Embedding Security in Product Lifecycle

Arvind Srinivasa Babu, Deepti Chauhan

Arvind_Babu@McAfee.com, Deepti_Chauhan@McAfee.com

Abstract

A product goes through several processes before it is released into the market. An oversimplified process will involve a lot of planning, analysis, design, development, testing and marketing before a release. But how many products involve security in this lifecycle? During a product's lifecycle, a lot of new features, bug fixes and other development activities take place which essentially means that new code is being added. Though this code might stand up against good product test cases, what is the confidence level that this new code has not opened a backdoor or added a vulnerability that would allow an attacker to exploit, monitor or cause damage?

This paper will explain how some best security practices can be incorporated within a product lifecycle. We will also demonstrate why it is essential to maintain threat models up-to-date for every release which allows automation of security test cases as the product evolves. Automated security test cases allow us to add security as part of a Continuous Integration(CI)/Continuous Delivery(CD) pipeline. There are several open source tools that allow testing different aspects of security on a product, we will glance over some of these tools and the functionality they bring to security testing.

Biography

Arvind Srinivasa Babu is a Software Development Engineer and Product Security Champion at McAfee, with more than eight years of experience designing and implementing software. He has been performing the role of a scrum master and strives for continuous improvement in testing and development. His areas of interest span over security, network, user interface interaction, system programming, product security and mobile application development.

Deepti Chauhan is a Software Development Engineer at McAfee, with five years of experience in developing products. She has participated in various product life cycles and been a key contributor to various releases within McAfee. Her areas of interest span over system programming, network and mobile application development.

1 Introduction

Software Quality is gauged by a lot of different metrics like complexity, reliability, efficiency, security, maintainability, scalability and much more. We believe that a software's durability relies on triangular aspects of Security, Performance/Efficiency, and Quality. We define quality as an aspect that includes simplicity, maintainability, scalability and most important of all offers minimum or no disruption of customer business. Product lifecycles have evolved over the past couple decades from linear validation phases to validating software in a continuous manner. An engineering team depending on the life cycle they have adopted would have acceptance criteria, which would try to balance security, performance, and quality. A software that doesn't find the right balance between these aspects will lose business. For example, if we define a metric to measure the quality, security and performance of a software in broadly 5 levels (0-4) as shown in the below table,

Maturity Level	Quality (least disruptive)	Security	Performance
0	Frequent crashes	No security process in place to mitigate vulnerabilities caused by the product.	Consumes resources that cause maximum CPU usage
1	Major functional failures	High severity vulnerabilities reported from field for every release. Few security processes followed.	Causes slowdown of entire system
2	Minor functional failures	Medium or low severity vulnerabilities reported from field for every release. Few security processes followed.	Causes noticeable slowdown when performing certain tasks
3	Mostly Non-Functional or some functional failures with no impact to customer business. (logs, spelling mistakes etc.)	Most security defects addressed before release of product, partial adherence to security process within a product life cycle. Occasional reports of vulnerabilities from field.	No degradation of system resources only optimization required is with the algorithms employed within the software.
4	No failures after released	No vulnerabilities reported from field. Adherence of secure product lifecycle.	No impact on business and product is at max efficiency.

The following diagram measures different version of a software each representing different maturity levels in every aspect. While every team aims for a perfect product, budget cuts, process execution blunders and cutbacks lead to compromises which impacts any of the aspects mentioned.

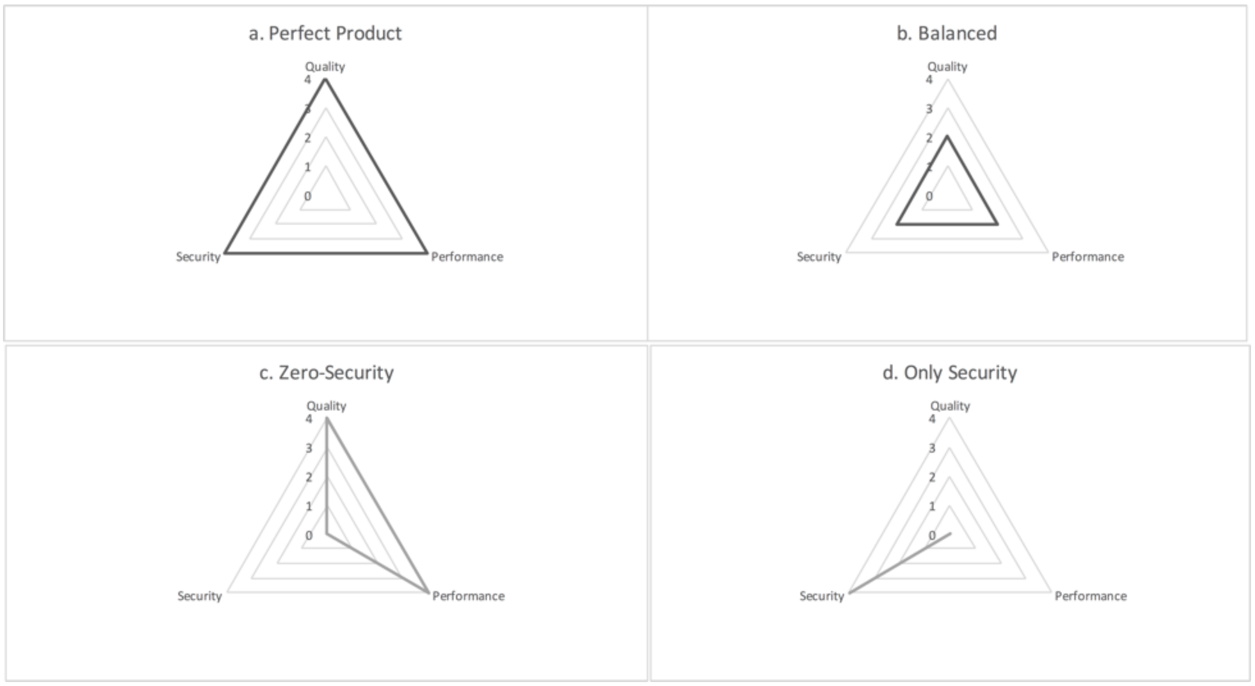

Figure 1 - Different charts that show the relationship triangular aspects of Quality, Performance and Security measured on their maturity levels.

In this paper, in the first section we will be talking about how typical product life cycles look with their activities, different methodologies, their philosophies and what are the advantages and disadvantages when executed well. The second section talks about differences between product defects and a vulnerability and how they need to be handled. The next section dwells on software security, the different activities involved when dealing with software security followed by a detailed section on how to embed these activities in different phases of a product lifecycle. The next section will provide an insight on what threat modelling is and why it is important to maintain threat models up-to-date during every release. We will finally have a glance on the functionalities provided by various tools that can help perform penetration testing on your products.

2 A Typical Product Life Cycle – Waterfall, Agile or DevOps

A textbook definition of a product life cycle is the progression of a product across multiple stages from inception, requirement gathering, design, development, validation, release and support. Over the past couple of decades, various teams have followed multiple product life cycles to improve their reliability in making releases in a stipulated time. A few largely popular lifecycle models are Waterfall, Agile Scrum, DevOps, most models are either a spin-off of the above or a hybrid among them. The key takeaway from comparing multiple product lifecycles is the improvement in the ability to assigning tasks in parallel and execute projects quicker.

Waterfall product lifecycle model [1], one of the oldest process models, focused on every stage in a sequential manner, this usually leads to software release taking up to a year or two.

An Agile product lifecycle model [2], allows a faster execution of a software release, by splitting products into features and minor incremental deliverables. These incremental deliverables are accumulated, validated and finally the product is released. Usually each sprint is a couple weeks for receiving minor product increments and product can be released every month, or quarter.

DevOps [3] is an evolving product lifecycle in its methodological stages that requires heavy collaboration between Product management, Engineering and Operations. It involves a collection of tools to help facilitate automated deployment or delivery of a product in a continuous fashion.

3 Product Defects vs Security Defects vs Product Vulnerability

The most popular terms used within engineering teams related to security are defects and vulnerabilities. While an engineering team might encounter such defects, and use of this terminology within their product lifecycle, it is important to understand the differences between these terms and the impact they have on the product, company and brand. We will focus on the differences between Product defects, Security defects and a Product vulnerability.

Product defects are those kinds of defects that typically affect the product's ability to fully function. This could be a result of poor design, poor algorithm, poor validation of functionalities that eventually lead to disruption of customer business thereby decreasing the quality of the product delivered. These are the kind of defects that are detected and corrected in a product life cycle's validation phase. There are multiple types of product defects that affect the performance, or functionality or provide poor integration with other components. These can either be detected early in the lifecycle or late. As the product moves along its lifecycle, it might not be feasible to address all product defects due to budget and time constrains eventually leading the defect to be deferred for a future release. Most low severity bugs that have least business disruption capabilities are picked up for deferrals.

Security defects are the kind of defects that arise from products that provide security features or controls like Confidentiality, Integrity and Availability or handle sensitive data like keys, passwords etc. but the security is compromised because of incorrect implementation of security controls that eventually leads the system to grant unprivileged user access rights to privileged information or denial of service. Security defects can range from incorrect memory handling (buffer overflows, incorrect de-initialization etc.), input validation errors, incorrect thread synchronization, incorrect authentication, or lack of privacy controls.

A **Product vulnerability** is the mechanism which exposes the weakness within a product that allows an attacker to exploit the system to gain entry and access confidential information or cause business disruptions. While a vulnerability is often confused with security defects, vulnerability is more focused on the methodology by which an attacker can gain access to the system. It requires a great deal of analysis to find vulnerabilities from a black box perspective. A security defect leaves an exploitable piece of code that may or may not remain discovered, vulnerability provides the mechanism to access the exploit and an attacker uses that to exploit a system.

Product defects, security defects and vulnerabilities are terminologies that are often interchanged and cross-referenced within product teams and customers. To a customer, the only terms they would be familiar with are "vulnerabilities" and "support cases". Vulnerabilities in products can send general alerts and panic within the customer's security team and would request the engineering team to provide a fix. A business disruption because of product defects on the customer side would lead to support case to be opened. To an engineering team, the operational term would be a "defect" or "bug". The details of these defects may never reach the customer view. When engineering team evaluates a vulnerability, the root cause analysis would point to a security defect which will be addressed within the lifecycle. A product that has a lot of vulnerabilities caused by undetected security defects within its lifecycle, would ultimately lead a customer to move to competitors who provide better security products with minimal or no vulnerabilities.

4 What is Software Security?

Software Security is the assurance that a product is provided free of vulnerabilities to its customers. Software security like Quality and Performance cannot be built into a system in a fixed timeframe, it's a continuously evolving process that needs to be properly executed and documented for future enhancement.

Software Security Process can be termed as a set of security activities that can be performed across various stages of a product life cycle to detect and correct security issues. As mentioned earlier, a product life cycle methodology focuses mainly on delivery of features and not on the set of life cycle activities. Following is the set of security activities that detects and addresses different security aspects for the product.

4.1 Security Activities

4.1.1 Architectural and Security Design Review

This activity will focus on architectural reviews and design reviews from a security standpoint. If there are no architectural changes, design reviews alone can be carried out for new features else both the activities can be combined as a single activity. Major focus areas in this activity would be to establish the trust, integrity and data handling of the product or feature in secure fashion. This activity would involve senior members of the product team like architects and senior leads who will brainstorm on implementing security from an architectural or design standpoint.

4.1.2 Threat Modelling

This activity involves having engineers who understand the product very well identify data flows within the product. Each data flow, and architectural input help the threat modeler to identify attack surfaces and threats that can compromise the system. This input can be taken into an architectural or design review to address any concerns on the threats and attack surfaces. Once a threat model is prepared, it will present a list of security requirements for the product to implement and provide security against identified threats. We will explore more on threat modelling in a later section.

4.1.3 Coding Standards and Manual Code Review

Every product development team follows a coding standard to keep the style of code in a readable format. Security engineers within a team are responsible to collect, collate and maintain a repository of secure coding standards that can be applied during development. This helps in building security right from the point where code is written. Each language used for development has its own secure coding standards and is responsibility of the engineer or team to monitor and update the repository as more standards are introduced.

We recommend making code reviews a mandatory activity that must be performed for every code commit to ensure that the product team is adhering to the defined coding standards. While most code reviews focus on functionality of the feature to catch early defects, it’s important to monitor the security standards being properly followed. Every code review must at least include two or more senior engineers, ideally where one can focus the review on functionality/architectural correctness and the other can focus on the security standards adherence.

4.1.4 Static Analysis

Static Analysis is a practice where the code is examined without executing it for common programming errors when handling resources or memory handling etc. While there are multiple static code analyzers like Synopsis Coverity, Fortify, Clang, Visual Studio Code Analysis, while each of these analyzers have their strengths they also have a weakness in detecting false positives. We recommend running multiple static analyzers on the code for every commit or on regular intervals if analysis takes a lot of time. We recommend making this a mandatory practice that must be followed throughout the lifecycle. Static analysis is the second line of defense for products to detect memory leaks from analyzing code patterns. Although there are risk of false positives arising from an analyzer, running multiple analyzers to get a comprehensive summary of issues and addressing them will improve the products stability and security posture.

4.1.5 Dynamic Analysis

While static analyzers examine code without executing it, Dynamic analysis focuses on examining code as it executes data in real-time. This gives more accurate information on memory leaks, input validations and other types of privilege escalations. This activity becomes conditionally required if you are dealing with web applications and new features dealing with lot of memory management. Fuzzing is a methodology by which random data is supplied to input interfaces of the software to check for potential errors in handling unrecognized data. Smart fuzzing involves mimicking the data to be a valid input format with few alterations which presents much more detailed coverage into your system. Dumb fuzzing is simply supplying a random blob of data which has the potential to cause crashes due to incorrect format parsing etc. While each method of fuzzing has its advantages and disadvantages, it is up to the security tester to employ the fuzzing techniques suited for the product.

4.1.6 Security Testing Plan

It’s important to charter a plan that would allow a team of quality assurance engineers to pick up security testing activities. This planning activity will be carried out every sprint and the test executions can be automated or performed manually single or multiple times in a release depending on the bandwidth availability of engineers and process lifecycle. Documenting the security plan for future use will help identify testing gaps when vulnerability slips through security testing.

4.1.7 Vulnerability Scanning

Vulnerability scans detect known vulnerabilities and exposures in a binary executable or in the environment where it is being deployed. This activity is conditionally required to be performed if there are new features that use system resources like opening socket connections or access new files etc. Mostly software implementation consumes third-party libraries, and these third-party libraries can be an older version with newly discovered vulnerabilities. Vulnerability scans can identify such libraries and provide information on the various vulnerabilities that have been directly or indirectly introduced in the software.

4.1.8 Privacy Reviews

Privacy should be the top most priority for any software product that handles Personally Identifiable Information (PII). A Personally Identifiable Information refers to any information that allows an attacker to identify a specific individual. This review process determines the storage access controls for PII that should be implemented. This is mandatory activity if you are accessing PII data anywhere in the system irrespective of the time the data is held within the product.

4.1.9 Penetration Testing

Penetration testing is a complex activity involving analysis of data collected from multiple tools to ethically break into a system and access privileged information. There are multiple types of testing performed, depending on the type of testing, platform, network, technology where the software is deployed. The level of penetration one can achieve in a system is entirely dependent on the skillset of the individual(s) performing the pen-testing so this activity should be performed whenever required either by discovery of vulnerabilities from the field or when there are significant architectural or design changes. There is a cost associated, since members with a specific skillset are required to perform pen-test activities.

5 Embedding Security Activities in Product Life Cycle

In the previous section, we had a glance at various security activities that handle various areas of security. Explaining what each product life cycle is beyond the scope of this paper and hence we will elaborate more on incorporating security activities in each type of model to maximize a product’s security posture.

5.1 DevOps

DevOps is a combination of Development and Operations that facilitates a reliable delivery of software. This is more of a methodology rather than a product life cycle but is being adopted in many organizations in recent times as a product life cycle. The success of DevOps is complemented by Lean and Agile paradigms and adopts the best of both.

5.1.1 DevOps Overview

DevOps primarily has seven stages which involves the following activities:

5.1.1.1 Plan (Stakeholders planning)

- Collect Business requirements.
- Define metrics
- Architecture
- Infrastructure planning
- Release Scope and Planning

5.1.1.2 Code (Continuous Integration)

- Design feature and configuration
- Develop software features
- Build

5.1.1.3 Validate (Automation)

- Automated testing
- Acceptance testing
- Configuration testing

5.1.1.4 Staging (Packaging)

- Approvals
- Package Configuration
- Pre-release pods

5.1.1.5 Release (Continuous Delivery/Deployment)

- Deployment
- Fallbacks, Recovery

5.1.1.6 Configure (Infrastructure Orchestration)

- Storage
- Database
- Application Provisioning

5.1.1.7 Monitor (Telemetry)

- Production Metrics
- Performance

While DevOps is aimed at continuously delivering or deploying products, every feature and defect must pass through these stages. The amount of code being released could be as simple as few lines or code or could be multiple huge complicated components.

5.1.2 Embedding Security Activities in a DevOps Model

The key to providing security in a DevOps model where code is to be delivered as early as possible with highest quality, performance and security, would require a cultural change in an organization following agile scrum or any other product lifecycle, as well as heavy reliance on automation. There is also a fundamental change in how product is viewed in a DevOps environment. DevOps is usually implemented for solution containing multiple products. So, there are few key activities that need to be performed on an overall security perspective.

5.1.2.1 Plan

As mentioned, stakeholders include product managers, IT operations, Engineering team, security engineers and architects discuss on the architecture, infrastructure and requirements which defines the scope of a release. Security engineers from each product contribute in performing **Architecture reviews**, **Infrastructure reviews** (an activity that will review the security controls in place to mitigate threats from accessing the infrastructure e.g. Private Cloud solutions, Amazon Web Services (AWS), Microsoft Azure etc.), and **Threat Modelling** the infrastructure and architecture. A **Security Audit** of consumed third-party modules is also initiated eliminating weak third-party libraries. Third-party evaluation is a continuous process by itself and evaluated versions are taken up for audit to decide whether libraries are no longer applicable or whether it needs to be updated to newer versions. **Security Test Plan** is created

5.1.2.2 Code

The only manual activities in this phase is development, code review of the feature by the engineering team. **Continuous Integration(CI)** should be implemented to ensure unit-tests are in place, **Code reviews** will be used to promote the code into the CI branches [4] from feature branches. After which Static Analysis is performed on the code. Unit tests also contribute to a certain level of **Dynamic Analysis**.

5.1.2.3 Validate

Automation is the key for **Continuous Delivery (CD).** While automation will focus on functional testing, security engineers will focus on automating the Security Test Plan into the CD pipeline. **Dynamic Analysis** is performed by providing fuzzed data to the various inputs to check for any failure in input validations. Security engineers also review how Personal Identifiable Information (PII) is being accessed and handled and perform **Privacy Reviews** to ensure that proper access controls are in place. Threat models also needs to be updated if issues are found during security testing.

5.1.2.4 Staging

Once all issues are addressed, packaging is performed to ready for release. Cloud solution would prepare for release by deploying to a staging area like Beta Pods, or pre-release pods. **Vulnerability Scans** can be performed in this stage to review any known vulnerabilities that is accidently deployed in pre-production environments. Issues discovered would then be addressed before deploying onto a production environment.

5.1.2.5 Release

At this point, security checklists must be met. Static Analysis, Dynamic Analysis, Vulnerability Scans, Privacy Reviews, Threat Models being up to date, Code Review data must be artifacts for post-release reviews or for simple recordkeeping.

5.1.2.6 Configure

Once the release is deployed, appropriate storage, virtual machines, databases are provisioned or configured with the new version of the application. Deployment tests should include an automated security validation to scan for configuration errors like failing to apply necessary patches, or configuration to ensure appropriate access controls are in place for the infrastructure. Closing out Infrastructure reviews in adherence to the Infrastructure plan is crucial and needs to be worked upon by both security engineers and IT operations managing the production environment.

5.1.2.7 Monitor

This is mostly IT operations to monitor the stability of cloud after deployment of the solution, but security engineers can periodically monitor the infrastructure by deploying internal scanners to periodically scan for vulnerabilities or malwares.

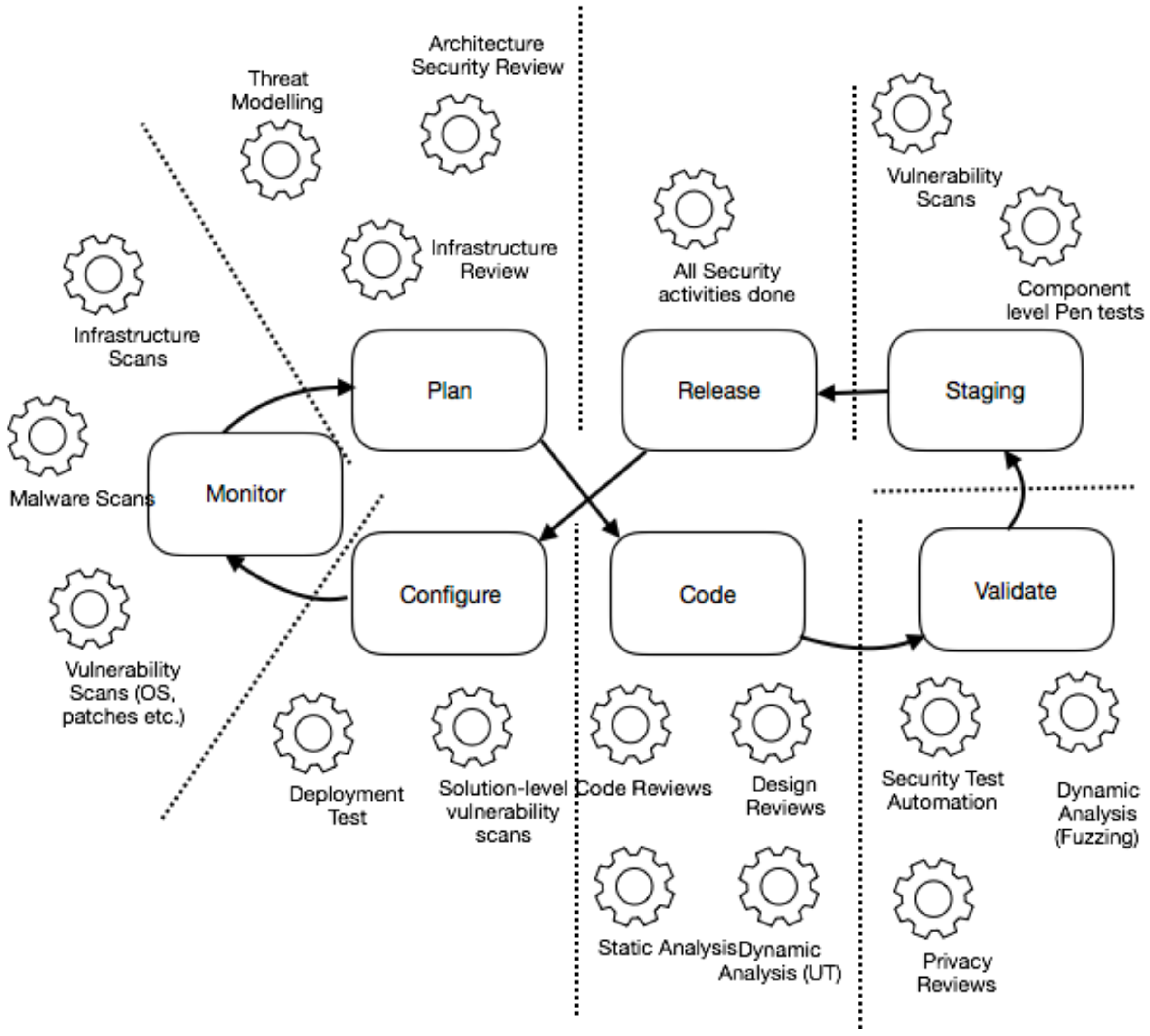

Figure 2 DevOps Lifecycle for every feature/bug.

5.2 Agile

5.2.1 Agile Overview

Agile product lifecycle is one of the most successful models in recent times, it is executed in a specific timeframe and is iterative to deliver the final product. The goal is to split the product into epics and user stories that can be delivered incrementally until the full product is developed. There are several methodologies within the agile philosophy and a few of which are worth mentioning is Kanban, Scrum, Extreme Programming, Lean Development etc. For the scope of this paper we will be embedding security within Scrum model.

An agile scrum model involves the following stages within which the following activities are followed

5.2.1.1 Release Planning

- Plan of Intent
- Product Backlog
- Epic creations
- Feature Prioritization

5.2.1.2 Sprint Planning

- Team backlog
- Story creations
- Task breakdown
- Story Estimation
- Prioritization
- Capacity planning

5.2.1.3 Sprint Tracking

- Design
- Daily Scrum
- Burndown
- Build
- Validation

5.2.1.4 Sprint Review

- Potentially shippable product
- Retrospective
- Reprioritization

5.2.1.5 Release

- Deployments
- Release to customers
- Documentation

5.2.2 Embedding Security in Agile Scrum

5.2.2.1 Release Planning

In this stage, security engineers are engaged early on to discuss on the requirements and defining the scope of changes for a planned release. **Privacy Reviews** are initiated and data gathering starts for all PII that is being handled. **Threat modelling** activity is initiated using architecture diagrams and **Architecture Reviews** are performed. Teams perform architecture within sprints or they do a complete architecture and break it into deliverable sprints. Either way, threat model needs to be updated accordingly as requirements keep changing every sprint and appropriate updates needs to be made.

5.2.2.2 Sprint Planning

In this stage, the architecture is broken into incremental deliverable modules. **Threat Modelling** activities are performed at this stage capturing the minor incremental code changes. **Security Test Plan** is created for the upcoming sprint and targets defined.

5.2.2.3 Sprint Tracking

Design of incremental deliverables are performed in each sprint and corresponding **Design Reviews** needs to take place. **Code Reviews** are performed for every code commit that is checked into the version control system. **Static Analyzers** are run on the codebase to identify issues for every build that is triggered. **Dynamic Analysis** can be scoped and performed on the incremental deliverables by performing fuzzing, detecting memory leaks etc. **Security Test Plan** execution is undertaken for every sprint and results updated accordingly. Automating these test cases wherever applicable allows faster execution of security tests. **Privacy Reviews** should be conducted regularly to ensure that proper security controls are being implemented when handling PII information in every sprint.

5.2.2.4 Sprint Review

Each sprint review will have a sprint checkpoint for security activities to meet a Definition of Done (DOD) where Sprint Security Test Plan is executed successfully, all issues addressed and fixed. Any defects that have low severity and moved into next sprint needs to be fixed by the next sprint and backlog is groomed to accommodate these bug fixes. This security checkpoint will also expect Static Analyzers and Dynamic Testing to be performed.

5.2.2.5 Release

All documentations and artifacts related to the security activities needs to be archived for future reference. Third party evaluations completed and updated to the product. **Privacy Review** must be completed. Security Test Plan execution must be completed and any issues arising from the same needs to be addressed. All issues that arise in the field both product and security defects enter into agile lifecycle as issues or stories and are picked up in the backlog of the new version during the release and sprint planning.

5.3 Waterfall

5.3.1 Overview

The waterfall model is one of the oldest product life cycles, it is a sequential execution of stages starting from inception, design, development & validation, release and support. The stages in waterfall model are self-explanatory. The inception stage involves lot of planning, identifying business requirements, once this is performed, design of the modules and architectural plan is formulated for the entire product and all the modules. Once the design and architecture has been finalized, the development team proceeds with implementation of the design and the product is given for validation. Once the product is validated it is released and post release support is provided. Though waterfall is the oldest and is phased out by most companies and product teams there are still some areas of the waterfall model which is still being followed as either a hybrid model like waterfall-agile etc. We will cover how security activities can be embedded within this model so that it can be applied on hybrid models.

5.3.2 Embedding Security Activities in Waterfall

5.3.2.1 Inception

During this phase, it is critical to include your product security engineers in the loop to ensure that they understand the impact of changes that is being planned for a release. Security engineers can use the opportunity to create a plan of security activities that can be achieved in each phase and execute it as they deem necessary. Security engineers can also use this phase to set up **Secure Coding Standards** database with list of all coding standards that will be applicable and developers needs to follow during the development phase.

5.3.2.2 Architecture & Design

During this phase, **Architecture and Design Security Reviews** can be performed. Security engineers should have a strong architectural review skills to identify basic security problems with an architecture or design. **Threat Modelling** should be performed here for the new architectural or design change and used in reviews. Identify all Personal Identifiable Information (PII) data that the product design is going to make use of and prepare for **Privacy Review**.

5.3.2.3 Development & Validation

Most activities performed here would be to run **Static Analysis** to detect problems with memory or resource management and fix them regularly. **Code Reviews** should include senior engineers and security engineers to ensure that the coding standards are adhered by the development team. Security engineers and Quality Assurance engineers should work together to formulate a **Security Test Plan** for the project. During validation, the security test plan should be executed alongside functional test plan. **Dynamic Analysis** can also be performed during this stage providing fuzzed data and detect memory leaks. **Privacy Review** must be performed to enforce secure handling and enforce appropriate access controls of PII data.

5.3.2.4 Release

Most of the product development would be complete at this point, with proper code reviews, static analysis, dynamic analysis, proper privacy access controls must be in place and complete security test plan executed. Any issues arising from these activities needs to be addressed within the release. An activity of **Penetration Testing** can be performed at this point which is entirely dependent on the skillset of engineer who is performing the pen-test and cost associated with it. With a final review of all PII data performed and reviewing all open source licensing agreements, the product can be ready to be released.

5.3.2.5 Post-Release and Support

With any product model, there are bound to be issues that can slip even in case of following above activities. There should be mechanism where external pen-testers or customers can report issues and you can address security issues through hotfixes or patches. In event of such cases, threat model should be updated with the new attack vector.

6 Threat Modelling

6.1 What is Threat Modelling?

Threat Modelling is a structured approach that involves identifying, accessing, quantifying and mitigating security issues. Threat Modelling is a defensive posture in security where the security engineer is the attacker and models the architecture in terms of identifying possible threat agents and attack surfaces and create a plan to address them as the software evolves through different stages of a product life cycle. With current technology standards, it is not possible to automate this activity and involves manual interaction and analysis.

6.2 Who can Threat Model?

Any engineer who has a sound product knowledge and understands what threats and attack surfaces are can come up with a threat model for their product or solution. Threat modelling is not a skill that can be learnt off the bookshelves and perform some diagrammatic representation of threat vectors and attack surfaces, it is about identifying all possible threats and this skill to identify threats from threat models improves by doing multiple threat models for smaller designs and multiple architectures.

6.3 Threat Modelling Guidelines and Vocabulary

6.3.1 Brook Schoenfield's: ATASM [18]

Brook's ATASM model focuses on the following

- **A**rchitecture
- **T**hreats
- **A**ttack **S**urfaces
- **M**itigation

A detail of this threat model is provided in Brook's book mentioned in the references section and conference presentations [17]. The goal is to arrive at security requirements that will be built to bring the system to the desired defensive posture.

6.3.2 Microsoft's STRIDE

Microsoft STRIDE model [5] is a threat classification model which focuses primarily on the following categories

- **S**poofing
  Any threats that allow an attacker to illegally access information using another user's authentication. In the threat model, all areas where such sensitive information is accessed or handled will be brought under the threat model and security test cases will be designed based on that.
- **T**ampering
  This category of threat deals with unauthorized access to modify persistent data like databases, files and data transmitted over network.
- **R**epudiation
  Repudiation deals with establishing trust information on the authenticity of the user. Audit logs, general product logs, purchase receipts, login credentials are all ways to establish actions performed by a user.
- **I**nformation Disclosure
  Information disclosure revolves around threats that provide access to users who are not authorized to read the data. Few examples would be to read a file without proper access controls etc. Personal Identifiable Information also falls under this category.
- **D**enial of Service
  Denial of Service attacks deny access to valid users by bombarding the system with too many negative authentication calls.
- **E**levation of privilege
  Elevation of privilege involves an unauthorized user penetrating into the trust level of the system. The level of escalation leads for untrusted entity to be part of the trusted system.

6.3.3 Process for Attack Simulation and Threat Analysis (P.A.S.T.A)

Process for Attack Simulation and Threat Analysis [6] is a Threat Modelling methodology which involves identifying the potential threats and risks assessment. The model involves identifying the Threats/Vulnerabilities from attacker's viewpoint. PASTA is a seven-step process. The process starts with identifying the business objectives and requirements along with the security requirements. Followed by decomposing the application using DFD and Use Case diagrams etc. It involves further Threats and Vulnerabilities analysis and using methods like Attack Trees and Attack Surface analysis for modelling the attacks. The process finally ends with Risk and Impact Analysis.

6.3.4 Vocabulary

Threat Model

A threat model is a diagrammatic representation of data flow diagrams in conjuncture with architectural components.

Asset

An asset is any device, data or component that is crucial for business continuity which generally requires an access privilege.

Attack Vector

An attack vector is a unilateral attempt or path by which an attacker can compromise a system

Attack Surface

Attack surface is the sum of all attack vectors which allows an attacker to gain access into the system

Attack Tree

It is a part of threat model that determines the flow of threat to target an asset.

Threat

A threat is a path by which an attack can traverse across boundaries in conjuncture with attack surface to compromise an asset.

Vulnerability

Vulnerability is the mechanism that allows an attacker to gain access to the system.

Exploit

Exploit is a tool or software that is designed to attack and take control of an asset using a vulnerability.

Boundary

A trust boundary is simply a representation of different access privileges a component has in a threat model.

6.4 Tools for Threat Modelling

There are various tools to perform threat modelling. It is dependent on comfort zone for an engineer to use the tool that represents all the attack surfaces and vectors. A few tools that can help are

- **Microsoft's Threat Modelling Tool** [7]
  This is a tool that heavily relies on representing components as processes data flow diagrams. It has all the data representation diagrams necessary to show how the data flows in the system traversing through various trust boundaries. Once the diagram is completed it allows a report generation based on the properties set for each data flow, process, boundaries etc. This is free to use tool.
- **Microsoft Visio**
  Visio is one of the best tools available for performing threat modelling as it is not restricted to Gane Sarson representation of Data flow diagrams. Users can freely mix architecture and threat modelling concepts to represent a model that allows them to express the product design and security workflows. It also has layers to hide specific components that allows deeper threat assessment on individual module.

There are various other free tools which users can use to perform threat model assessments on their products.

6.5 Frequency and Benefits of Threat Modelling

There is no hard-set standard or timeframe to the frequency of threat modelling. It takes time to understand different threats and how they affect your system. The idea to incorporate threat modelling within a product lifecycle is to gradually ramp up the threat modelling activity frequency. Security engineers or architects can gradually ramp up threat modelling as per release to per sprint. Keeping threat models up-to-date with attack vectors from externally reported vulnerabilities improves the overall security posture of the product. Threat modelling allows engineering to harden the product against various attacks thereby securing without compromising on quality or performance. Having a good documented threat model also allows engineering to catch early defects in design and architecture before code is even written thereby saving cost and time.

7 Security Standards and Tools

This section will provide a glance of various security tools and their usage and few coding standards.

7.1 Coding Standards

Coding standards is probably followed in every development team. Generally, there are different types of standards that every engineering team adopt when developing a product. They are not limited to code style, function prototyping, naming conventions etc. Secure Coding Standards are specifically guidelines that enable developers to make informed decision on to how to securely load sensitive data into variables and how data should be handled.

Every language has its own set of secure coding standards and discussing all the standards is beyond the scope of this paper. SEI CERT has a confluence website [8] set up where the standards are constantly updated and new ones being worked out. It is a good starting place to deep dive into secure coding standards.

In C or C++, Secure Coding standards can be very simple enforcement like initializing variables to a value to prevent garbage value interfering with application logic or can be very complex not limited to the programming language or platform like how passwords or keys should be handled in memory. SEI Cert has all the language specific standards covered.

When dealing with passwords or keys within memory, we must explore into native support on how memory regions can be protected so that debuggers don't attach themselves and start analysis memory for patterns like username etc. User authentication information should be encrypted for maximum amount of time even though they are stored in memory. A simple example would be how we would have a structure or class that stores username and password in an object. It might seem very plain and harmless, but when you take a memory dump either by attaching a debugger to the process or analyzing a full memory dump, all the attacker needs to look for is a username and start investigating the memory blocks around a username password to extract the password. It is impossible to protect secret keys and passwords as we will end up in a chicken and egg problem as to where to store the key for the memory encryption. The goal of security when dealing with secure information handling is to make it harder for an attacker to look for the information. Secure Zeroing the memory where such sensitive information also adds to the coding practice of securing the memory if it was reallocated to some other pointer.

7.2 Security Tools

There are a bunch of security tools that help with providing data and a collation of these data help penetrate a system. Detailing over each tool is not in the scope of the paper, we will mention some of the most popular tools and their uses and how they help with penetrating a software. For detailed documentation, you can refer to the respective tools website.

7.2.1 Network Tools

7.2.1.1 NMap [9]

NMap is a network scanner that is flexible, powerful, free and easy to use, to scan networks. It uses raw sockets to do a variety of discovery, protocols used, the ports open on a target machine and to a level identify applications running on the open ports.

7.2.1.2 WireShark [10]

Wireshark is a network sniffer tools that allows an attacker to monitor network traffic. Unsecured data over TCP or any protocol can be viewed using this tool.

7.2.2 System Tools

The list provided here are specific to Windows OS.

7.2.2.1 SysInternalSuite [11]

SysInternalSuite is a set of tools that is currently offered by Microsoft. It has a whole bunch of tools that allows monitoring, collect data or and launch programs with escalated privileges. All of the tools

7.2.2.2 ProcMon

Process Monitor is a tool within the SysInternalSuite that needs a special mention because of the trove of data that it collects on all process. Most system calls are captured and security attacks like DLL sideloading can be analyzed from such logs.

7.2.2.3 Dependency Walker

Dependency Walker is a tool that lists out all the DLL imports and the all the exports that a DLL makes. It is useful to analyze the DLL dependency tree.

7.2.2.4 Process Explorer

Process explorer is a tool within the SysInternalSuite that allows exploring all the process, the threads, the call stack in each thread, the resources opened by a process, DLL handles, Event handles, namedpipes etc.

7.2.3 Vulnerability Scanner

7.2.3.1 Nessus [12]

Nessus is one of the popular vulnerability scanners that is regularly updated with vulnerabilities and uses it to scan machines in a network for unpatched vulnerabilities.

7.2.3.2 Microsoft Baseline Security Analyzer [13]

Microsoft Baseline Security Analyzer provides a streamlined method to identify missing security updates and security misconfigurations.

7.2.3.3 NetSparker [15]

NetSparker is a web application security scanner, with support to detect vulnerabilities in web applications.

7.2.4 Exploit Tools

7.2.4.1 Metasploit [14]

Metasploit is an advanced open-source platform for developing, testing and using exploit code. It also provides a purposeful insecure linux environment to test metasploit and other tools.

7.2.5 Pentest Framework

7.2.5.1 Kali Linux [16]

Kali Linux is a debian based linux distribution aimed at advanced penetration testing and security auditing. It includes more than **600** penetration testing tools and is free.

8 Conclusion

In this paper, we went over the basics of what software security is, what are the fundamental differences between different lifecycle models and how security activities can be embedded in product lifecycle. One of the key takeaways is to breakaway the false notion of compromising quality or performance for security and vice versa and have a structured process to build in security right into a products lifecycle as it moves through different stages. We explored into the vocabulary of threat modelling and what threat modelling is all about with different methodologies and glanced through few of the tools that are commonly used to penetrate a system. The paper will provide a good starting platform for product teams to adopt security within their respective lifecycle for the first time or improve their existing methodologies.

9 References

1. Waterfall Model, https://en.wikipedia.org/wiki/Waterfall_model
2. Agile Model, https://en.wikipedia.org/wiki/Agile_software_development
3. DevOps Toolchain, https://en.wikipedia.org/wiki/DevOps_toolchain
4. Successful Git branching model, http://nvie.com/posts/a-successful-git-branching-model/
5. Microsoft STRIDE Threat model, https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx
6. Process for Attack Simulation and Threat Analysis, https://www.owasp.org/images/a/aa/AppSecEU2012_PASTA.pdf
7. Microsoft Threat modelling tool, https://www.microsoft.com/en-in/download/details.aspx?id=49168
8. Secure Coding Standards for most high-level languages, https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards
9. NMap, https://nmap.org
10. Wireshark, https://www.wireshark.org
11. SysInternalSuite, https://docs.microsoft.com/en-us/sysinternals/downloads/sysinternals-suite
12. Nessus, https://www.tenable.com/products/nessus-vulnerability-scanner
13. MBSA, https://www.microsoft.com/en-in/download/details.aspx?id=7558
14. Metasploit, http://www.metasploit.com
15. Netsparker, https://www.netsparker.com
16. Kali Linux, https://www.kali.org
17. Brook's RSA Conference presentation, https://www.rsaconference.com/writable/presentations/file_upload/lab3-w04_threat-modeling-demystified.pdf
18. Brook Schoenfield's book, **Securing Systems: Applied Security Architecture and Threat Models**.