Hardening your Development Process

Nicolas Guini – Andres More

{nicolas_guini, andres_more2}@mcafee.com

Abstract

Every day, we can find new software security issues, exploits and new ways to misuse the software to do things never thought. Security criminals look every day at the software we produce and try to find a simple breach to exploit it and gain control to use it as they wish.

Many times, the security tests are not considered because it is difficult to see results, it is difficult to implement, or it costs time and money. But, when attackers find security flaws in our software, we regret the decisions made previously and we wish to have checked again with a security perspective.

For these reasons, it is important to add controls and security checks in our development process. The application of the Secure Development Process (SDL) added to a Security Maturity Model (SMM) helps our development teams find security infractions in the early stages before the software comes out into the open.

The purpose of this paper is to review an introduction to the life cycle of secure development and how it can be improved by verifying it with a measure of the security maturity model. We discuss how this process helps to find security problems in a new feature development. And a practical application of lessons learned to make this process faster and easier for development teams.

Biography

Nicolas Guini is a Software Security Architect at McAfee Argentina. Nicolas has a Computing Engineer Degree where worked with Security Tools for pentesting, a Specialization on Computer Security and two Ethical Hacking Certifications. Lecturer at OWASP and local universities.

Andres More is a Software Product Architect at McAfee Argentina. Andres has a CS degree where implemented mandatory security control at Linux's network stack, and a MSc in High Performance Computing where developed infrastructure to support performance optimization. Lecturer at OWASP and postgraduate courses in security at a local military institution.

1 Introduction

This section introduces software security and how it should be incorporated in a software development lifecycle, considering an agile implementation. Following sections go into detailing best practices around security on each different development activity.

1.1 Software Security

Software security is the idea of developing software so that it continues to function correctly under malicious attacks of vulnerabilities. Security is necessary to provide integrity, authentication and availability as non-functional requirements.

Software security best practices leverage good software engineering and involve thinking about security at early stages of the software development lifecycle, considering and identifying common threats, designing for security and subjecting all software artifacts to thorough objective risk analysis and testing.

Security and privacy perspectives should be included in all software and application development practices. Product security policies and processes must be applied, to proactively find and remove software security defects or vulnerabilities as early as possible.

1.2 Secure Software Development Lifecycle

Software Development Life Cycle

The Software Development Life Cycle (SDLC) is a process used by software engineers to design, develop and test high quality software [9]. It describes the stages and tasks involved in each step of a project to write and deploy software. A mature process should be documented so it can be properly communicated, incorporated, measured and ideally improved over its own lifecycle.

Secure Development Lifecycle

The Security Development Lifecycle (SDL) is a software development process that helps developers build more secure software and address security compliance requirements while reducing development cost [1], [4], [8]. The goals of SDL therefore are twofold:

- Reduce the number of security vulnerabilities and privacy issues.
- Reduce the severity of the vulnerabilities that remain unaddressed.

Agile SDLC

An agile SDLC favors collaboration thru frequent and incremental releases, made by self-organizing teams in direct interaction with end users, it was made popular after a manifesto were customer satisfaction is the top priority [11].

An agile implementation of a SDLC following the scaled agile framework [10] is depicted in Figure 1.

Investment themes are used to drive definition from a high-level plan of intent until concrete development stories, refining work items from a program backlog into a team backlog. Team ceremonies including daily syncs, sprint reviews and retrospective sessions drive the team pace on shippable increments until a finished product is completed.

Agile SDL

For a new product, the security process typically begins at project initiation. A seasoned security architect assesses a proposal for its security implications. The output of this engagement is any additional security

features that will be added for software self-protection so that the software can be deployed in accordance with the different security postures.

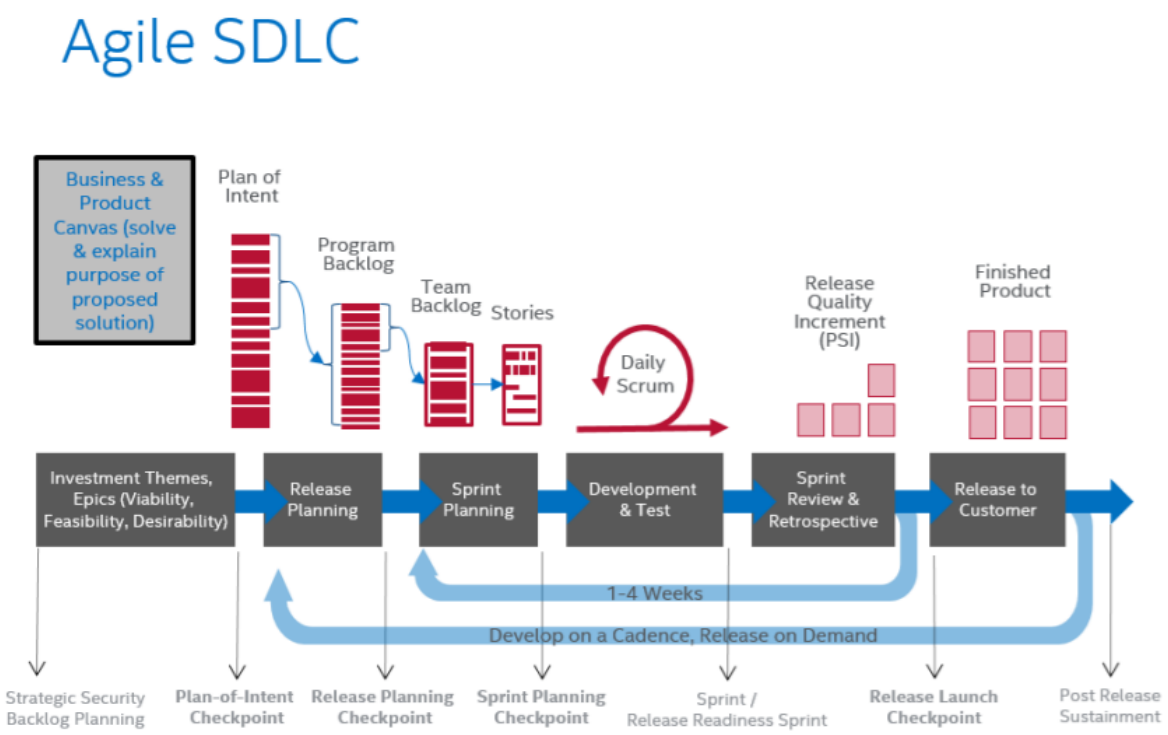


Figure 1 - Agile SDLC

Any project that involves a change to the architecture of the product is required to go through a security architecture review. The proposed architectural changes are analyzed for security requirements, as well as analyzed within the whole of the architecture of the product for each change's security implications. A threat model is created or updated. The output of this analysis will typically be the security requirements that must be folded into the design that will be implemented. An architecture review may be a discreet event or may be accomplished iteratively as the architecture progresses (Agile).

The Agile SDL requires that designs that contain security features or effects are reviewed to make sure that security requirements will be built correctly. The Security Architect signs off when the design meets expectations. All functional items, including security design elements, are included in the thorough functional test plan. Like architectural reviews, a design review may be a discreet event or may be accomplished iteratively when design work occurs (Agile).

In tandem with architecture and design reviews, privacy reviews are conducted. A Privacy Impact Assessment (PIA) is performed to determine if any additional privacy activities are required to protect personal data. Privacy reviews cover the whole lifecycle of personal data and often extend beyond the product collecting the data and include backend systems and infrastructure.

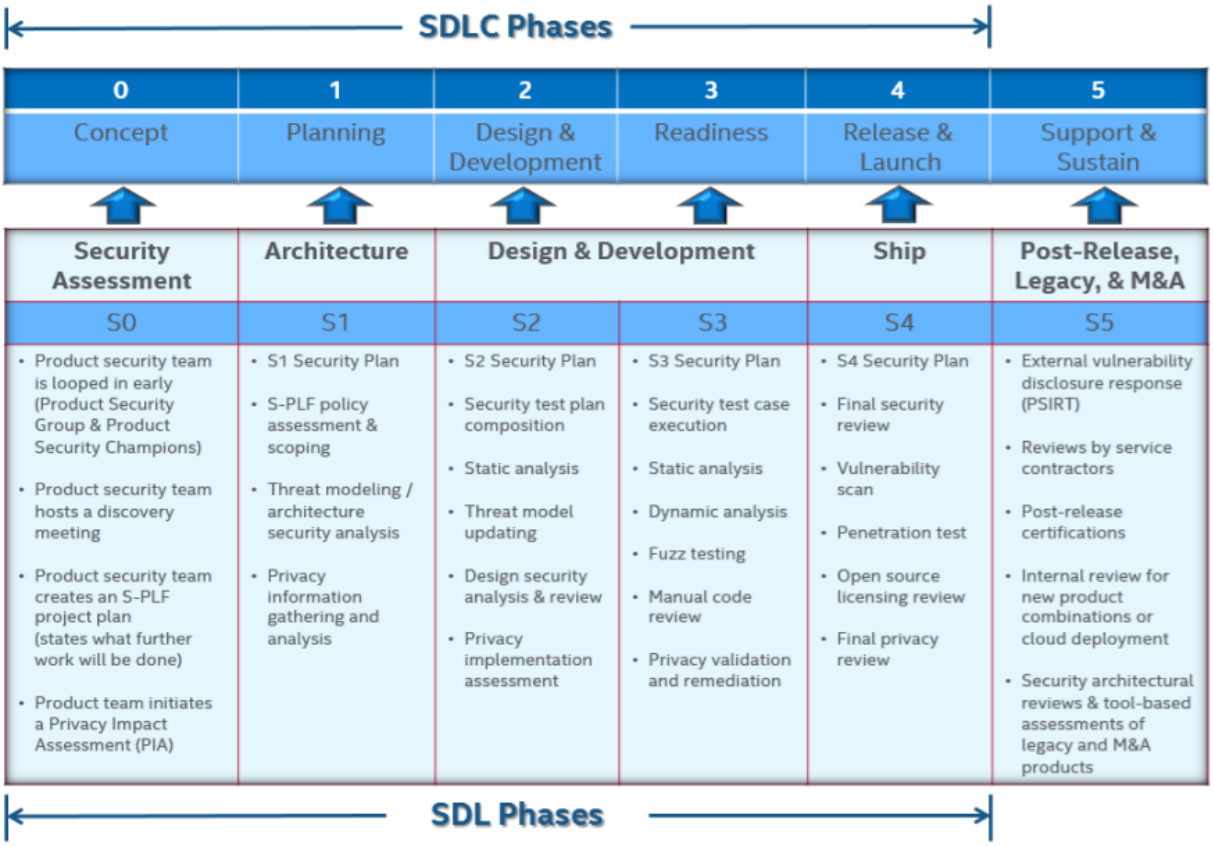


Figure 2 - Waterfall SDL and Agile SDLC security tasks

The above chart shows how can a traditional Waterfall SDLC can be adapted to Agile methodology and security tasks can be applied on each phase. Integrated into Secure Development, security and privacy tasks are executed as seamless and holistic process designed to produce software which has an appropriate security and privacy attributes built into it.

The set of security tasks can be applied either as a Waterfall development or in Agile development, performing them across the iterations. For Continuous Integration, SDL activities are determined by certain triggers which are set by milestones, events, and time intervals.

Security Maturity Model

A Security Maturity Model applied to SDL is a multi-layered measurement approach that can be applied to any product to understand security practices. Is not different from other maturity models [12] but having security-related practices as the focus.

The McAfee Product Security Maturity Model [2] is how McAfee evaluates developing team's work and procedure related to implemented tasks, activities and findings before, during and after the development process. The McAfee's PSMM covers around 16 topics which are evaluated in all aspects related to secure development, since new initial requirements are defined, architecture and design of new changes are reviewed, the whole process is monitored, and final checks are executed.

Post-release activities should be discussed independently when considering security, as vulnerabilities on the field might impact not only the product but the customer relying on it. McAfee establishes a company-wide Product Security Incident Response team (PSIRT), who oversees all answers of product security concerns to customers including vulnerability handling.

Security Bulletins (SB) are security documents that ISVs publish when they fix some vulnerability on their products. An SB contains a brief description of each fixed vulnerability including affected products. Fixes and their release date are specified sometimes including workaround mitigations until patches are applied. McAfee follows this practice as well [3].

2 Tools and Automation

This section describes what and how security can be considered on each different task to be completed during process development stages. Lessons learned are included so other team can leverage.

2.1 Concept

2.1.1 Security Training

Training is a tool that avoid bugs up-front as they are not even created. Training should be considered a never-ending activity delivered to the team incrementally. Each training increments should cover topics around:

1. *Security Domain*: the team should be familiar with the security domain, including corporate practices around managing products on the same vertical.
2. *Used Technology*: the team should know language or framework specific hurdles around potential vulnerabilities when not implemented properly.
3. *Security Tools and Processes*: eventually the whole agile team should understand how to run security scans, how to test security communications, etc. Every task with changes to be required again should be documented for the rest of the team, and documentation on tasks should evolve when the tasks reoccurs.

2.2 Requirements

2.2.1 Definition of Done (DoD)

The definition of done explicitly defines to the team members what needs to be completed before closing an assigned epic or story. From a security point of view:

1. Critical code should be reviewed by a senior or security SME (Subject Matter Expert)
2. All code changes should be statically analyzed, and found defects addressed
3. New and modified interfaces should be dynamically tested using scanners and fuzzers.
4. New or modified dependencies (such as 3rd party libraries) should be reviewed.

Following the agile concept, we have established a flexible approach for DoD as well. There are different levels: Good, Better, Best. For instance, an epic might be in a good state if functionally is completed but not all peer review findings are addresses yet. With regards to security, an epic might not be in best state until security testing has confirmed proper cryptography is being used in a new communication channel.

This incremental DoD follows the idea of tracking technical debts but does it without splitting the tasks and helping the team to avoid postponing resolving it. The incremental DoD also allows integrating components on a complex product to delivery incremental value altogether.

2.2.2 End of Life Components

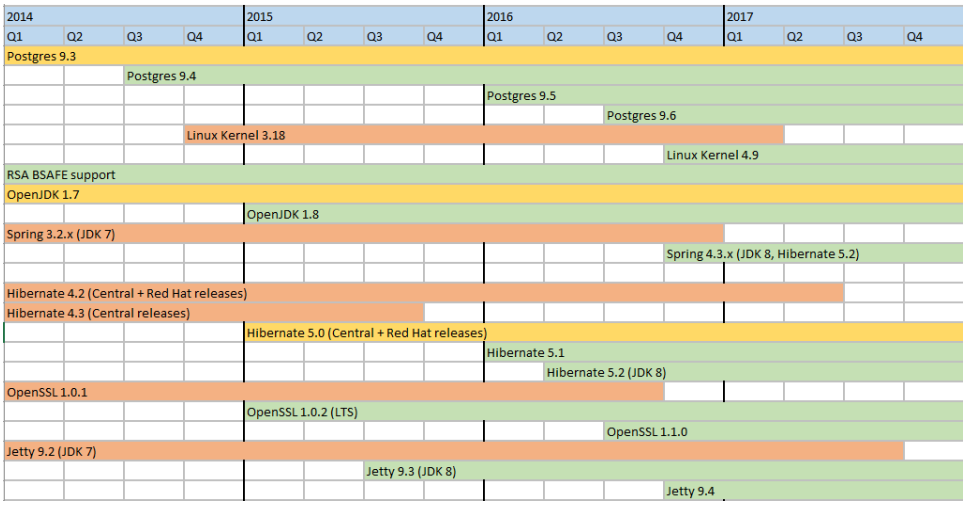
Most products today rely on multiple third-party components. Each component should be reviewed in search for vulnerabilities and signals of being out-of-maintenance. The company keeps track of whitelisted and blacklisted projects so the analyzed does not need to be repeated across teams.

Factors while analyzing the health of a dependency are:

1. User base (downloads, discussion posts quantity)
2. Frequency of releases
3. Known and patched vulnerabilities

For Java and Python based components we leverage and contribute into OWASP dependency checker [13], and recently added Safety [14] for Python as an early adopter in the community.

A roadmap of component updates should be keep updated and each product release should incorporate upgrades to avoid vulnerabilities.



This is an example of how EOL libraries and their versions can be tracked.

2.2.3 SDL Kanban

The security architect creates a storyboard or Kanban with all mandatory, required, and optional SDL activities. Those activities can be ordered by sprints, how they can be implemented, and ordered by importance and/or effort.

Once this is prioritized together by the team, it should be captured in the product backlog itself. During this phase a standard questionnaire is used to dimension the SDL work for the release, for instance:

1. Is the target release vehicle a major update or just contains minor maintenance changes?
2. It the product architecture changed?
3. Is there any change on dependencies?
4. Is any new PII information being handled at the product?

Explicitly listing what security-related work not will be included in the release has been found useful to direct effort and communicatee with stakeholders.

A discovery meeting involving stakeholders and reviewing target features is also held towards early discussion of security concerns and identification of next steps on each case.

2.2.4 Privacy Impact Assessment (PIA)

Feature review from privacy point of view. Detailed documentation of information being collected, persisted and ship to cloud-based services if that is the case. Also details about how long the information will be stored and documentation on how to delete them if required.

For instance, product telemetry is the usual feature being considered here, a spreadsheet with each particular data attribute is kept updated with field name, purpose, examples, etc. Product documentation

should clearly discuss what, when and why telemetry is sent. Product logging should include full telemetry payload as part of special log level to allow confirmation of telemetry content by customers. Opt-in and not opt-out.

2.3 Design

2.3.1 Security Plan

A security plan documents targeted tasks (also non-targeted ones) after prioritizing and ranking requirements. All tasks in scope are then added to the backlog thru user stories. The security plan should include user stories towards maintaining or increasing the product security maturity level.

2.3.2 Architecture and Design Security Analysis

A security architecture review fully documents the product's architecture. This architecture review is then used to produce the Security Design Review and Threat Model. These help to identify possible attack vectors early in the product development lifecycle before design. The security architecture review is intended to be a starting point to build security into the product.

2.3.3 Threat modeling

A threat model uses the security architecture to identify all possible attack vectors. The attack vectors that exposer areas of potential security exploitation weaknesses (a.k.a. vulnerabilities) are then addressed by defining countermeasures to prevent, or mitigate the effects of, threats to the system early in the design phase. Threat models are used to prioritize the threat/attack vector based on risk and business requirements.

To identify those weaknesses, many methodologies can be applied, among them STRIDE is widely used for this purpose. STRIDE means **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service and **E**levation of Privilege. Can be used to identify new threats answering typical questions around each component (or new integration).

It's recommended to divide the working project into multiple layers, from a very high-level scheme were the component under analysis is just a minimum part of the whole ecosystem, and then create new layers with more details (zooming in). Making an analysis of each layer and applying a detection methodology to each one, is the easiest approach to identify all possible vulnerabilities.

To reach this purpose Microsoft offers a specific tool, which is Microsoft SDL Threat Modeling Tool.

2.3.4 Privacy information gathering and analysis

Privacy protects the confidentiality, integrity and availability of personal and corporate data. In this stage known what kind of data will be collected, how long and where, it's necessary to be compliant with standards regulations and company policies.

2.4 Implementation

2.4.1 Secure Coding Standards

Ensure secure coding standards and best practices are used throughout the development of the product.

One way to achieve that is trainings the development team and adding plug-ins with custom information to the development IDE.

One tool that can be used is Checksyle. Checkstyle is a tool that can be installed as a plug-in on the development IDE and checks Java source code for adherence to a Code Standard or set of validation rules.

Findbugs is another tool that can be implemented on the development environment. Findbugs is an open source tool that executes a static analyzer over Java code.

PMD also can be used for this purpose, it's open source code analyzer which finds common programming flaws. Supports multiple languages like Java, C, C++, Python and Ruby.

2.4.2 Static analysis

Static analysis is a method of examining software for vulnerabilities at compile and/or build-time, reporting potential defects in source code. The analysis searches the code for known weaknesses such as buffer overflows, null pointer exceptions, among others.

A lot of tools exist on static analysis field, multi-language or specialized can be found on Internet. Coverity by Synopsis is one of them and can be integrated to a CI/CD plan, in a way that a developer make code changes and generates a new entry into the code repository. A dedicated Coverity servers is listening for new changes and if it found a new one, it triggers a static scan. Once the analysis finishes and new findings are discovered, it can trigger an email to the entire development team and assigned owners to fix them.

Other tools used for this purpose also are Checkmarx, Klockwork, Veracode. IDEs like Eclipse or IntelliJ also have integrated code inspection on compile time. A developer can install those IDEs, plus static code analyzer plug-ins such as Findbugs, to detect vulnerabilities, before submitting their code.

2.4.3 Threat Modeling (updates)

On this phase we recommend looking again the Threat Model in order to find new changes, missing vectors due to design changes.

We found that a threat model update on this phase also can be useful to make because architecture changes can be in place due to unexpected conditions or requirement changes.

2.4.4 New Tools Evaluation

In a mature development team, multiple tools can be integrated during the development pipeline. All tools already implemented in the CI/CD process helps at the beginning to get new findings and then to keep the quality code standards.

Every day new tools are created and shared into the market. Testing or implementing those tools helps not only to get new findings

During the implementation phase, new tools can be used and tested.

Running different tools vs always running the same one.

2.5 Verification

2.5.1 Dynamic analysis

Dynamic analysis tools find vulnerabilities in the binary executables and other runtimes, like Web applications at run-time, by executing the code over multiple test cases, reporting actual defects.

Most dynamic solutions test only exposed HTTP and HTML interfaces of Web-enabled applications, they're also called Web Application Scanning.

A difference to static code analyzers, dynamic tools need a running application. Most of tools implemented in this field are Web Application Scanners and for this purpose tools like Netsparker, ZAP Proxy, w3af, Burp Suite and homemade scanners can be used.

In case of Zed Attack Proxy Project (a.k.a. ZAP Proxy), it's an Open Source tool created by OWASP community helps to find security vulnerabilities in a web application automatically. This tool provides you an automatic web spider which helps you to collect the information and then make the analysis but, one of the most useful features that this tool provides, it is his proxy. Using the proxy, you can navigate thru your web page and collect only data for those modified web pages (or where the spider can't reach), and then analyze them. The tool also provides a Jenkins plug-in which can be installed.

2.5.2 Interactive analysis

Interactive analysis is a different form to test your application security that stems from a combination between dynamic and static analysis technologies. This approach analyzes leverages information from inside the running application, including runtime requests, data flow, control flow, libraries, and connections, to find vulnerabilities.

2.5.3 Fuzz Testing

Fuzz testing or "fuzzing" is a subset of dynamic analysis. It finds vulnerabilities and system crashes in the running binary executables by probing all inputs, network protocols, and file formats with intentionally invalid data. It permutates the packet boundaries with illegal formats to see if the running program will trigger an error condition or fault. These error conditions can lead to exploitable vulnerabilities.

We found that not all tools support automatic generation of payloads after a JSON Schema, only after JSON examples of the payloads.

2.5.4 Manual Code review

Security-focused manual code reviews involve humans to search for and find vulnerabilities and weaknesses in the source code.

Code review it's a fundamental step when a developer wants to add his own code to the main repository. Github has his own Code Review tool and it's widely used cross all development teams. CodeCollaborator helps to accomplish the objective and has the capability to integrate with other versioning software like SVN.

2.5.5 Vulnerability Scan

A vulnerability scanner is a tool used to find known vulnerabilities and exposures in a binary executable and the environment to which it is deployed and runs.

This stage where most of the tools can be found. The most used ones are based on environment scanner like OpenVAS by OWASP, Nessus and Nexpose.

2.5.6 Penetration Test

Pen testing requires sophisticated use of tools by human analysts to find and then prove vulnerabilities in executables and the environment in which they run.

On this stage mostly, custom scripts should be implemented and integrated with common tools and frameworks like Nmap, Metasploit.

We have made pre-build templates so team members can easily spin up new instances of the required tooling to perform security testing.

2.6 Release

2.6.1 Conduct Final Security Review

After the development a testing conduct a final security review helps to summarize and improve all security controls made during the development phase.

2.6.2 Retrospective

A retrospective review, from a security related perspective is useful. During this meeting, new tools can be discussed, how to track security tasks, know what was useless during the process, what can be improved or modified, what can be a good thing to start doing.

2.7 Response

2.7.1 External Vulnerability Disclosure Response (PSIRT)

It's very common that new vulnerabilities appear once the product is running in production. You can either document the vulnerabilities and their impact, and/or implement changes fixing the vulnerability,

A Security Bulletin is a document where the vulnerabilities are described and analyzed by the security experts. In this document information related to the vulnerabilities such as vulnerability ID, widely known as CVE, their impact on the software using an attack vector known as CVSS, and if it's a workaround or software fix update.

2.7.2 Post-Release Certifications

We have undergone multiple independently product certifications, as they are required by many verticals in the financial and aerospace industries.

FIPS requires evidence of using proper cryptography providers and their configuration, we recommend putting automation in-place to gather evidence without investment effort each time a dependency is upgraded. Performance gaps need to be considered and they are usually caused by lack of entropy pools being properly configured at platform level.

Common Criteria requires thorough architecture and design documentation. Finding an individual with experience in the paperwork is key.

FedRamp compliance is not a big difference when previous two certifications are completed, we have identified the requirement of having an antivirus and file integrity components in place at platform level as the main gap.

3 Conclusions

This section covers a quick summary of interesting talking points in previous sections.

The impact on having automation in place for defect identification and testing APIs and communication channel cryptography cannot be highlighted enough. Removing security issues in the code is cheap when they are identified earlier in the development process. Quick re-test of cryptography over communication channels allow the usual fine-tuning on default (and allowed for negotiation) algorithms and key sizes when undergoing compliance certifications such as Common Criteria and FIPS. The same happens with vulnerability scanners against dependencies, they should be hooked into build systems for best results.

The identification and tracking of EOL components helps planning of current and future releases, allowing to establish what not will be updated and how it would be mitigated if required up-front.

Having an agile mindset towards the application of DoD and the team training has allowed teams to harden components and add expertise incrementally and globally.

References

The authors would like to thank the Argentina Software Development Center for the provided support to make this research paper to happen. Also, the authors would like to recognize the product security expert community around McAfee who provided valued feedback on early and late drafts.

[1] Howard, Michael; Lipner, Steve (June 2006). The Security Development Lifecycle: SDL: A Process for Developing Demonstrably More Secure Software. Microsoft Press. ISBN 0735622140.

[2] James Ransome, Harold Toomey (October 2017). McAfee Product Security Practices. Available at https://www.mcafee.com/enterprise/en-us/assets/legal/product-software-security-practices.pdf.

[3] McAfee Product Security Incident Response Team (June 2018). McAfee Product Security Bulletins. Available at https://www.mcafee.com/enterprise/en-us/threat-center/product-security-bulletins.html.

[4] Microsoft (May 2012). Security Development Lifecycle, SDL Process Guidance Version 5.2. Available at http://www.microsoft.com/sdl.

[5] S. Lipner, "The trustworthy computing security development lifecycle," 20th Annual Computer Security Applications Conference, 2004, pp. 2-13.

[6] M. Howard, "Building more secure software with improved development processes," in IEEE Security & Privacy, vol. 2, no. 6, pp. 63-65, Nov.-Dec. 2004.

[7] Joshua Cajetan Rebelo, Patrick McEnany (October 2015), "Moving up the Product Security Maturity Model". Available at http://uploads.pnsqc.org/2015/papers/t-051_Rebelo_paper.pdf.

[8] NIST Special Publication (SP) 800-64, Revision 2, Security Considerations in the System Development Life Cycle

[9] IEEE Std 1074-2006 (Revision of IEEE Std 1074-1997) - IEEE Standard for Developing a Software Project Life Cycle Process

[10] Leffingwell, Dean (2007). Scaling Software Agility: Best Practices for Large Enterprises. Addison-Wesley. ISBN 978-0321458193.

[11] Jim Highsmith (2001). "History: The Agile Manifesto". Agilemanifesto.org.

[12] "CMMI for Development, Version 1.3". CMMI-DEV (Version 1.3, November 2010). Carnegie Mellon University Software Engineering Institute. 2010. Retrieved 16 February 2011.

[13] OWASP Dependency Check. https://www.owasp.org/index.php/OWASP_Dependency_Check.

[14] Safety. https://pyup.io/safety/