may differ from the underlying assumptions, the security and privacy themes addressing your needs might differ as well. If you identify topics or needs not covered by the chapter, supplement it with additional information or create your own guidance matching those needs.

Architecture areas

General quick tips

- Follow technology stack best practices
- Use threat and risk modelling to identify solution specific issues and areas of interest
- Make sure security requirements are included in NFRs, acceptance criteria and stories
- Minimize attack surface by defining a strict set of features and input, and disabling unnecessary functionalities
- Create simple solutions, avoid complexity. Simple is easy to review and understand.
- Harden the configuration of chosen technology components. Investigate what are the applicable vendor or industry best practices.

Implementation focus areas

- Review threat modelling findings, or hold a new threat modelling session if a solution is undergoing major changes which may affect privacy or security
- Focus on authentication and authorization
- Validate all inputs
- Understand what data is stored locally by the application
    - Areas of interest
        - Intentionally stored data
        - Cached data
            - What information is cached, where, for how long, are the caches cleared?
            - Note – web view caches data, unless otherwise instructed
    - How sensitive is this information?
    - How is it protected?
        - How does the application protect locally stored information?
        - How does the application prevent access to live services?
        - Utilize keychain whenever information needs to be protected
- Do not write sensitive data to client-side logs (preferably disable client-side logging in production builds)
- Use password input field when requesting user to enter PIN or password
- Protect network connections
    - Enforce the use of TLS
    - Pin certificates for organization-controlled resources
- Do your homework on any third-party libraries

Threat scenarios

The developer must assume the following when developing mobile clients

Attacking the backend

- The attacker has root access on the device, can view the file system & debug or modify applications
- The attacker can view and modify application traffic

Attacking the user

- The mobile phone contains a malicious application, which does not have root access
- The end user will eventually lose their device.

Assumptions for backend APIs

Attacking the backend

- The attacker can modify the API requests sent by the client

Perform a risk analysis for the above scenarios and mitigate as required.

Android

Available documentation

- Android security best practices: https://developer.android.com/topic/security/best-practices

Special considerations

- Do not store sensitive data on external storage or in SharedPreferences
- By default, set android:exported to false for all components
- For more information about certificate pinning, see documentation on Network Security Config or OkHttp CertificatePinner
- If application handles special categories of personal data, use FLAG_SECURE in WindowManager.LayoutParams
- When sensitive information is entered by the user, use textNoSuggestions or textVisiblePassword (note that this disables gestures typing as well)

iOS

Available documentation

- Developer documentation: https://developer.apple.com/documentation/security
- https://www.ncsc.gov.uk/collection/application-development/apple-ios-application-development/secure-ios-application-development

Special considerations

- Do not store sensitive data in UserDefaults
- For more information about certificate pinning, see documentation on AlamoFire ServerTrustPolicy
- Hide sensitive information from the UI when applicationDidEnterBackground is called (either remove the sensitive information or replace screen with LocalTapiola logo)
- Use URLSession Task (:dataTask:willCacheResponse) to control caching of sensitive data
- When sensitive information is entered by the user, set UITextField autocorrectionType property to UITextAutocorrectionTypeNo

Backend APIs

Special considerations

- Verify that user has been authenticated
- Perform authorization checks for incoming requests
  - Store security relevant data in the session on the server side, not on the client side
  - If the client needs to choose between different items (say watch in a watch winder), do not refer to the serial number directly - store this information in session, and use indices on client side
    - that, is instead of having parameters like `{ "watch": "<serial number here>" }`, use parameters like `{ "watch" : 2 }`
    - this practice minimizes the available attack surface and helps avoid authorization vulnerabilities
- Validate all inputs
  - Read and use only the expected parameters
  - Verify type, format and length of received input, whenever possible
- Decide explicitly on what fields need to be returned from backend requests to the client
  - Avoid returning all received data, unless it is actually used by the application
  - Avoid unnecessarily returning sensitive values, such as serial numbers
- Minimize technical error message contents
- Use Cache-Control headers to advise client on whether to cache results (instruct not to cache sensitive information)
- Design logging to preserve end user privacy (e.g. do not unnecessarily log sensitive information) while preserving a sufficient audit trail
  - Distinguish between error logging used for resolving technical issues (no sensitive information present, many people have access) and detailed audit trail (contains sensitive information, only a handful of people have access)
  - Follow organization logging practices
  - Avoid debug logging in production environments

Logging

Although some companies operate under tight regulation and a regulated industry does add requirements to logging, logging should be a part of all software. Non-repudiation is the knowledge that someone (customers, personnel) cannot deny the validity (the fact that something happened or occurred) of something. In practice, proving this requires thorough logging. Whenever there is a user that does something with data, non-repudiation becomes a valid case. Cases for non-repudiation may include

- A customer has submitted information that he/she later denies knowledge of
- A customer has changed account numbers
- Personnel has changed customer information
- Admin has accessed data

Logging actions and activities is the key to non-repudiation. Investigations may reach many years into the past. Depending on what information or data the application is pro-

cessing or making available, different laws may also apply. Under the law, forensic investigations may be relevant. Investigations may reach many years into the past. Logging - and managing logs - is key for success.

What to log

Logging is part of application's accessibility, information availability and backup. In such cases where traceability is required, logging is the answer. Logging must be implemented systematically throughout the codebase.

The log source is the system that generates information that is logged. At a minimum, it must generate logs on usage, errors and security exceptions. Logs must be written for at least the following situations:

- Security exceptions and anomalies
- Errors and operational exceptions
- Accessing or processing personal information (but not necessarily logging the information itself)
- Administrative tasks within the application

Log events are the actual events that generate logs.

- Login / logout
- Adding, deleting or changing data using the application UI
- By default, logs shall not contain critical information such as personal data of individuals. If storing personal data is required, log must be treated as an audit log.

Operational logging is what we know as "normal" logging - logs that are used to solve problems in production, trace events from the past and get an overall understanding on how the application is working. This log data is usually used both in normal as well as exceptional situations. Operational logging is usually very technical and may contain instrumentational data.

Operational logging does not and should not contain debug information in abundance. It is not a place where everything is dumped.

Audit trail logs contain information about transactions - who, what, when and why. Audit trails are used to prove non-repudiation, usage statistics and forensic information related to application usage.

Audit trail logs MAY contain PII data although PII data should be masked if logged regularly. Audit trail logs may not be accessible by anyone - access to audit trail logs must be restricted.

How to log

The events must be consistently formatted when written to logs. Logs should be at least partially human readable, but log events must be consistently formatted and log events should be kept short. One event should always be on one single row without a carriage return - if the log data may contain carriage returns, filter those out. Overall, log output

must be sanitized just as any other output - don't dump raw user input to the logs. Logs should be written as key="value" pairs so that they can be processed by a log management system.

Rotating logs is the responsibility of the party creating or writing logs, not the consumer. In practice, make sure that there are routines for rolling and rotating logs either using (for instance) application server built in features or operating system tools to achieve this. Rotating logs is very important - it is the only way to make sure disk space is not filled or wasted. In addition to rotating, old log files should be deleted - depending on the type of log.

Good practice is to rotate logs by their date, not by their size. This makes it much easier to find the correct logs later on.

Many times the value in logs lies in correlation to other logs. For this to be possible and straightforward, it is of utmost importance that the timestamps within the logs are matching. This in turn, requires that the log sources (servers, applications, devices, appliances, ...) are using the same (or of equal reliability) source for their time. In practice, servers MUST use NTP and their clocks (at worst) must be within a few seconds of each other.

Finally, logging into one centralized location where logs can be searched, protected and archived is very much recommended.

Security review guidelines

TL; DR

- Who: Project team (non-security professionals)
- What: Internal review done by the project team, during systems development
- When: Latest at sprint review, preferably before
- Why: To protect end users, their privacy, and employees and assets
- How: Described in this document

These instructions are aimed at the project team, with the assumption that the reader has no specific security expertise or background. The purpose of this document is to introduce to non-security experts security reviews and the process of doing them. These reviews performed during the project are not a replacement for threat & risk modelling, or a security assessment, performed for systems. The security review is best done by members of the project team, as the discussions also help shape future design choices.

Security reviews must not be delegated or outsourced to people outside the project team. The biggest long-term value comes from learning about building effective security. These decisions must be done and understood by the project team. The most effective security reviews answer precise, predetermined questions. Coming up with those precise questions is half the battle. To best equip for answering those questions, learning the ins and outs of the used technology is in a key role.

The focus of a security review should be on results and improving the security of the system. Security reviews must be blameless. The point is not to find flaws from someone's work, but to improve the security of systems benefiting customers and users. Sometimes improvement happens by making someone's code or design stronger. Findings should be treated as issues affecting the system, not as personified mistakes.

The review process should be light and agile enough to allow for an incremental approach. It's always better to have several quick and lightweight reviews during the development, than try to squeeze everything into a one session which "adds security and privacy into the project". Security and privacy team is available to assist in the security reviews, if security or privacy expertise is needed. The responsibility for the reviews remains with the project, however.

Preparations

TL; DR

- Information required for a security review
  - business case
  - protected assets
  - risks identified during threat & risk modelling workshop
  - scope
- Key personnel

Review scope should primarily focus on the system under development, taking extra care to spend time and effort on areas highlighted by threat and risk modelling. By choosing and carefully understanding the scope, you gain valuable information on the system itself and may come up with potential issues while thinking what really are the affected components.
A security review relies on information - the business case, protected assets, high level design and technical details about the implementation. The service should be approached from the top down - business process, design, implementation. If something is fundamentally insecure on a higher level, lower level choices can rarely solve those issues. Compared to design-level choices, lower level mitigations are also often more resource-intensive and complex, thus making them error-prone and expensive to implement.

Is the idea to do a peer review for target X beforehand, walk-through the findings and decide on what to do, OR is this a session where something is analysed? Former is a good way to have open discussion with a larger team, the latter is more suited for a handful of people who need to drill down to a critical, possibly complex and difficult component, which requires varied and wide expertise on interconnected systems, data, business etc.

Make sure everyone knows what is expected of them, and decide the approach. Examples are described below.

Open discussion with a larger team:

- do a peer review for target X beforehand
- walk-through the findings and decide on what to do

Expert team analysis:

- an analysis session for a complex and difficult component
- a handful of participants
- varied and wide expertise on data, business, interconnected systems

Checklist of preparations (project manager)

The preparations checklist is intended for the development team project manager or the technical team lead.

- Target and scope of the security review
  - What are we reviewing (target and scope)?
  - What are the key questions we want to answer with the review?
  - How much do we have time and how much preparation is expected of participants?
- Level of security review
  - High level review
    - Process
    - Design
  - Peer review for implementation
- Approach
  - Walk-through and discussions for peer review findings
    - Participants prepare for the session beforehand and come up with possible findings or follow-up questions
  - Peer review session
    - The analysis is done during the session (works best with only a handful of people)
  - Something else you need?
- Required documentation
  - Process description
  - Design documentation
  - Technology documentation
  - Source code
  - Applicable NFR (including related compliance requirements)
  - Identified risks (from threat modelling and elsewhere)
  - Attacker stories (explicitly chosen, and potentially applicable)
- Roles
  - Who leads the review?
  - Who demonstrates the target?
  - Who's taking notes?
- Participants
  - in-scope developers
  - architect
  - security coach

- Optional participants
  - security & privacy team representative

Invite security & privacy team representatives only when explicitly needed. For example, when new technology or architectural approach is introduced, article 9: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679#d1e2051-1-1 category personal data is handled (special categories of personal data, such as health information), threat and risk modelling has identified severe risks, or when you feel that additional expertise is required.

Security review

When doing a review, one of the most important decisions is choosing the right abstraction level. Jumping between high and low level is most often counterproductive, due to unnecessary complexity and context switching. That is - don't analyse business logic or high-level architecture and attempt to do a code review in a same session. Rather, focus on one abstraction level at a time. There are exceptions to the rule, and a small expert team might agree to do that in a goal oriented limited scope technical session.

Get everybody into the room. Have the key security questions distributed beforehand. Put project architecture pictures, data flow or code on the screen. Get everybody focused on the topic (and not reading their e-mails), with a cup of suitable beverage available. Spend as much or little time as needed.

Below are topics and questions to help you get started. Your security review can choose to use (some of) them, or approach the topic using a different style. As long as you get findings or assurance that required security is in place, and otherwise improve the overall security posture, it's all good.

Business case and data flow

- Do we understand the business use case?
- What is the data flow? Is there documentation available

Assets

- What are we trying to protect?
  - Customer information
  - Sensitive Personally Identifiable Information
  - Credit card data
  - Health information
  - Brand image
  - Authentication tokens
  - ...
- Where?
  - Server side
  - In transit
  - Client side
  - When cached in ..
  - ...

- Are there third parties or partners involved?
  - Treat external systems or services as such

Confidentiality, integrity, availability are key properties of an asset.

Attacker actions

- Modify information with something they can control
  - web, mobile client, something they can touch digitally or physically
- View or read something they can see
  - cookie values, parameter values, HTML and JavaScript source code, mobile binary, reverse engineer the firmware from a physical device, ..
- Repeat a same valid transaction or modify it slightly
- Attempt to send in an invalid transaction without doing thing X
- Come up with an unexpected way of using the system to achieve something of value to them
- Take cryptographic values (e.g. a hash) and use offline resources like GPUs to crack a secret value used to calculate it
- Steal client's computing device X and attempt to gain access to their information
- ...

Attacker tries to have an impact on confidentiality, integrity or availability with their actions.

Impact

- What's the business impact or technical impact?
  - Take impact cost into account - not all risks or vulnerabilities are worth mitigating, and some risks are too great for the business to accept
- What are the implications if an attacker can view, modify, repeat, spoof, do thing X to parts of the data flow?
- What happens if/when the data leaves our system to other internal or partner systems
  - is the integrity intact?
  - what are the assumptions and responsibilities we're passing onwards?
    - does the recipient know this?
    - is this approach future proof?
    - is there a reason to do so?

The effect an attacker can have on a system is called impact. Business impact is the most relevant criteria for determining the importance of a security issue. Sometimes technical impact can be small, sometimes big. Remember that an attacker can also try combine multiple low impact issues into a chained attack, with an impact exceeding the sum of the small issues.

Security mechanisms (lead developer / tech lead)

- What controls (if any) exist already?
  - Have we followed platform / framework best practices?
  - What are provided by the architecture?
- Are these sufficient?
  - What is the right level?
  - Does adding extra controls cost too much or hinder usability?
  - Does removing security controls lower cost or improve usability while risk level stays the same?
- Do they work as intended?
- If additional controls are needed
  - Does platform or framework offer security features?
  - Is there a popular library available?
  - Do we need a commercial product?
  - Should we develop them ourselves?
  - Do the capabilities offered by the architecture need improvement?
  - Is the resulting solution in alignment with enterprise architecture?

Properly working security mechanisms (also called controls) prevent attacker from affecting confidentiality, integrity or availability.

Remediation

- How can we remediate the finding?
  - Completely
  - Partially
  - Can only be mitigated
  - Cannot be fixed at all
- Work effort or cost
  - Trivial
  - Needs some effort
  - Needs major effort
  - Needs major design or architectural changes

Remediation describes what needs to be done to prevent the attacker from achieving their goals, or making the impact smaller. Smaller impact or greater attack cost deters some of the attackers.

Architect and product owner decide on remediation or a need for further evaluation. If needed, they will communicate the situation to the security & privacy team.

Example focus areas

- Authentication
- Authorization
- Session Management
- Input Validation
- Handling personally identifiable information
- Error Handling
- Secure Deployment
- Cryptographic Controls
- ...

Documentation

Document your findings in issue tracking or wiki, and a review summary in wiki referencing the findings. It makes sense to document in your project documentation also items, which are okay (with applicable reasoning and implemented controls) - this makes it easier to organize future security reviews, facilitate assessments, analyse bug bounty reports and library vulnerabilities, and to maintain the solution in the future.

Minimum documentation level is a memo. There is no need to copy & paste or replicate material into a new document, it's enough to reference them with a link.

Further reading

https://github.com/OWASP/ASVS (mostly for developers)

Secure input and output handling

A good amount of vulnerabilities can be avoided with strict input and output handling. It can be considered one of the foundation stones of application security, and lack of explicit input validation is a good sign your application may be vulnerable to a host of security issues. Validating input against a predefined expected format acts as a first of defence. Understanding the business requirements and planning parameter use beforehand helps create robust solutions.

When planning your input and output handling, take a look at the complete dataflow and expectations encountered along the way, and minimize all accepted content.

Complement input validation with thorough dataflow analysis, authorization, using indices instead of raw data elements (when possible) and expecting the unexpected. Output handling comes into play, when that received input is stored, processed or otherwise handled elsewhere inside the organization. Assume the worst.

Input

- Validate the following whenever possible
  - Type
  - Format
  - Schema
  - Length
  - Character set
- Design input validation to be strict and enforce the underlying restrictions
  - Invalid input will rarely be encountered in normal use cases
- Design the GUI to support a smooth user experience
  - Inform user what is expected (and don't submit invalid content further), if user provided input is asked
  - Do not rely on GUI for input validation, though
- Use whitelist-based approach
- If a blacklist needs to be used, make sure you have a really solid reason for it
  - Do not attempt to sanitize the data - just stop processing and return an error
  - How has the blacklist coverage been verified?
  - How do your frameworks, libraries and other components work with different encodings?

Output

- Understand the context where output happens
- Encode the information based on the output context and take care of special control characters accordingly
  - Different output environments have different structures and special characters (Angular template vs HTML page vs JavaScript code block vs JSON)
- When dealing with data structures (such as JSON) decide what is needed by the front end, and return only those elements
  - Avoid passing received backend responses "as is" to the client if the responses contain information not needed by the client

Integrations

- Understand the special requirements of the receiving application(s) (throughout the whole data flow) when it comes to user input.
  - What kind of input do they expect and what happens if the received input contains control characters?
  - What are the control characters in each context? Are the processed securely before passing the data onwards?
  - For example, you have a legacy system in the chain which uses character § as a field separator. User input with § might break this processing.
- Document what the receiver can expect from the data, especially if the input has been received from the end user
  - e.g. verified and checksum validated credit card number vs raw free-form text data with full UTF-8 support
- Ensure the received input cannot be confused into control structures during integration transit, assuming end user submits content containing e.g. JSON, CSV or other data structures

3rd party libraries

According to studies, news (https://www.veracode.com/security/open-source-risk and https://www.zdnet.com/article/backdoor-code-found-in-11-ruby-libraries/) and practical experience from LocalTapiola's bug bounty program, third party software and libraries can introduce severe vulnerabilities into developed software. These flaws many times go completely under the radar. Managing external software and libraries is not only a development phase issue, but third-party components have to be taken into account during the whole life cycle of an application. Libraries must be managed regardless of whether the application is under active development or not.

Third-party libraries or software as a security domain include all third-party code that is being used in the applications regardless of the programming language used. This includes all solution stack components from the core systems to client-side JavaScript and everything in between. The text refers to terms third-party library, software or code interchangeably - in all cases the intention is to cover all uses of externally developed and maintained code used in development projects, regardless of terminology.

As a technology, SCA (Software Composition Analysis) provides help through automated scanning of code. Teams should consider using suitable tools for managing libraries and vulnerabilities.

Guidelines for compliance

These are the guidelines that must be followed at all times.

Choosing a library

- Selection criteria
  - Libraries that are included in an application must be well known, trusted and widely used
  - If library is not actively maintained, analyse carefully whether this is acceptable or does it introduce unwanted risks for the application in the long term
  - Licensed in a way that does not put the organization in a legal conundrum. A good source for information around licenses can be found here: https://tldrlegal.com/licenses/browse. Discuss with your project manager who can consult legal if needed.
- Document the chosen libraries using the provided template

Using a library

- Do follow all license terms and include acknowledgements and/or license information if mandated. Not following the license terms and their requirements can result in legal issues.
- Do not create private internal forks of libraries. Maintaining in-house versions of open source libraries is not what our organization does.
- Do submit bug fixes to the library maintainer and create pull request if you come across issues needing remediation

- Read the library documentation and understand how it is supposed to be used. Be aware of any security decisions and assumptions the library maintainer has made. Library documentation often contains useful security and practical information.

Updating a library

- Libraries must be regularly updated.
  - During development
    - For applications that are under active development the required timeframe for updating is one month/every sprint/every release/other.
    - During sprints use available tools, such as npm audit or Veracode, to verify the status of libraries in use.
  - During maintenance
    - For applications in production but not under active development, libraries must be updated every six months.
- When libraries are updated, the requirement is to update libraries to the latest minor version of the major version that is currently being used in the application. Library major versions must be kept within one version of the current publicly available stable major version.

Practical approach

A practical but pragmatic approach to library maintenance is needed. We do realize that updating a library might cause the application to behave in mysterious ways. This however, is not a valid reason for not updating. During application development, libraries should be updated in the most fearless way - as applications are nevertheless tested for functionality, one should assume that if no issues are uncovered, the updated library version is working correctly. For production applications not under current development, simple functional tests must be conducted before going into production. The probability for incompatibility is many times found in the release notes. By maintaining near-current versions of libraries in applications, the probability for a catastrophic failure is heavily reduced.

Vulnerability and patch analysis

Understanding the impact and technical details of vulnerabilities is paramount for cost-efficient security. Vulnerabilities in libraries, even serious ones, may have no real-world business impact. For example, if the vulnerability affects a function or module in a library, which is not used by the application directly or indirectly (through library's other functions), there is no attack path for a malicious user to exploit the issues. Especially during development, it may be easier to update the library than to perform a thorough vulnerability and patch analysis for a suspected or identified issue. During maintenance phase the situation may be different and if no new releases are planned for some time, it may be worthwhile to do the analysis.

Steps for vulnerability analysis

1. Collect technical vulnerability details
    a. Description of impact and exploitability
    b. Original vulnerability report (and proof of concept if available)
    c. Change log, patch or commit
2. Analyse the data flow
    a. Determine what parts of the component need to be reached in order to exploit the issue
    b. Analyse how user input passes through the application to the library
        i. Does the application utilize vulnerable function(s) of the library
            1. directly by calling them explicitly
            2. indirectly by utilizing other features, which end up invoking vulnerable features
        ii. Is there input validation or other transformation affecting the data flow
            1. strict input validation may prevent exploitation. Transformation or partial validation may increase complexity of exploitation but not prevent it
    c. If input may reach the vulnerable functions, does the application logic or other behaviour impact exploitation
        i. reachable and exploitable in normal application flow
        ii. reachable, but not exploitable because of reason X
        iii. reachable in corner cases (such as a specific error condition) and exploitable
        iv. reachable in corner cases but not exploitable because of reason X
3. Analyse the business impact for exploitable vulnerabilities
    a. What can the attacker achieve with successful exploitation
        i. Impact on assets (personally identifiable information, financial instruments, brand and reputation, ..)
        ii. Technical impact
    b. Are there other elements in the architecture, which mitigate the issue partially or completely
        i. Does the mitigation work with current configuration or does it require additional steps
        ii. Has this been verified in practice
    c. What are the prerequisites for successful exploitation?
    d. Does the impact (considering exploitability requirements) constitute a real world risk
4. Recommend remedial, corrective and preventive actions
    a. Actions
        i. Update the affected software component with version n.nn
        ii. Mitigate the vulnerability class / similar instances / this particular vulnerability by implementing X,Y,Z in solution stack
        iii. Mitigate the vulnerability class / similar instances / this particular vulnerability by Z,Y,X in architecture component A,B,C
    b. Schedule
        i. Given the real-world risk, when do we need to act

HTTP headers and CORS

Application security can be improved or weakened by configuring certain HTTP level headers. Setting up your environment the right way in the beginning strengthens the security posture, and makes it harder to conduct certain attacks against the end users. Suitable values can always be adjusted based on individual use cases.

HTTP configurations and hardening are implemented on load balancing / SSL termination level, main application server, and/or web server - depending on your architecture and current configuration. Technically speaking, configuration hardening is pretty straightforward and usually has little to no effect on application functionality (your mileage may vary). It is highly recommended to harden the configuration in the test environment as early as possible, so that any unwanted behaviour can be detected and the configuration is an integral part of the testing process. Try to avoid situations where the configuration differs in load balancer and web or application server. Understanding where the response originates from and which component is controlling the headers (and when) may sometimes be tricky.

Header configuration

HTTP hardening consist of following subparts:

- required HTTP headers -- adding the headers and removing redundant headers
- cookie policies and cookie hardening
- CORS (Cross-Origin Resource Sharing)
- allowed HTTP methods
- other hardening, requirements and exceptions

Example configuration:

```
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Strict-Transport-Security: max-age=86400; includeSubDomains
Referrer-Policy: strict-origin-when-cross-origin
X-Content-Type-Options: nosniff
Clear-Site-Data: "*"
Cache-Control: no-store
Pragma: no-cache
```

Note! `Clear-Site-Data` should be served after logout only.

CORS configuration

Consider to implement only if:

- application or API is intended to be used by a third-party web site

Default CORS configuration must apply - additional CORS headers for any part of the application must not be defined. Differing needs must be documented separately using a sequence or communication diagram. It is equally important to understand CORS as a technique and the basis for using it.

If CORS options are implemented without careful planning:

- your application may leak data
- could make the application vulnerable to data alteration

Before attempting to create a CORS configuration, do familiarize yourself with Same Origin Policy, Cross Origin Resource Sharing and pitfalls associated with opening up the CORS configuration.

Following CORS headers are explicitly forbidden:

- `Access-Control-Allow-Origin: <generated dynamically from client sent Origin header>`
- `Access-Control-Allow-Origin: *`

Further reading

- https://www.owasp.org/index.php/OWASP_Secure_Headers_Project
- https://en.wikipedia.org/wiki/List_of_HTTP_header_fields
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS
- https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy
- https://www.w3.org/TR/cors/
- http://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html
- https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29
- https://www.owasp.org/index.php/SameSite

Afterword

The document you have in your hands is by no means meant to serve as a single truth. The information presented here is based on processes, best practices, lessons learned and collections of guidelines developed in LocalTapiola. We maintain and constantly develop these guidelines internally and developers and service providers that deliver projects to us always have the latest and finest versions available. The documentation is based on our internal experience as well as many public sources - we have not really invented anything new, we have only made an effort to gather important information into one single document. We wanted to make a version of our security guidelines for secure development available to the public for anyone to use as they might deem suitable. Hence, the ideas and suggestions in this document may suit you well, - or not at all. It is up to the reader to interpret and apply anything in this document and make it suitable for their own use. As far as the more technical parts of this document goes - caveat emptor - they will become outdated.

Also please notice - if you are working on LocalTapiola projects, please remind yourself and your team that you should only use the latest version of these guidelines available internally.

Reference list

- https://www.bsimm.com/
- https://www.owasp.org/index.php/Security_Champions
- https://www.owasp.org/index.php/Security_Champions_Playbook

Credits

- Leo Niemelä, LocalTapiola
- Elina Saartoala, LocalTapiola
- Markus Forsström, LocalTapiola
- Teemu Talvitie, LocalTapiola
- Maija Raitio-Hirvi, LocalTapiola
- Elina Partanen, Mint Security
- Thomas Malmberg, Mint Security
- Teemu Turpeinen, Mint Security
- Saku Tuominen, Mint Security
- Henri Lindberg, hmask