Open Source Software Security

A research summary

December 2020

Table of Contents

Executive Summary

This report is intended to offer a summary of some of the key information sources researched, so that the reader can relatively quickly gain an overview of open source software security considerations and undertake their own research into the referenced sources. This document is structured as a series of topic areas relevant to open source software security. Each topic has reference links and some direct copies of relevant excerpts.

Open source code has a number of advantages, notably including that source code is accessible and subject to inspection, a wide community of developers can contribute and there is potential to accelerate telco cloud implementation. There are various best practice steps that aim to 'make secure software' but none of these are mandated in the open source community whose main focus is functionality[1].

The report outlines two perspectives on managing and implementing change: *systems* and *component*. When considering security controls and mitigations, it is useful to assess these within these perspectives. The ultimate aim will be to identify those security controls that can be applied to mobile networks as they evolve and are upgraded. The underlying security principles are often described at the *component*-level. Dependent on the delivery model for network changes, it is entirely possible that this level of detail will require to be delivered through a third party systems integrator / lead vendor. In this case, consideration will be needed as to how to achieve the security *outcome* through *systems*-level steps such as requirement and design activities.

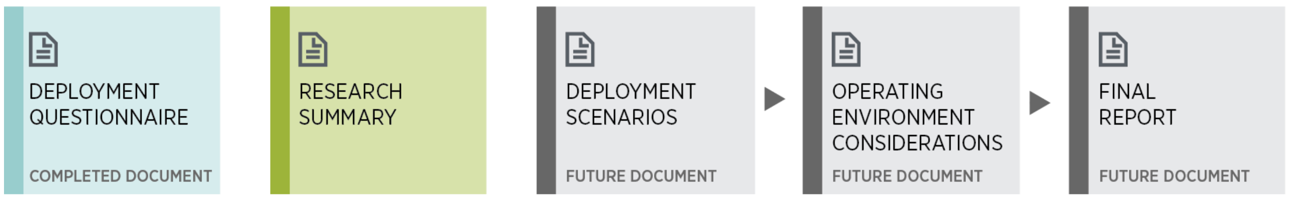
The next stage of activity will seek to combine these security considerations with the output from a parallel project seeking mobile network operator feedback on their considerations for open source software security to identify deployment scenarios and associated security considerations. A later stage of activity will consider broader system security considerations particularly in the infrastructure area. The longer-term aim is to build a set of best practice considerations for operators to consider when dealing with open source software. It is intended to complete this sequence of activity, however, it is recognised that this cycle may need further development as deployment scenarios and security controls evolve.

[1] https://www.linuxfoundation.org/wp-content/uploads/2020/12/2020FOSSContributorSurveyReport_121020.pdf

Welcome

Welcome to this initial GSMA report summarising research undertaken for the open source software security project as part of the broader open networking initiative. This report is intended to offer a summary of some of the key information sources researched, so that the reader can relatively quickly gain an overview of open source software security considerations and undertake their own research into the referenced sources.

The next stage of activity will seek to combine these security considerations with the output from a parallel project seeking mobile network operator feedback on their considerations for open source software security to identify deployment scenarios and associated security considerations. A later stage of activity will consider broader system security considerations particularly in the infrastructure area. The longer-term aim is to build a set of best practice considerations for operators to consider when dealing with open source software. It is intended to complete this sequence of activity, however, it is recognised that this cycle may need further development as deployment scenarios and security controls evolve. This sequence of activity is illustrated below:



DEPLOYMENT QUESTIONNAIRE	RESEARCH SUMMARY	DEPLOYMENT SCENARIOS	OPERATING ENVIRONMENT CONSIDERATIONS	FINAL REPORT
Canvassing 7 MNOs on their plans for deployment of open networks and open source software	This document describes and references a range of security controls for open source software that may be applicable dependent on deployment scenarios	This document describes the conclusions and considerations of the likely deployment scenarios	This document describes the wider operating environment and will focus on some of the broader security activity in the implementation stacks to position the software work in a broader context	This document consolidates all previous work into over-arching report (4) focusing on deployment models, systems and component considerations and pointing to the wider operating environment

This report and the Open Source Software Security project is undertaken as part of the broader GSMA Open Networking initiative where work is ongoing to define Minimum Viable Products and Use Cases. The Open Source Software Security project informs these models and provides high level considerations for security.

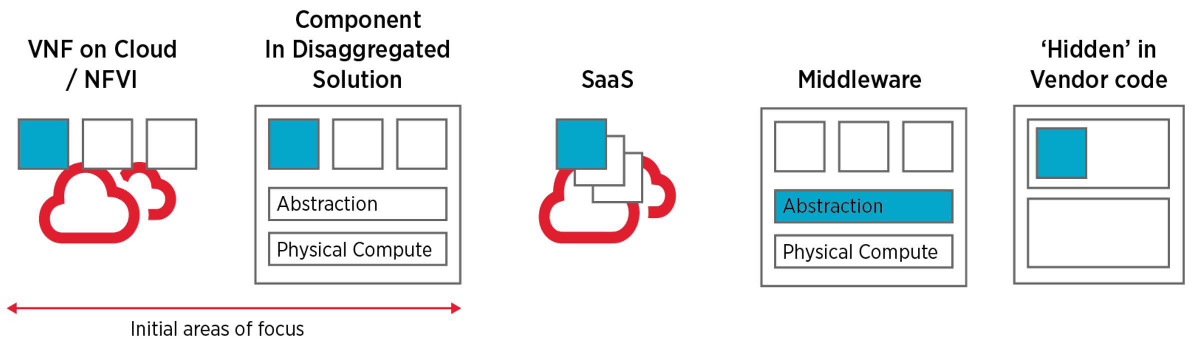
The following MNOs supported the open source software security project: AT&T, China Mobile, China Unicom, Hutchison MTN, Telefonica, and Vodafone.

Introduction

There are many initiatives driving open architectures and virtualised telecoms infrastructure such as Telecoms Infrastructure Project (TIP), Open-Radio Access Network (O-RAN) Alliance, Linux Networking Foundation, Open Networking Forum. The use of software from open source in a range of architectural deployments is rapidly increasing such as a software component running on virtualised infrastructure, to provide virtualised middleware, or within proprietary code implementation.

Open source code has a number of advantages, notably including that source code is accessible and subject to inspection, a wide community of developers can contribute and there is potential to accelerate telco cloud implementation. There are various best practice steps that aim to 'make secure software' but none of these are mandated in the open source community whose main focus is functionality[2].

Consideration has been given to the differing deployment arrangements for open source software as illustrated below.



Open source software may be applied in a wide range of ways including:

- as discrete code (such as a Virtual Network Function (VNF) running on top of Cloud / Network Function Virtualisation Infrastructure (NFVI),
- as a component within a disaggregated solution,
- as Software As A Service (SaaS),
- as the middleware within a disaggregated solution or 'hidden' or re-used within a vendor's proprietary code.

The initial focus of this work has been on the former two examples which may be considered relatively new to telecoms and will provide a best practice read-across to other deployment models. Operator engagement will be used to assess the likely deployment approaches that are envisioned in the immediate future. This may shift focus to middleware and re-use within vendor code.

The implementation of new capabilities can occur in varying levels of scale and at different parts of the network architecture. Accordingly, the processes for managing security will vary from a *systems* level approach (e.g. upgrade part of the RAN to a virtualised solution) to a more *component* level change (e.g. adding a new Virtual Network Function / Cloud-native Network Function). Operators may need to have different security approaches for both *systems* and *component* level network change so they can secure the open source software used within their networks. The components within open and disaggregated networks are changing as virtualisation of infrastructure increases. The move from physical integrated network functions towards virtualised and cloud-native network functions running on virtualised infrastructure introduces new security considerations. Virtual machines, hypervisors and containers will enable these newer virtualised functions to be deployed and their deployment stacks must be properly secured.

[2] https://www.linuxfoundation.org/wp-content/uploads/2020/12/2020FOSSContributorSurveyReport_121020.pdf

There are consistent themes across these references including a lifecycle approach to structured development. For open source code, some of the early stages (including Coding) of any lifecycle approach are undertaken by the developer community and our ability to influence security outcomes is limited. Instead, focus can be considered across the rest of the lifecycle and in the longer-term by beginning to influence developed education and skills development. These component-level lifecycles may have a short cycle time due to emerging approaches such as Continuous Integration / Continuous Deployment (CI/CD) and more integrated Development, Security & Operations (DevSecOps). Some sources in this document focus on 3rd Party Code which represents a superset of open source so there is a strong read-across to our problem area.

Background

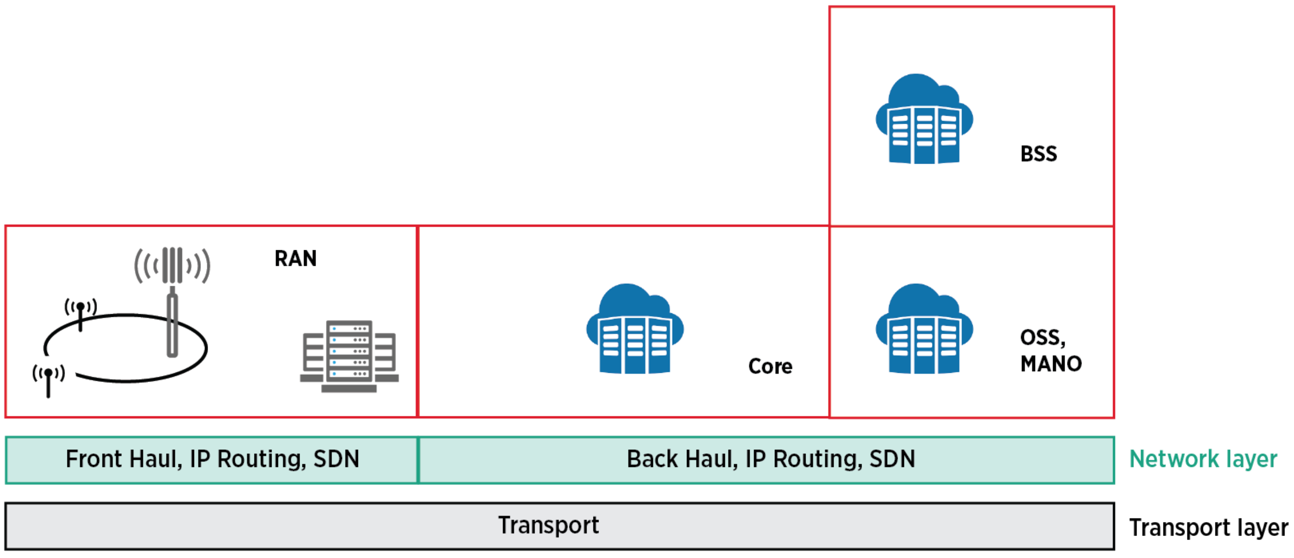
When a set of best practices is gathered, analysed and agreed, they can be used by operators to drive strong security outcomes for their open source deployments. There is opportunity to feed in our current knowledge of best practices to recognise their value and drive cross-industry security outcomes; essential in our inter-connected networks. Adopting a consistent set of practices across operators can allow quicker, better and more consistent security outcomes and learning and improvements can also be delivered. The best practices could deliver a framework for end-to-end security throughout the life of the operational systems.

The plan is to use this research to identify lifecycle activities which will drive good security outcomes when using open source software in virtualised networks. Each activity will be mapped to a specific lifecycle step, e.g. 'Requirement Definition', 'Test', 'In-life'. As open source software is available for inspection, static and dynamic code testing may be an appropriate activity within 'Test', as might Penetration testing be. Example lifecycles are illustrated in the Background section (below).

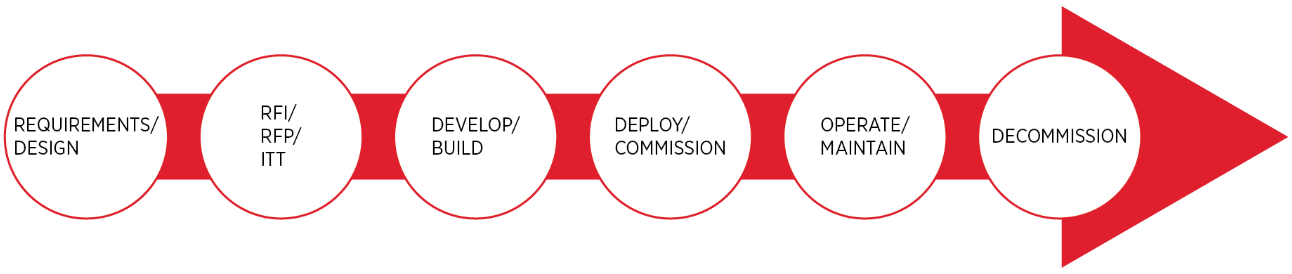
The report outlines two perspectives on managing and implementing change: *systems* and *component*. When considering security controls and mitigations, it is useful to assess these within these perspectives. For example, if consideration is being given to mandating the need for a Software Bill of Materials (SBOM) then this might be achieved at the systems level through a contractual requirement placed on a systems integrator, whilst at the component level this might be achieved through implementing a software composition analysis tool. The same control is achieved through differing means but to achieve the same goal. The details developed in the Research section aim to help identifying these controls, before we can consider how to effect these with a systems or component manner.

Systems-level Approach

The level of change (procurement, implementation) is at the *systems* level where the detailed view of change is indirectly controlled through higher-level activities. Change happens over a longer period (months) and applies to parts or all of a mobile network (e.g. RAN, Transport, Core etc.).



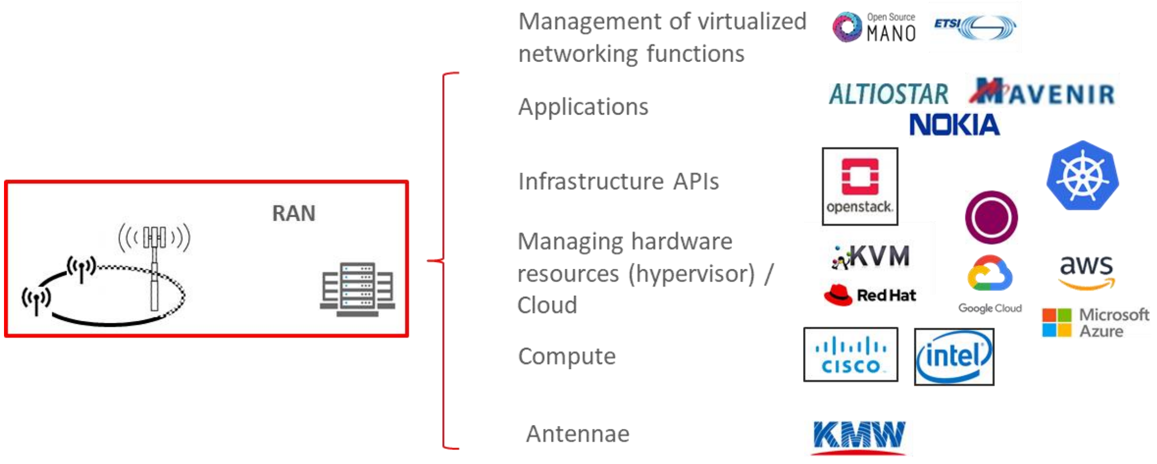
The process steps through which change is effected are illustrated below in two lifecycle examples. There are a variety of actions at each step that help deliver security outcomes. We aim to capture a set of best practice security steps to take at each stage that will drive a set of strong security outcomes which are linked to a component-level change cycle (described later).



Security considerations can and should be built within this lifecycle, aligned to its progress or through other approaches such as gate approvals.

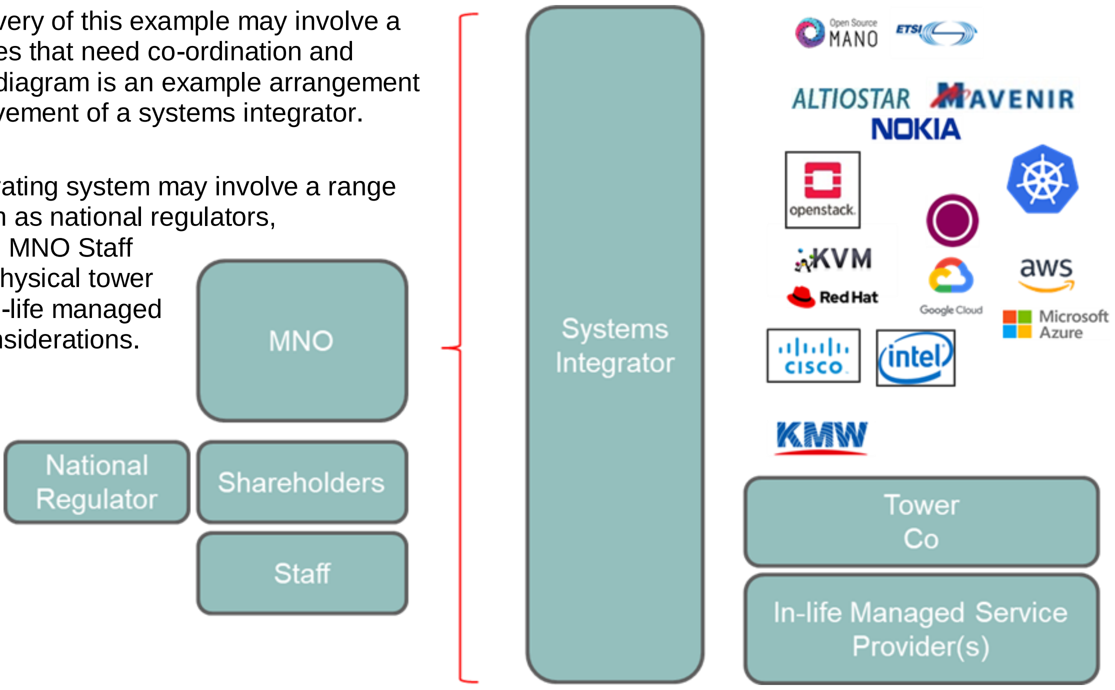
In practice, delivery can be managed directly by the operator or by engaging a systems integrator (SI) or lead vendor. Using a systems integrator can provide a single point of responsibility for delivery and integration of different vendor equipment / software / services etc but can also form a long-term in-life reliance on the SI for support and maintenance etc. A more direct approach exposes increased detail of integration operations, direct risk management of deployment issues and provides in-house expertise for the system when in-life.

For example, implementation of a new RAN solution may involve several technology providers. Example non-exhaustive set of providers illustrated below.


RAN
Management of virtualized networking functions
Applications
Infrastructure APIs
Managing hardware resources (hypervisor) / Cloud
Compute
Antennae

To manage the delivery of this example may involve a wider range of parties that need co-ordination and management. This diagram is an example arrangement illustrating the involvement of a systems integrator.

In practice, the operating system may involve a range of other parties such as national regulators, MNO Shareholders, MNO Staff and skill interests, physical tower infrastructure and in-life managed service provider considerations.

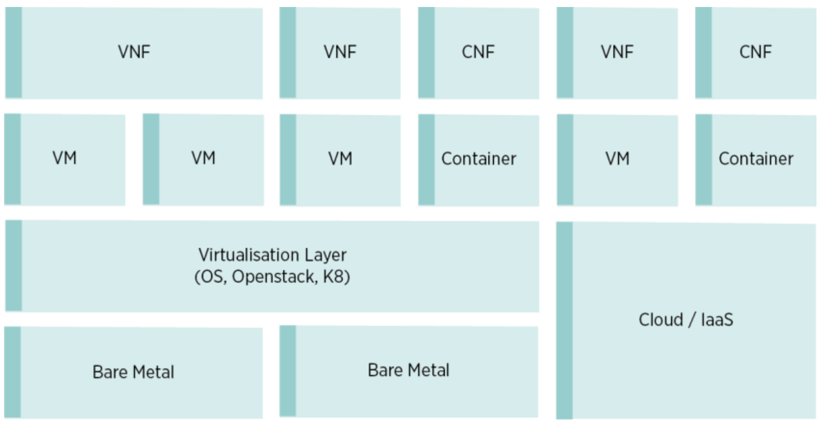


Component-level approach

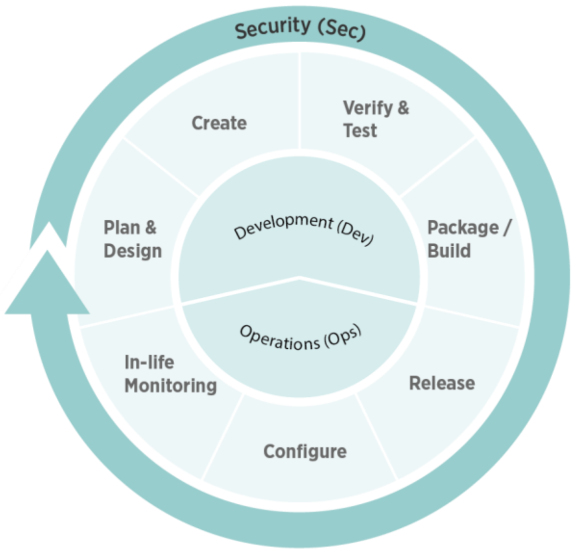
The level of change (detailed capability and code delivery) is at the *component* level where the detailed level of activity is influenced directly.

Non-exhaustive examples of these components are:

- A Virtual Network Function
- A vRAN Control Unit
- A Cloud-native Network Function
- A middleware virtualisation layer



These changes can happen quickly (days) with different versions of code developed on collaborative platforms and lifecycles may have a short cycle time due to emerging approaches such as Continuous Integration / Continuous Deployment (CI/CD) and more integrated Development, Security & Operations (DevSecOps) or (DevOps; shown below).



Examples of the process steps through which change is effected are illustrated below. There are a variety of actions, tools and best practice that help deliver security outcomes at each step. We aim to capture a set of best practice security steps to take at each stage that will drive a set of strong security outcomes.

Research Topics

The following Research Overview offers a series of relevant security considerations. These are largely technical in nature. Broader security considerations will form part of any overall control set: these might include physical, personnel and procedural security controls. These aspects are not explored in this paper as they will be addressed in future work. The table below conveys some of the recurring themes in the research and is developed to convey the interest for this document.

Concept / Organisation	NIST	NTIA	Safecode	OWASP	OASIS	LNF	NCSC (UK)	SSA	SynOpSys	BCS	BSA	Whitesource	BSIMM	IEEE	ENISA	CNCF	CIS	DARPA	ATIS	CNTT
Zero Trust	🟥														🟥			🟥		
SBOM / Inventory /Config Management		🟥	🟥						🟥		🟥	🟥	🟥			🟥	🟥			
Source Code Analysis	🟥			🟥		🟥		🟥	🟥			🟥				🟥				
Lifecycle approach	🟥	🟥	🟥	🟥				🟥	🟥	🟥	🟥		🟥		🟥		🟥			
Threat / Risk Model	🟥		🟥	🟥			🟥	🟥	🟥	🟥			🟥		🟥				🟥	
System and Component Approach			🟥				🟥													
Approved' Code / Re-use / Approve list	🟥		🟥	🟥	🟥					🟥										🟥
Secure Coding / Avoid vulnerabilities			🟥	🟥		🟥	🟥	🟥			🟥		🟥		🟥	🟥	🟥			
Security Requirements (&Test)	🟥			🟥											🟥	🟥	🟥			
Open source licensing						🟥			🟥											
Cyber secure design				🟥			🟥							🟥				🟥		
Cloud / Virtualisation Security							🟥									🟥				
Secure by Default	🟥						🟥						🟥	🟥			🟥			
Toolchain	🟥						🟥					🟥					🟥			
Dev(Sec)Ops				🟥											🟥					
Pentest	🟥						🟥										🟥			
Least Privilege	🟥						🟥						🟥				🟥			
Privileged Access Management	🟥						🟥						🟥				🟥			
Vulnerability Management	🟥						🟥		🟥				🟥				🟥			
Recovery / Incident Management	🟥						🟥										🟥			

The table is illustrative and on reviewing the detailed activities, one might find more detailed alignments for each organisation. There are a number of recurring themes (such as adopting a lifecycle approach, Software Bill of Materials, using threat / risk modelling, using secure coding techniques) and also some interesting more unique considerations (such as Zero Trust, DevSecOps and protecting the development toolchain). These considerations can form the basis for a set of additional component / systems security approaches. The precise approach for each area will vary depending on the implementation method. For example, many of the considerations are appropriate for direct application at the component level but naturally lend themselves to a translation applicable at the systems level. The organisations themselves are abbreviated and full details are available in the next section. The organisations have been chosen to provide a range of views from differing geographic regions.

Research Overview

Document Structure: This document is structured as a series of topic areas relevant to open source software security. **Each topic has reference links and some direct copies of relevant excerpts. In this section, the black text is taken from the referenced source. GSMA comments are included in a call-out box like this.**

The underlying security principles are often described at the component-level. As discussed earlier, dependent on the delivery model for network changes, it is entirely possible that this level of detail will require to be delivered through a third party systems integrator / lead vendor. In this case, consideration will be needed as to how to achieve the security *outcome* through systems-level steps such as requirement and design activities. Alternatively, should the operator choose to act as their own integrator then these principles can be more directly applied. The intention is this research can be used to develop a model of best practice that can be applied at systems and component level, as required.

For example, there may be a requirement to have an explicit record of the software composition or 'Software Bill of Materials' (SBOM). SBOM is described and referenced within the NTIA section below. This may be discharged at the system level through ITT requirements to a vendor that a standards-compatible SBOM exists and is maintained. Alternatively, at the component level it might be by selecting a suitable SBOM tool, using it on the selected code and maintaining the build as the code changes over its lifetime.

National Telecoms and Information Administration (NTIA)[3]

A strong set of principles that can be aligned to a lifecycle approach. A number of SBOM tools are identified.

NTIA's SBOM project began in 2018 to focus on dependency tracking (in 6 fields) including a Proof-of-Concept in the Healthcare sector. More work is planned on code integrity (signed-code). SBOM specifications include Software Identification (SWID), Software Package Data eXchange (SPDX) and OWASP CycloneDX.

Report: A Survey of Existing SBOM Formats and Standards

A report describing the current SBOM approaches and a comprehensive list of approaches / tools.

The working group identified two formats in widespread use: Software Package Data eXchange (SPDX), an open source machine-readable format stewarded as a de facto industry standard by the Linux Foundation, and SWID, a formal industry standard used by various commercial software publishers.

SPDX, a product of the open source software development community, is geared for ease-of-ingestion within a developer workflow. The open source nature of the format, as well as the availability of open source tooling to generate it, supports broad adoption by a large and distributed population of commercial international organizations, as well as developers who may not be associated with vendors. The accessibility of SPDX means that the sole developer of an experimental library can generate an SBOM with minimal effort at no cost. These cost savings and ready availability of open source tools is attractive to commercial organizations as well. SPDX is useful in the "long tail" of upstream open source software componentry.

[3] https://www.ntia.doc.gov/SoftwareTransparency

How to produce SBOM? Information that goes into SBOMs can be best obtained from the tools and processes used in each stage of the software lifecycle (See Figure 1, below). One may leverage existing tools and processes to generate SBOMs. Such tools and processes include intellectual property review, procurement review and license management workflow tools, code scanners, pre-processors, code generators, source code management systems, version control systems, compilers, build tools, continuous integration systems, packagers, compliance test suites, package distribution repositories and app stores.

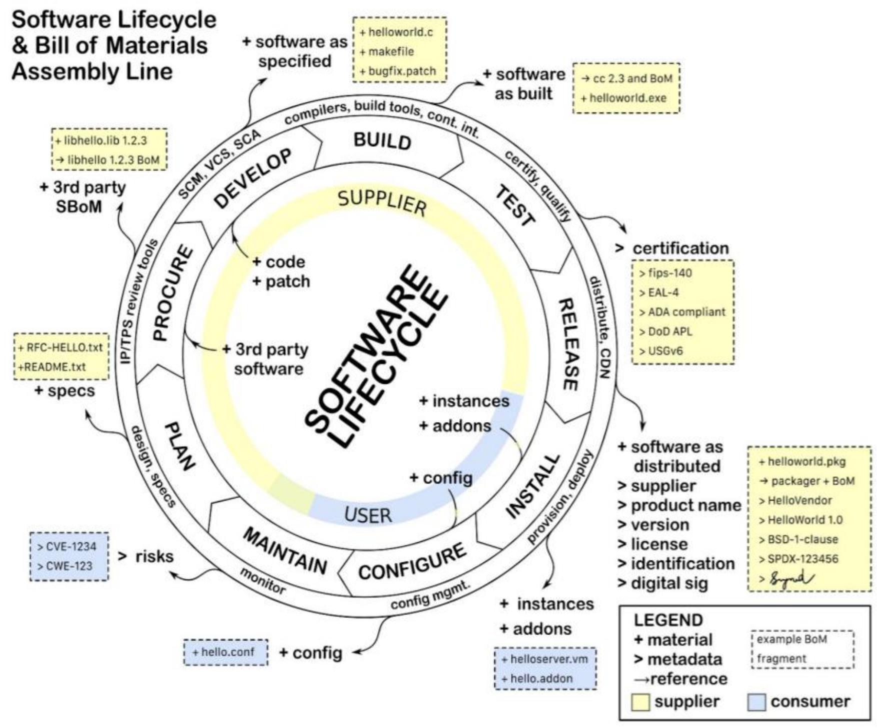


Figure 1: The Software lifecycle with multiple stages where underlying code might change, and thus the SBOM would be updated to reflect the changes.

CycloneDX is a lightweight SBOM specification designed specifically for software security requirements and related risk analysis. The specification is written in XML with JSON in development. It's designed to be flexible, easily adoptable, with implementations for popular build systems. The specification encourages use of ecosystem-native naming conventions, supports SPDX license IDs and expressions, pedigree, and external references. It also natively supports the Package URL specification and correlating components to CPEs.

Report: Roles and Benefits for SBOM Across the Supply Chain

Most software includes other software. Software changes and evolves over time due to optimization, new features, security fixes, and so forth. As a result, software producers throughout the supply chain have to continually evaluate how changes might impact their software. This includes changes to 3rd-party components used to compose software. How can organizations make confident, informed decisions? How can they manage the complexity of their software supply chain in a sustainable manner? In a complex supply chain, roles can blur.

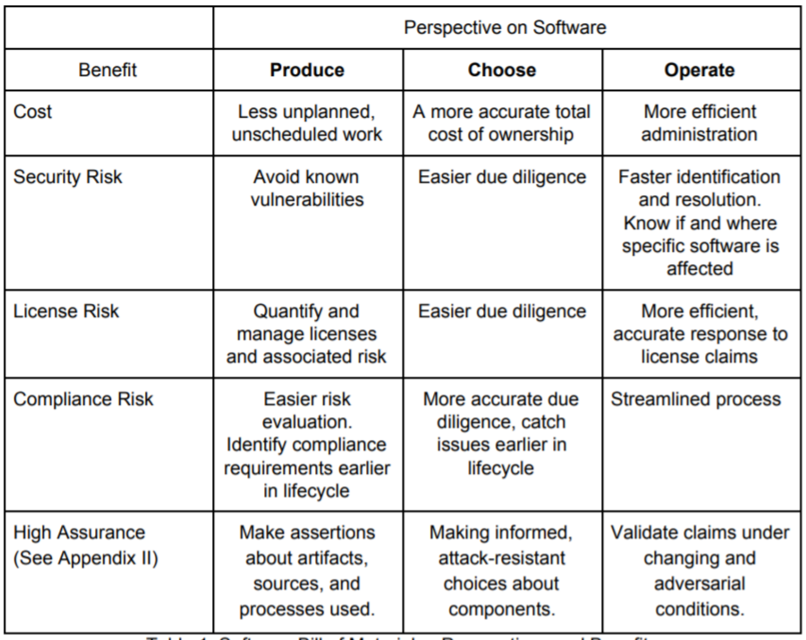
For simplicity, we will initially describe the software supply chain from three perspectives:

● I produce software - the person/organization that creates a software component or software for use by others [write/create/assemble/package]

● I choose software - the person/organization that decides the software/products/suppliers for use [purchase/acquire/source/select/approve]

● I operate software - the person/organization that operates the software component [uses/monitor/maintain/defend/respond]

This list is not a comprehensive list of perspective, there are other roles such as auditors, insurers and such who will also benefit from an SBOM as it matures and the various use cases evolve. These three perspectives are summarized in the following table:



	Perspective on Software		
Benefit	**Produce**	**Choose**	**Operate**
Cost	Less unplanned, unscheduled work	A more accurate total cost of ownership	More efficient administration
Security Risk	Avoid known vulnerabilities	Easier due diligence	Faster identification and resolution. Know if and where specific software is affected
License Risk	Quantify and manage licenses and associated risk	Easier due diligence	More efficient, accurate response to license claims
Compliance Risk	Easier risk evaluation. Identify compliance requirements earlier in lifecycle	More accurate due diligence, catch issues earlier in lifecycle	Streamlined process
High Assurance (See Appendix II)	Make assertions about artifacts, sources, and processes used.	Making informed, attack-resistant choices about components.	Validate claims under changing and adversarial conditions.

Table 1: Software Bill of Materials - Perspectives and Benefits

A Software Bill of Materials (SBOM) can help a software supplier produce their software in the following ways:

● reduce unplanned, unscheduled work
● reduce code bloat.
● adequately understand dependencies within broader complex projects
● know and comply with the license obligations
● monitor components for vulnerabilities
● end-of-life (EOL)
● make code easier to review
● a blacklist of banned components
● provide an SBOM to a customer

An SBOM can help an organization choose their software in the following ways:

● identify potentially vulnerable components
● a more targeted security analysis
● verify the sourcing
● compliance with policies
● aware of end-of-life components
● verify some claims
● understand the software’s integration
● pre-purchase and pre-installation planning
● market signal

An SBOM can help an organization configure, maintain, and administer its software in the following ways:

- Organization can quickly evaluate whether it is using the component
- Drive independent mitigations
- Make more informed risk-based decisions
- Alerts about potential end-of-life
- Better support compliance and reporting requirements
- Reduce costs through a more streamlined and efficient administration

SAFECode.org

A report: Managing Security Risks Inherent in the Use of Third-party Components

Summary: this is an excellent paper with many useful ideas on process and effective approaches. It is attempts to combine both systems and component lifecycles.

One of the challenges associated with using third-party components is their discoverability, along with establishing and maintaining the product bill of materials (BOM). There are automated solutions and tools available that identify included third-party components and generate a BOM; however, there is not a one size-fits-all solution that can be used for every different scenario. A company that uses several different programming languages and frameworks would require a tool that understands all of them in order to be able to find all included TPCs. Without an accurate BOM, including third-party component names and exact versions, it is very difficult to correctly and consistently identify new or existing vulnerabilities in the TPCs used, or identify all relevant patches. In the absence of a BOM, when new security vulnerabilities are published, organizations must scramble to identify which of their products, if any, are affected. This can be a painstaking process for organizations that do not know what TPCs they are using.

In order to create a product BOM covering all utilized third-party components, there must be a way to uniquely identify each TPC. Unfortunately, a single TPC is sometimes known by multiple names, and it can be difficult to find the correct or most commonly used name

Dependencies – Identifying the TPCs used in a product or by an organization overall is further complicated by the hierarchical nature of TPCs. A single TPC may use several TPC sub-components, each of those further referencing additional TPC sub-components, and so on

When a new security vulnerability is reported for a TPC, teams are faced with the challenge of determining 1) whether that TPC is included in their product and 2) whether the product is affected by the specific vulnerability. It is not uncommon for a product to utilize a component and not be affected by a particular CVE. Often, products will utilize a subset of the functionality contained in a TPC. Answering these questions is made more difficult by the naming challenge, dependency challenge and vulnerability documentation.

One can save a lot of grief later on by taking security into consideration during the selection process. Some third party components may not have been designed or implemented with security in mind, resulting in security risks that could affect products or services that use them.

“**Overview of the Third-party Component Management Life Cycle**

The high-level steps in TPC management are depicted in Figure 2 and described in detail below. While Maintain, Assess, and Mitigate depend on each other, Monitor can be seen as an independent step that is required throughout the entire third-party component life cycle.

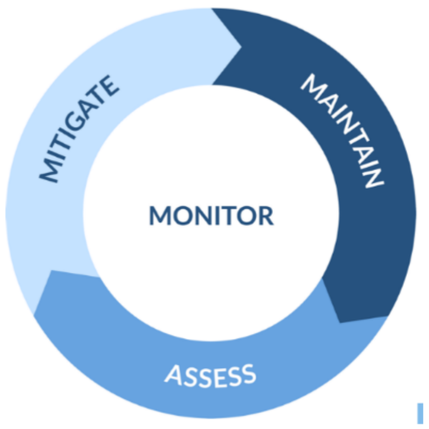


Figure 2: The high-level steps of TPC management are Maintain, Assess, and Mitigate. The Monitor step is a central aspect of TPC management and thus valid throughout the entire life cycle.

I) Maintain a List of TPCs

Having a list of TPCs in use or to be used is the first step in managing them. Intuitively, this is similar to having a bill of materials, with the key difference that it should include TPCs slated for future use as well as those in current use.

II) Assess Security Risks from TPCs

Once a list of TPCs is available, the TPCs must be assessed to gauge risks in their use. A good and easy starting point is determining known security vulnerabilities of a TPC and their impact on the TPC's intended use. This provides insight into potential issues with integrated TPCs. The risk assessment should consider aspects that could hint at unknown security issues or impending problems in using a TPC. These aspects should include assessing the maturity of the TPC provider, such as maintenance cadence, stability of the TPC over time, development practices employed by the TPC provider, whether the TPC will reach end of life within the expected lifetime of a product, etc. The outcome of this step can be a risk score for a TPC of interest.

III) Mitigate or Accept Risks Arising Due to Vulnerable TPCs

With access to the risk profile for a TPC, an organization must decide whether its use is acceptable or whether it needs mitigations. Mitigations can be done in a number of ways. The most straightforward is to look for a newer patched version of the same TPC or an alternative TPC that has an acceptable risk score. However, upgrading or changing TPCs may be difficult at times: the TPC in question may be providing a unique functionality or it could have been incorporated in the legacy code already. In such circumstances, mitigations should aim to bring down the impact of risks. For example, vulnerabilities in TPCs could be mitigated by strict input validation/output sanitization by the embedding product or by reducing privileges/access of code involving the TPC. This also includes hardening TPCs, such as by disabling unused services, changing the configuration of a TPC or removing unused parts of it. In the event that a patched version is not available, the organization using the component can submit a patch to the managing entity or can remediate the vulnerability internally itself. There are pros and cons to each approach.

IV) Monitor for Changes

Once a TPC is incorporated in a product, it needs continuous monitoring of different information resources to ensure that its risk profile remains acceptable over time. Discovery of new vulnerabilities in a TPC or the TPC reaching end of life are scenarios that may tip the TPC's risk profile to unacceptable. This step should leverage public resources such as the TPC provider's website and vulnerability databases, as well as company-level policies, such as defining when a TPC is considered to be at its end of life.

TPC Life Cycle and Software Development Life Cycle

As shown in Figure 3 and Figure 4, the TPC life cycle and software development life cycle (SDLC) go hand in hand. Both figures show the four typical phases of the SDLC -- Requirements, Design, Develop and Support -- and that the TPC life cycle relates to them and is valid during all phases of the SDLC.

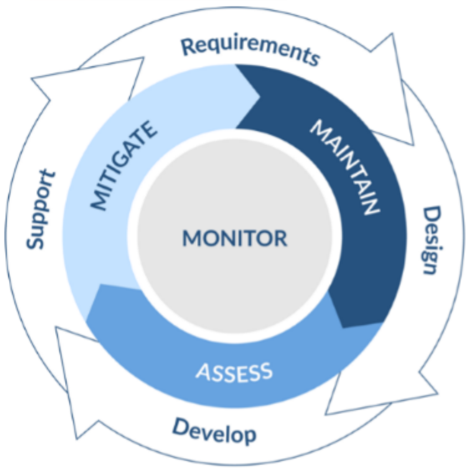


Figure 3: The TPC life cycle and the Software Development Life Cycle go hand in hand. This figure shows the repetitive nature of the both life cycles. Figure 4 shows how individual stages in these life cycles correlate.

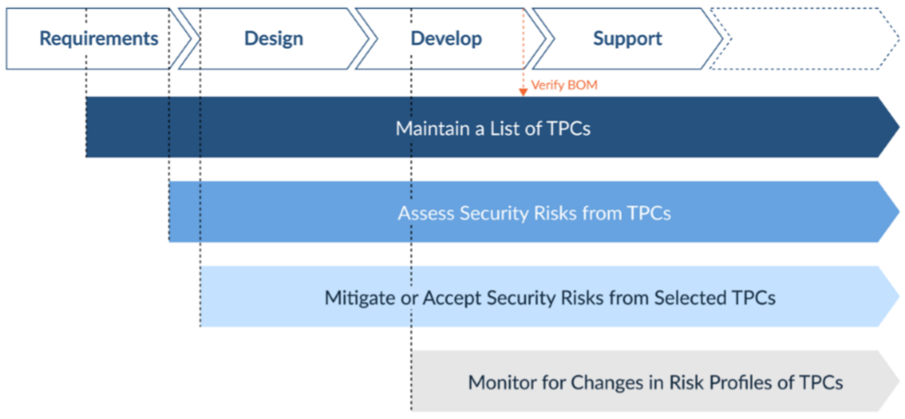

Requirements
Design
Develop
Support
Verify BOM
Maintain a List of TPCs
Assess Security Risks from TPCs
Mitigate or Accept Security Risks from Selected TPCs
Monitor for Changes in Risk Profiles of TPCs

Figure 4: TPC life cycle steps should start early and occur at distinct points of the SDLC.

One of the most effective ways to kick off a TPC management program is to piggyback it on an already established process, e.g., SDLC or legal review. The TPCs should be tracked starting as early as the SDLC requirements phase, when functional requirements may dictate the use of specific TPCs (step Maintain). The bulk of TPC selection usually happens in the design and develop phases and populates the list of TPCs. For legacy applications without any TPC life cycle management, TPC enumeration often starts in the support phase of the SDLC.

The risk assessment process, step Assess, starts as soon as a candidate TPC is identified and continues until no new TPCs are needed. If a TPC is determined to have high risk, the mitigation step includes exploring alternatives, such as using a newer version of the TPC, using a different TPC with lower risk, or choosing to accept the risk. In turn, this could cause the list of TPCs to change and risk assessment to commence for the newly selected TPCs. Risk mitigation, step Mitigate, continues in the design and develop SDLC phases, as in some cases design or code-level safeguards may be necessary to mitigate risks of a TPC that must be used.

After risk mitigation, TPCs used (or the BOM) should be monitored for changes in risk profiles, step Monitor, in response to newly discovered vulnerabilities or being marked EOL by the TPC provider. This step also triggers risk assessment if new or updated TPCs are added to the BOM. This monitoring could kick off risk assessment and a hunt for a new TPC version/alternative.

An important step of the TPC life cycle is to verify and confirm the bill of materials before a product is shipped to customers and enters the support phase ("Verify BOM" in Figure 4). This ensures that no TPCs were The overall TPC life cycle management steps, including the four high-level steps Maintain, Assess, Mitigate and Monitor (red boxes), are depicted in Figure 5. These four main TPC life cycle management steps have already been discussed in section 3.1. The key ingredients (green boxes) of each main TPC life cycle step are also shown in Figure 5 and further discussed in this section. Green boxes with stars are considered the bare minimum for a meaningful TPC life cycle management and thus at least these steps should be covered by a quick starter TPC management process. missed during the develop phase and that the BOM reflects reality.

Centralized "Approved" Components Store

One emerging best practice that addresses many of the challenges presented in this document is the use of a centralized, curated set of "approved" third-party components. This could take the form of a repository of components or a list of approved components and versions. Workflow can be as simple or complex as the organization requires, and policy and automation should be used to ensure that only thirdparty components from the approved list are used.

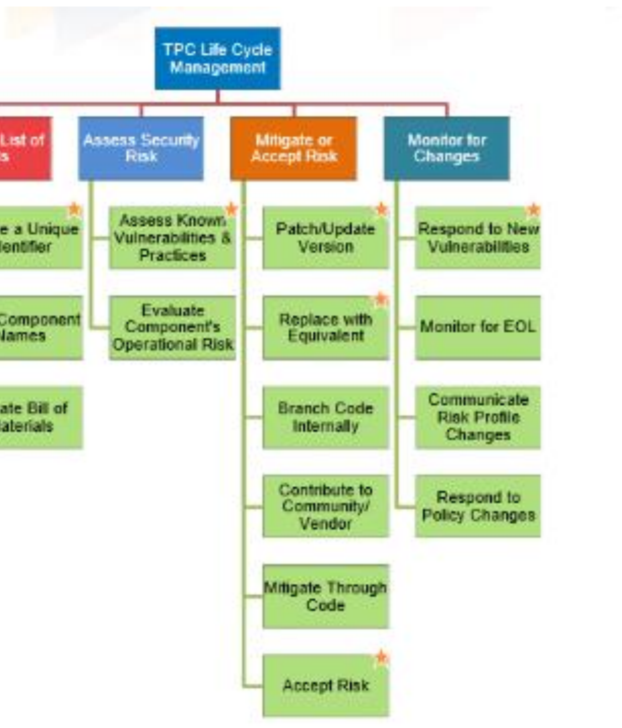

TPC Life Cycle Management
Maintain List of TPCs
Assess Security Risk
Mitigate or Accept Risk
Monitor for Changes
Define a Unique Identifier
Map Component Names
Create Bill of Materials
Assess Known Vulnerabilities & Practices
Evaluate Component's Operational Risk
Patch/Update Version
Replace with Equivalent
Branch Code Internally
Contribute to Community/ Vendor
Mitigate Through Code
Accept Risk
Respond to New Vulnerabilities
Monitor for EOL
Communicate Risk Profile Changes
Respond to Policy Changes

Figure 5: This chart depicts all TPC life cycle management steps of the TPC management process. Green boxes with stars are considered the bare minimum.

Practices Used by the Community/Supplier in Handling Vulnerabilities

Where possible, the development practices of the provider should be examined to determine whether the provider practices aspects of secure development. If the provider has a well-established and published set of secure development practices, it is more likely to produce components that satisfy security objectives. The development practices of a community or supplier are a key indicator of the security risk associated with its software components. Proactive efforts of the suppliers should be recognized and used in assessing risk. An example of such an effort is the Linux Foundation Core Infrastructure Initiative (CII), which provides "Best Practice Badges" for Free/Libre and Open Source Software (FLOSS) projects to show that they follow best practices. Projects that voluntarily self-certify receive a badge, which allows consumers to quickly assess which FLOSS projects are following the best practices.

The following should be considered:

- Does the community/supplier provide clear vulnerability/patch reporting methods, to include reporting to commonly used repositories (e.g., CVE ID in the National Vulnerability Database), and provide frequent feedback on submitted vulnerabilities?
- Is there a dedicated website for security issues?
- Is there a way to (privately) submit security patches?
- Does the supplier's process incorporate security best practices?
- Does the supplier perform automated security testing (e.g., static analysis, dynamic analysis, vulnerability scanning) of the components, both periodically and on an ongoing basis (since tooling quality usually improves over time)?
- Do the supplier's automated standards-based assessment tools utilize public vulnerability and security flaw repositories (Common Weakness Enumeration, CVE, Common Attack Pattern Enumeration and Classification, etc.)?
- Does the community/supplier routinely disclose vulnerabilities and prepare customers for patch deployment?
- Does the community/supplier have a history and reputation for actively patching reported vulnerabilities?
- Does the community/supplier have a way for researchers or customers to responsibly submit a security vulnerability to it?

- Does the community/supplier issue security advisories or alerts as a way to notify customers of remediation of security vulnerabilities?

When evaluating operational risk, consider the following:

- Does the component have a regular maintenance and update cycle?
- Does the component have a clearly defined and consistent set of maintainers?
- What controls does the supplier have to protect against unapproved changes/updates?
- What is the expected lifetime of the component?
- What criteria or process will be used to determine when to update the component?
- How does the TPC maintainer manage security response?
- How much documentation is available on the component, and what is the quality of that documentation?
- What kind of community surrounds the component (this can take the form of support forums (Stack Overflow, paid support desk), user blogs, IRC chat rooms, email groups or books)?
- How long has the component existed and when was the last major release?
- How widely used is this component both publicly and within your organization?
- What is the reputation of the component, author, supplier or community?

Mitigate or Accept Risk (MITIGATE)

When security vulnerabilities are discovered in TPCs used or included in the product, the team must understand the risk and choose the appropriate response.

The response to a vulnerability will vary, depending on certain factors, such as the severity of the vulnerability, availability of a patch/update, ease of patching/updating and the context-specific risk. Just because a CVE is rated "high severity" in the National Vulnerability Database does not mean it is high severity for its usage in a given product. The team may choose to mitigate the risk, via a variety of mechanisms, or accept the risk and not mitigate.

MITIGATE1: Patch/Update the Version.

MITIGATE2: Replace with an Equivalent.

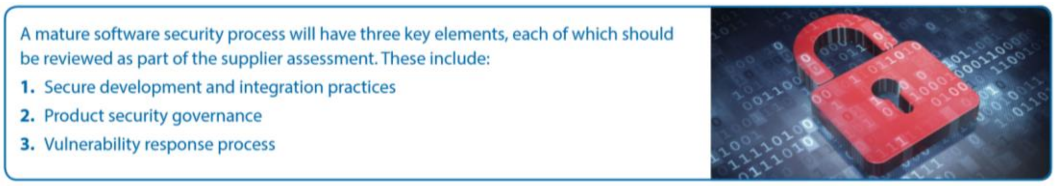
MITIGATE3: Branch Code Internally.

MITIGATE4: Contribute to Community/Vendor.

MITIGATE5: Mitigate Through Code.

MITIGATE6: Accept Risk.

Safecode have published another document entitled Principles for Software Assurance Assessment - A Framework for Examining the Secure Development Processes of Commercial Technology Provider. It describes a risk-based software assurance approach. The high level approach is illustrated below and each area is developed more fully in the text of the document.


A mature software security process will have three key elements, each of which should be reviewed as part of the supplier assessment. These include:

1. Secure development and integration practices
2. Product security governance
3. Vulnerability response process

Safecode published another document entitled Fundamental Practices for Secure Software Development. It provides an overview of some of the key activities within software development and some of the lifecycle priorities. These include:

- Design (including Threat Modelling (see below)
- Secure Coding Practices
- Managing the security risk inherent in the use of Third-Party components
- Testing and validation
- Manage security findings
- Vulnerability responses and disclosure
- Planning the implementation and deployment of secure development practices.

Safecode published an approach to Tactical Threat Modelling to describe an overview approach to threat modelling within a development context.

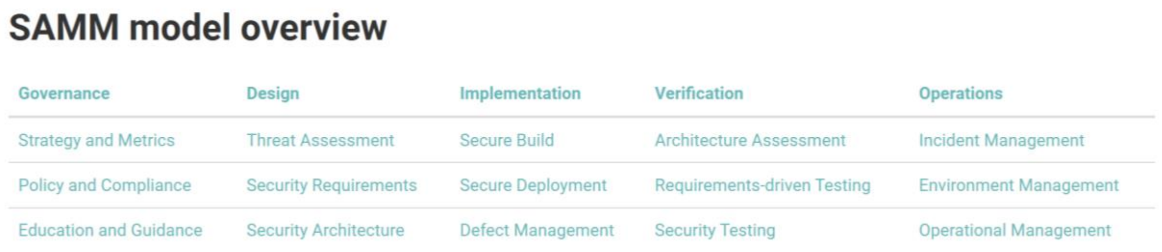
OWASP

The Open Web Application Security Project (OWASP)[4] is a nonprofit foundation that works to improve the security of software. Through community-led open source software projects, hundreds of local chapters worldwide, tens of thousands of members, and leading educational and training conferences, the OWASP Foundation is the source for developers and technologists to secure the web.

OWASP have 66 Cheetsheets available covering a range of topics including Authentication, Access Control, Crypto Storage, Docker Security, MFA, Threat Modelling and Vulnerability Disclosure.

OWASP also publish their Top 10 vulnerabilities for Mobile and Web Applications.

The mission of OWASP Software Assurance Maturity Model (SAMM) is to be the prime maturity model for software assurance that provides an effective and measurable way for all types of organizations to analyze and improve their software security posture. OWASP SAMM supports the complete software lifecycle, including development and acquisition, and is technology and process agnostic. It is intentionally built to be evolutive and risk-driven in nature. SAMM is a prescriptive model, an open framework which is simple to use, fully defined, and measurable. The solution details are easy enough to follow even for non-security personnel. It helps organizations analyze their current software security practices, build a security program in defined iterations, show progressive improvements in secure practices, and define and measure security-related activities.



SAMM model overview

Governance	Design	Implementation	Verification	Operations
Strategy and Metrics	Threat Assessment	Secure Build	Architecture Assessment	Incident Management
Policy and Compliance	Security Requirements	Secure Deployment	Requirements-driven Testing	Environment Management
Education and Guidance	Security Architecture	Defect Management	Security Testing	Operational Management

Dependency-Check is a Software Composition Analysis (SCA) tool that attempts to detect publicly disclosed vulnerabilities contained within a project's dependencies. It does this by determining if there is a Common Platform Enumeration (CPE) identifier for a given dependency. If found, it will generate a report linking to the associated CVE entries.

[4] https://owasp.org

OWASP Devsecops Maturity Model

From a startup to a multinational corporation the software development industry is currently dominated by agile frameworks and product teams and as part of it DevOps strategies. It has been observed that during the implementation, security aspects are usually neglected or are at least not sufficient taken account of. It is often the case that standard safety requirements of the production environment are not utilized or applied to the build pipeline in the continuous integration environment with containerization or concrete docker. Therefore, the docker registry is often not secured which might result in the theft of the entire company's source code.

The DevSecOps Maturity Model shows security measures which are applied when using DevOps strategies and how these can be prioritized. With the help of DevOps strategies security can also be enhanced. For example, each component such as application libraries and operating system libraries in docker images can be tested for known vulnerabilities. Attackers are intelligent and creative, equipped with new technologies and purpose. Under the guidance of the forward-looking DevSecOps Maturity Model, appropriate principles and measures are at hand implemented which counteract the attacks.

The model includes a number of dimensions including Build, Deployment, Patch Management:, Education and Guidance, Culture and Org, Process, Monitoring, Logging, Infrastructure Hardening, Dynamic depth for applications, Static depth for applications, Test-Intensity, Consolidation, Application tests, Dynamic depth for infrastructure and Static depth for infrastructure.

OASIS Open Standards, Open Source (OASIS)

OASIS Open offers projects—including open source projects—a path to standardization and de jure approval for reference in international policy and procurement.

OASIS has a broad technical agenda encompassing cybersecurity, blockchain, privacy, cryptography, cloud computing, IoT, urban mobility, emergency management, content technologies. In fact, any initiative for developing code, APIs, specifications, or reference implementations can find a home at OASIS.

The OASIS community is committed to advancing work that lowers cost, improves efficiency, stimulates innovation, grows global markets, and promotes interoperability. Each project operates independently under industry-leading process and clear IPR policies.

Some of the most widely adopted OASIS Standards include AMQP, CAP, CMIS, DITA, DocBook, KMIP, MQTT, OpenC2, OpenDocument, PKCS, SAML, STIX, TAXII, TOSCA, UBL, and XLIFF. Many of these have gone on to be published as ISO, IEC, or ITU standards. New work is encouraged, and all are welcome to participate.”

Summary: there is potential to think of aligning software packages to this sort of validation approach

The Linux Foundation (LF)

CII Best Practices Badge Program (CII)

The Core Infrastructure Initiative (CII) Best Practices badge is a way for Free/Libre and Open Source Software (FLOSS) projects to show that they follow best practices. Projects can voluntarily self-certify, at no cost, by using this web application to explain how they follow each best practice. The CII Best Practices Badge is inspired by the many badges available to projects on GitHub. Consumers of the badge can quickly assess which FLOSS projects are following best practices and as a result are more likely to produce higher-quality secure software.

More information on the CII Best Practices Badging program, including background and criteria, is available on GitHub.

Summary: Although there may be limited value in a 'badge' based scheme, the controls are a useful (but not complete) set.

The LF Openchain initiative maintains the industry-standard for the key requirements of a quality open source compliance program. Companies, governments and non-profit organizations use our standard for open source compliance every day to build trust in the supply chain.

The OpenChain Project establishes trust in the open source from which software solutions are built. It accomplishes this by making open source license compliance simpler and more consistent. The OpenChain Specification defines inflection points in business workflows where a compliance process, policy or training should exist to minimize the potential for errors and maximize the efficiency of bringing solutions to market.

The Linux Foundation published a document in February 2020 entitled Improving Trust and Security in Open Source Projects. It describes:

The Eight Best Practices describes a set of "activities" that teams producing secure software should do. This section balances guidance that is meaningful and relatively easy to implement without being overly prescriptive or rigid. They are:

1. Roles and Responsibilities
2. Security Policy
3. Know Your Contributors
4. The Software Supply Chain
5. Technical Security Guidance
6. Security Playbooks
7. Security Testing
8. Secure Releases and Updates

The final section of the document describes a Certification Scheme that is designed to enable open source projects to self-certify, and for commercial open-source companies to provide higher levels of independent third party certification through a network of Linux Foundation certified security consultants.

Throughout the document we allow for levels of security maturity and ease of getting started by describing varying depth of specific practices as Basic, Standard and Advanced.

Basic practices are considered things that everyone should do, regardless of their project type and maturity. They are generally easy to implement and have a low overhead to the team while providing a basic level of assurance. Knowing that a team applies all of the Basic practices allows consumers to quickly appreciate that all the basics have been thought about and are being implemented.

Standard practices provide a higher level of assurance but usually require a higher degree of overhead therefore are suited to more mature projects and teams. Standard practices require some thought and come with some over-head but are appropriate to software teams producing applications that run in production. Knowing that a team applies all of the Standard practices allows consumers to quickly appreciate that security is important to the project.

Advanced practices go further than Standard and are designed for teams producing mission critical software or for teams wishing to use and or demonstrate security as a differentiator. Advanced practices usually require careful implementation and come with a cost. Knowing that a team applies all of the Advanced practices allows consumers to quickly appreciate that security is of utmost importance to the project.

This is focused on open source software and as such has a set of valuable considerations in managing the trust in open source.

ETSI TR 103 306 V1.3.1 Technical Report CYBER;

A report: Global Cyber Security Ecosystem

Summary: A comprehensive list of global and national cyber security organisations with a brief summary of their activity.

National Institute of Standards and Technology (NIST)

The Zero Trust Architecture project is also underway (due 12m from April 2020). The proliferation of cloud computing, mobile device use, and the Internet of Things has dissolved traditional network boundaries. Hardened network perimeters alone are no longer effective for providing enterprise security in a world of increasingly sophisticated threats. Zero trust is a design approach to architecting an information technology environment that could reduce an organization's risk exposure in a "perimeter-less" world.

The NIST National Cybersecurity Center of Excellence (NCCoE) project started Oct 2019 and aims to build a reference model: Security Practice Guide for VMware Hybrid Cloud Infrastructure as a Service (IaaS) Environments with hardware roots of trust (Draft).

A zero trust architecture treats all users as potential threats and prevents access to data and resources until the users can be properly authenticated and their access authorized. In essence, a zero trust architecture allows a user full access but only to the bare minimum they need to perform their job. If a device is compromised, zero trust can ensure that the damage is contained.

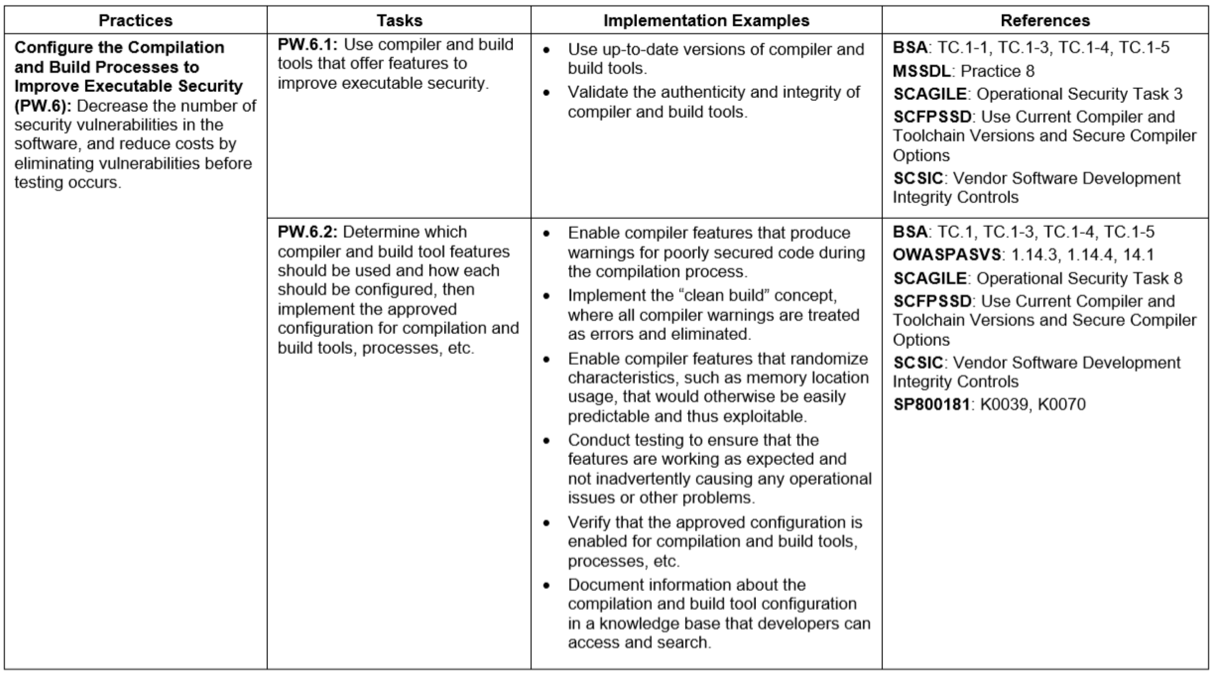
The concept of zero trust has been around for more than a decade, but technology to support it is now moving into the mainstream. A zero trust architecture leans heavily on components and capabilities for identity management, asset management, application authentication, network segmentation, and threat intelligence. Architecting for zero trust should enhance cybersecurity without sacrificing the user experience. The NCCoE is researching ongoing industry developments in zero trust and its component technologies that support the goals and objectives of a practical, secure, and standards-based zero trust architecture.

NIST Cybersecurity White Paper: Mitigating The Risk Of Software Vulnerabilities By Adopting An SSDF: April 23, 2020

Few software development life cycle (SDLC) models explicitly address software security in detail, so secure software development practices usually need to be added to each SDLC model to ensure the software being developed is well secured. This white paper recommends a core set of high level secure software development practices called a secure software development framework (SSDF) to be integrated within each SDLC implementation. The paper facilitates communications about secure software development practices among business owners, software developers, project managers and leads, and cybersecurity professionals within an organization. Following these practices should help software producers reduce the number of vulnerabilities in released software, mitigate the potential impact of the exploitation of undetected or unaddressed vulnerabilities, and address the root causes of vulnerabilities to prevent future recurrences. Also, because the framework provides a common vocabulary for secure software development, software consumers can use it to foster communications with suppliers in acquisition processes and other management activities.

The SSDF contains structured reference to a number of the other sources cited in this research paper. It contains some excellent content including these excerpts:

Practices	Tasks	Implementation Examples	References
Review the Software Design to Verify Compliance with Security Requirements and Risk Information (PW.2): Help ensure that the software will meet the security requirements and satisfactorily address the identified risk information.	**PW.2.1:** Have a qualified person who was not involved with the software design review it to confirm that it meets all of the security requirements and satisfactorily addresses the identified risk information.	• Review the software design to confirm that it addresses all of the security requirements. • Review the risk models created during software design to determine if they appear to adequately identify the risks. • Review the software design to confirm that it satisfactorily addresses the risks identified by the risk models. • Have the software's designer correct failures to meet the requirements. • Change the design and/or the risk response strategy if the security requirements cannot be met.	**BSA**: TV.3, TV.3-1, TV.5 **BSIMM10**: AA1.2, AA2.1 **ISO27034**: 7.3.3 **OWASPTEST**: Phase 2.2 **SAMM15**: DR1-A, DR1-B **SP800181**: T0328; K0038, K0039, K0070, K0080, K0119, K0152, K0153, K0161, K0165, K0172, K0297; S0006, S0009, S0022, S0036, S0141, S0171
Verify Third-Party Software Complies with Security Requirements (PW.3): Reduce the risk associated with using acquired software modules and services, which are potential sources of additional vulnerabilities.	**PW.3.1:** Communicate requirements to third parties who may provide software modules and services to the organization for reuse by the organization's own software.	• Define a core set of security requirements, and include them in acquisition documents, software contracts, and other agreements with third parties. • Define the security-related criteria for selecting commercial and open-source software. • Require the providers of commercial software modules and services to provide evidence that their software complies with the organization's security requirements. • Establish and follow procedures to address risk when there are security requirements that third-party software modules and services do not meet.	**BSA**: SM.1, SM.2, SM.2-1, SM.2.4 **BSIMM10**: CP2.4, SR2.5, SR3.2 **IDASOAR**: Fact Sheets 19, 21 **MSSDL**: Practice 7 **SAMM15**: SR3-A **SCFPSSD**: Manage Security Risk Inherent in the Use of Third-Party Components **SCSIC**: Vendor Sourcing Integrity Controls **SP80053:** SA-4, SA-12 **SP800160**: 3.1.1, 3.1.2 **SP800181**: T0203, T0415; K0039; S0374; A0056, A0161
	PW.3.2: Use appropriate means to verify that commercial, open source, and all other third-party software modules and services comply with the requirements.	• See if there are publicly known vulnerabilities in the software modules and services that the vendor has not yet fixed. • Ensure each software module or service is still actively maintained, which should include new vulnerabilities found in the software being remediated. • Determine a plan of action for each third-party software module or service that is no longer being maintained or available in the future. • Use the results of commercial services for vetting the software modules and services. • [See **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**] • [See **Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8)**]	**BSA**: SC.3-1, TV.2 **IDASOAR**: Fact Sheet 21 **MSSDL**: Practice 7 **OWASPASVS**: 10, 14.2 **PCISSLRAP**: 4.1 **SCAGILE**: Tasks Requiring the Help of Security Experts 8 **SCFPSSD**: Manage Security Risk Inherent in the Use of Third-Party Components **SCSIC**: Vendor Sourcing Integrity Controls **SCTPC:** 3.2.2 **SP80053**: SA-12 **SP800160**: 3.1.2, 3.3.8 **SP800181**: SP-DEV-002; K0153, K0266 [See **Review and/or Analyze Human-Readable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.7)**] [See **Test Executable Code to Identify Vulnerabilities and Verify Compliance with Security Requirements (PW.8)**]



Practices	Tasks	Implementation Examples	References
Configure the Compilation and Build Processes to Improve Executable Security (PW.6): Decrease the number of security vulnerabilities in the software, and reduce costs by eliminating vulnerabilities before testing occurs.	**PW.6.1:** Use compiler and build tools that offer features to improve executable security.	• Use up-to-date versions of compiler and build tools. • Validate the authenticity and integrity of compiler and build tools.	**BSA**: TC.1-1, TC.1-3, TC.1-4, TC.1-5 **MSSDL**: Practice 8 **SCAGILE**: Operational Security Task 3 **SCFPSSD**: Use Current Compiler and Toolchain Versions and Secure Compiler Options **SCSIC**: Vendor Software Development Integrity Controls
	PW.6.2: Determine which compiler and build tool features should be used and how each should be configured, then implement the approved configuration for compilation and build tools, processes, etc.	• Enable compiler features that produce warnings for poorly secured code during the compilation process. • Implement the "clean build" concept, where all compiler warnings are treated as errors and eliminated. • Enable compiler features that randomize characteristics, such as memory location usage, that would otherwise be easily predictable and thus exploitable. • Conduct testing to ensure that the features are working as expected and not inadvertently causing any operational issues or other problems. • Verify that the approved configuration is enabled for compilation and build tools, processes, etc. • Document information about the compilation and build tool configuration in a knowledge base that developers can access and search.	**BSA**: TC.1, TC.1-3, TC.1-4, TC.1-5 **OWASPASVS**: 1.14.3, 1.14.4, 14.1 **SCAGILE**: Operational Security Task 8 **SCFPSSD**: Use Current Compiler and Toolchain Versions and Secure Compiler Options **SCSIC**: Vendor Software Development Integrity Controls **SP800181**: K0039, K0070

Includes an excellent set of references:

[1] Newhouse W, Keith S, Scribner B, Witte G (2017) National Initiative for Cybersecurity Education (NICE) Cybersecurity Workforce Framework. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-181. https://doi.org/10.6028/NIST.SP.800-181

[2] National Institute of Standards and Technology (2018), Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1. (National Institute of Standards and Technology, Gaithersburg, MD). https://doi.org/10.6028/NIST.CSWP.04162018

[3] Migues S, Steven J, Ware M (2019) Building Security in Maturity Model (BSIMM) Version 10. Available at https://www.bsimm.com/download/

[4] BSA (2019) Framework for Secure Software. Available at https://www.bsa.org/reports/bsa-framework-for-secure-software

[5] Hong Fong EK, Wheeler D, Henninger A (2016) State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation 2016. (Institute for Defense Analyses [IDA], Alexandria, VA), IDA Paper P-8005. Available at https://www.ida.org/research-and-publications/publications/all/s/st/stateoftheartresources-soar-for-software-vulnerability-detection-test-and-evaluation-2016

[6] International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), Information technology – Security techniques – Application security – Part 1: Overview and concepts, ISO/IEC 27034-1:2011, 2011. Available at https://www.iso.org/standard/44378.html

[7] Microsoft (2019) Security Development Lifecycle. Available at https://www.microsoft.com/en-us/sdl

[8] Open Web Application Security Project (2019) OWASP Application Security Verification Standard 4.0. Available at https://github.com/OWASP/ASVS

[9] Open Web Application Security Project (2014) OWASP Testing Guide 4.0. Available at https://www.owasp.org/images/1/19/OTGv4.pdf

[10] Payment Card Industry (PCI) Security Standards Council (2019) Secure Software Lifecycle (Secure SLC) Requirements and Assessment Procedures Version 1.0. Available at https://www.pcisecuritystandards.org/document_library?category=sware_sec#results

[11] Open Web Application Security Project (2017) Software Assurance Maturity Model Version 1.5. Available at https://www.owasp.org/index.php/OWASP_SAMM_Project

[12] Software Assurance Forum for Excellence in Code (2012) Practical Security Stories and Security Tasks for Agile Development Environments. Available at http://www.safecode.org/publication/SAFECode_Agile_Dev_Security0712.pdf

[13] Software Assurance Forum for Excellence in Code (2018) Fundamental Practices for Secure Software Development: Essential Elements of a Secure Development Lifecycle Program, Third Edition. Available at https://safecode.org/wpcontent/uploads/2018/03/SAFECode_Fundamental_Practices_for_Secure_Software_Dev elopment_March_2018.pdf

[14] Software Assurance Forum for Excellence in Code (2010) Software Integrity Controls: An Assurance-Based Approach to Minimizing Risks in the Software Supply Chain. Available at http://www.safecode.org/publication/SAFECode_Software_Integrity_Controls0610.pdf

[15] Software Assurance Forum for Excellence in Code (2017) Managing Security Risks Inherent in the Use of Third-Party Components. Available at https://www.safecode.org/wpcontent/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf

[16] Software Assurance Forum for Excellence in Code (2017) Tactical Threat Modeling. Available at https://www.safecode.org/wpcontent/uploads/2017/05/SAFECode_TM_Whitepaper.pdf

[17] Joint Task Force Transformation Initiative (2013) Security and Privacy Controls for Federal Information Systems and Organizations. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-53, Revision 4, Includes updates as of January 22, 2015. https://doi.org/10.6028/NIST.SP.800-53r4 [18] Ross R, McEvilley M, Oren J (2016) Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-160, Volume 1, Includes updates as of March 21, 2018. https://doi.org/10.6028/NIST.SP.800-160v1

The European Union Agency for Cybersecurity (ENISA)

ENISA have published 2 reports in this area: Advancing Software Security in the EU and Good Practices for Security of IOT- Secure Software Development Lifecycle.

These are relevant documents pointing to a number of relevant security best practices and a summary of various approaches and standards.

Advancing Software Security in the EU

This study discusses some key elements of software security and provides a concise overview of the most relevant existing approaches and standards while identifying shortcomings associated with the secure software development landscape, related to different inherent aspects of the process. Lastly, it provides a number of practical considerations relevant to the different aspects of software development within the newly established EU cybersecurity certification framework and the EU cybersecurity certification schemes.

Section 2 discusses some key elements of software security in order to allow a better comprehension of the document's direction, being completed with an overview of the most relevant existing approaches and standards. Section 3 provides an overview of the shortcomings inside the software security landscape, related with different inherent aspects of the process, of the products and of the concepts surrounding the software security concept itself. Section 4 provides a number of practical considerations that can be considered and adopted with regards to the different aspects of software development within the newly established EU cybersecurity certification framework and the EU cybersecurity certification schemes.

There are several standards and good practices focusing on software security. Most notably:

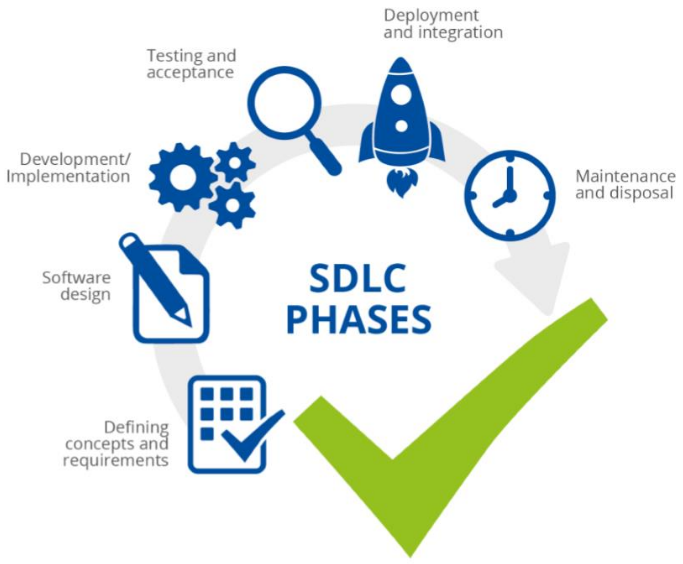
- Common Criteria is a consolidated and widely recognized framework for product (often times meaning software) security evaluations. The evaluation process, in essence, pertains to the assessment against pre-defined security functional and assurance requirements
- The OWASP ASVS (Application Security Verification Standard) is a community developed verification framework focusing on technical security controls verifiable in the software product and in the development process. It distinguishes maturity levels that reflect a system's security profile; this suggests which security requirements must or may apply.
- BSIMM is a commercial initiative, based on a bottom-up approach that starts with listing activities of large successful software companies. This benchmark shows which controls can be considered in maturity terms, in which higher maturity levels are typically a step-up from lower maturity levels.
- BSI PAS 754 "Software Trustworthiness. Governance and management. Specification" is a standard developed by the British Standards Institution encompassing both security and reliability concerns (safety, reliability, availability, resilience and security).
- ISA 99 / IEC 62443 provides a set of standards aimed at industrial control systems and provides a flexible framework to address and mitigate current and future security vulnerabilities in industrial automation and control systems
- ISO/IEC 27034 is a multipart, guidance international standard focusing on application security. Each of its numerous parts goes down in deep details on how software security should be achieved
- ISO/IEC 62304 is a certifiable standard in the field of medical (device) software focusing on life cycle requirements for the development of medical software and software within medical device
- PCI SSC has been relying for the last decade on the PA-DSS standard for payment applications certification

- OWASP Software Assurance Maturity Model (SAMM) is a community developed framework to help organizations formulate and implement a strategy for software security.
- The Microsoft SDL is considered a classical model in secure development frameworks. It distinguishes Training (a prerequisite), Requirements, Design, Implementation, Verification, Release, and Response (to external or unexpected events).
- The Dutch SSA (Secure Software Alliance) has defined a framework for secure software development intending to conform to all phases of the SDLC. It focuses on threat modeling as a prerequisite for secure software development.
- Safecode.org is an initiative to identify and promote best practices for secure software development. Their "Fundamental Practices for Secure Software Development" follows steps in the development process from governance to design, coding, testing, and vulnerability response

When analysing available SDLC (Software Development Lifecycle) standards it becomes clear that there is consensus on what activities should be included. This consensus centers on the key activities that can be distinguished in the development lifecycle like: training, requirements, coding guidelines, design, design review, threat modelling, secure verification (automated testing, static analysis tools, test tools, manual code review and penetration testing), dependency management, incident management, vulnerability management and environment hardening.

<u>Good Practices for Security of IOT- Secure Software Development Lifecycle</u>

…the aim of this study is to define a set of good practices and guidelines to be applied in the different phases of the secure SDLC of IoT solutions.


Deployment and integration
Testing and acceptance
Development/ Implementation
Maintenance and disposal
Software design
SDLC PHASES
Defining concepts and requirements

To organise the domains in a logical manner, they were classified into three main groups:

- People: security considerations that affect all stakeholders involved in the life cycle of IoT solutions, from the software developers, to the end users of the product.
- Processes: secure development addresses security in the process of software development when a software project is conceived, initiated, developed, and brought to market.

Technologies: technical measures and elements used in order to reduce vulnerabilities and flaws during the software development process

The document contains a comprehensive set of best practices (referenced in an Appendix). Its focus is IOT but there is a broader read across to software development more generally. It is written as if there is full influence over the development lifecycle (which may not be the case for open source software where some of the early stages (including Coding) may be delivered by others).

UK National Cyber Security Centre (NCSC)

The NCSC has a range of advice and best practice including:

- A *systems* risk management approach*:* System-driven risk analyses are best suited to identifying risks which emerge from the *interaction* of all a system's components. These risks can occur without any *individual* component breaking or being compromised, so they identify risks that component-driven approaches cannot. Identifies several commonly used system-driven cyber risk management methods and frameworks: STAMP, TOGAF and SABSA.
- A *component* driven risk management approach: component-driven risk assessments are focused on system components. Typical examples include: hardware (computers, servers, etc.), software, data sets, services, personal information, business critical information, staff." Identifies a brief description of commonly used component-driven cyber risk management methods and frameworks: ISO/IEC 27005:2011, Information Security Forum (ISF) IRAM 2, HMG Information Assurance Standard 1 & 2, US National Institute of Standards and Technology (NIST) SP 800-30, Octave Allegro, and ISACA COBIT 5 for Risk
- Risk Management
- Secure development and deployment guidance: 8 principles are listed; Secure development is everyone's concern, Keep your security knowledge sharp, Produce clean & maintainable code, Secure your development environment, Protect your code repository, Secure the build and deployment pipeline, Continually test your security & Plan for security flaws. Each principle area include more detailed actions.
- Guides for the design of cyber secure systems: These guides are not specifically written for open source software solutions but still contain useful approaches to system design.

The Cyber Security Principles offer the most generally applicable advice.

The Virtualisation Design Principles apply to the more specific case of systems which rely on virtualisation technologies.

We have divided each set of principles into five categories, loosely aligned with stages at which an attack can be mitigated:

- Establish the context
  Determine *all* the elements which compose your system, so your defensive measures will have no blind spots.
- Making compromise difficult
  An attacker can only target the parts of a system they can reach. Make your system as difficult to penetrate as possible
- Making disruption difficult
  Design a system that is resilient to denial of service attacks and usage spikes
- Making compromise detection easier
  Design your system so you can spot suspicious activity as it happens and take necessary action
- Reducing the impact of compromise
  If an attacker succeeds in gaining a foothold, they will then move to exploit your system. Make this as difficult as possible

Cloud security guidance

- Cyber Assessment Framework (CAF): The CAF Collection consists of a set of 14 cyber security & resilience principles, together with guidance on using and applying the principles, and the Cyber Assessment Framework (CAF) itself. The CAF cyber security principles define a set of top-level outcomes that, collectively, describes good cyber security for organisations performing essential functions. Each principle is accompanied by a narrative which provides more detail, including why the principle is important. Additionally, each principle is supported by a collection of relevant guidance which both highlights some of the relevant factors that an organisation will usually need to take into account when deciding how to achieve the outcome, and recommends some ways to tackle common cyber security challenges.

The CAF itself has 14 Principles and is written as a set of security *outcomes*. These are Governance, Risk Management, Asset Management, Supply Chain, Service Protection Policies and Processes, Identity and Access Control, Data Security, System Security, Resilient Networks & Systems, Staff awareness and training, Security Monitoring, Proactive Security Event Discovery, Response recovery & planning and Improvements.

- Secure by Default

Secure Software Alliance - Agile Secure Software Lifecycle Management

In an ideal world the Framework Secure Software would try to answer the utopian question: “Is this software system completely secure?”. Unfortunately, it is impossible to give that answer. The very definition of secure (“protected against threats”) implies that one would need to know all possible threats against the software system to answer this question, which cannot be guaranteed. However, if the question is rephrased as “Is this software system protected against all known threats?”, answering that question becomes easier. The framework will and cannot not guarantee that a software system is completely secure, but they will give the assurance that security has been sufficiently implemented.

To answer that question, one needs to know what the known threats are for a software system, and when protection against these threats is sufficiently implemented.

In a product as diverse as software, there is no set of threats that applies to all software systems and that is valid under all circumstances. Threats to a software system depend on the type of software system, how it is built, the environment in which it is used and how it is used. This context will determine the threats that are applicable to a specific software system.

Once the threats are defined, it becomes possible to verify if the software system is sufficiently protected against these threats. This protection is realized by a secure design, by implementing mitigation measures, by coding securely, security testing and by not making mistakes when doing this. The Framework Secure Software aims to provide a secure software development process that ensures correctness and completeness as much as possible, and that can be evaluated to make sure it has been applied appropriately.

To achieve this, essential security practices throughout the Software Development Life Cycle (SDLC) were observed and criteria to evaluate the results of those practices were created. By covering the full SDLC and connecting the essential security practices, it is possible to bridge the gap between the software purchaser’s abstract idea of security and the developer’s concrete understanding of a secure implementation.

By focusing on evaluating results instead of process as much as possible, it becomes possible to evaluate the security of a software system in a more objective way. This is where the real value of the Framework Secure Software is to be found compared to other frameworks: it measures the security of the result of a development process, namely the produced software itself. Other frameworks mostly focus on good processes, hoping these will result in secure software, but they don’t provide a method to verify this in a manner that can be performed in an objective and reproducible way by a developer or auditor.

The result of applying the Framework Secure Software is an intrinsically secure and traceable development process, and a method that allows software purchasers and software developers to communicate about software security in a common language both can understand.

Framework Secure Software

The evaluation process of software security is divided into four phases.

1. In the context phase, the software system is described along with its desired security properties and assumptions. This is the basis for the rest of the evaluation and will be part of the public audit report.
2. The threats phase deals with identifying possible attacks against the software system and the associated mitigating measures against these threats.
3. In the implementation phase, the code and configuration of a software system is inspected.
4. The verification phase looks at how the development organization verifies whether the implementation really is secure. This should not be confused with the assessment that an auditor performs on the implementation.

In each phase, developers create something that an auditor can assess. The exact developer actions and audit criteria are described as controls.

Summary: this approach is interesting as it attempts to focus objectively on the actual code produced (much akin to the start point in assessing open source software that has already been written).

Synopsys Cybersecurity Research Center

2019 Open Source Security and Risk Analysis https://www.synopsys.com/software-integrity.html

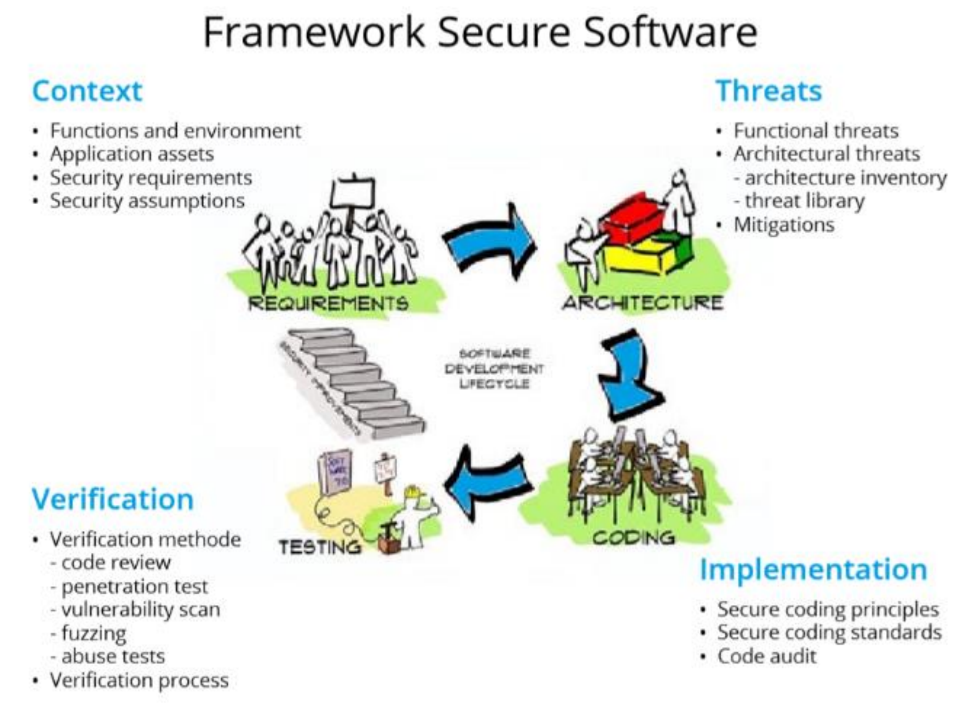


Figure 16. Framework Secure Software

For over 15 years, security, development, and legal teams around the globe have relied on Black Duck® solutions to help them manage the risks that come with the use of open source. Built on the Black Duck KnowledgeBase™—the most comprehensive database of open source component, vulnerability, and license information available—Black Duck software composition analysis solutions and open source audits give organizations the insight they need to track open source in code, mitigate security and license compliance risks, and automatically enforce open source policies using existing DevOps tools and processes.

Black Duck Audits found open source in over 96% of codebases scanned in 2018, a percentage similar to the figures from the last two OSSRA reports. It's worth noting that most of the codebases found to have no open source consisted of fewer than 1,000 files. More than 99% of the codebases scanned in 2018 with over 1,000 files contained open source components.

The reality is that open source is not less secure than proprietary code. But neither is it more secure. All software, be it proprietary or open source, has weaknesses that might become vulnerabilities, which organizations must identify and patch.

The ubiquity of open source in both commercial and internal applications provides attackers with a target rich environment as vulnerabilities are disclosed through sources such as the National Vulnerability Database (NVD), mailing lists, GitHub issues, and project homepages. The widespread use of open source can lead to another issue: Many organizations don't keep accurate, comprehensive, and up-to-date inventories of the open source components used in their applications.

Can our readers say with confidence that the open source components used in their public and internal applications are up-to-date with all crucial patches applied? If you can't answer that question and can't produce a full and accurate inventory of the open source used in your applications, it's time to create a bill of materials for your open source. It's impossible to patch software when you don't know you're using it.

The average age of vulnerabilities identified in 2018 Black Duck Audits was 6.6 years, slightly higher than 2017—suggesting remediation efforts haven't improved significantly. The oldest vulnerability identified in the scans is probably older than some of our readers: CVE-2000-0388, a 28-year-old high-risk vulnerability in FreeBSD first disclosed in 1990.

Sixty-eight percent of the 2018 audited codebases contained components with license conflicts. Most common were GNU General Public License (GPL) license violations, with 61% of the codebases having some form of GPL conflict. This makes sense, as GPL is one of the most common licenses and one of the most likely to conflict.

Open source licenses generally fall into one of two categories: permissive and copyleft. Permissive licenses are sometimes referred to as "attribution style," and copyleft licenses are also known as reciprocal and viral licenses.

The permissive license is the most basic type of open source license. In most cases, it allows you to do whatever you want with the code as long as you acknowledge the authors of the code and follow other obligations such as redistribution and documentation requirements.

Copyleft licenses add further requirements to the permissive license. For example, if you distribute binaries, you must make the source code for those binaries available. You must make the source code available under the same copyleft terms as the original code. You can't place additional restrictions on the licensee's exercise of the license.

Many open source components in use are abandoned. In other words, they don't have a community of developers contributing to, patching, or improving them. If a component is inactive and no one is maintaining it, that means no one is addressing its potential vulnerabilities. Therefore, organizations determining the risks in a codebase must also consider the operational factors in open source components. Black Duck Audits found that 85% of codebases contained components that were more than four years out-of-date or had no development activity in the last two years

Use of open source itself is not risky; unmanaged use of open source is. To defend against open source security and compliance risks, we recommend organizations take these steps:

CREATE AND ENFORCE OPEN SOURCE RISK POLICIES AND PROCESSES. Educate developers about the need for managed use of open source. Put in place an automated process that tracks the open source components in a codebase and their known security vulnerabilities, as well as operational risks such as versioning and duplications, and prioritizes issues based on their severity.

PERFORM A FULL INVENTORY OF THEIR OPEN SOURCE SOFTWARE. Organizations can't defend against threats that they don't know exist. It's essential to obtain a full, accurate, and timely inventory of the open source used in their codebases. The inventory should cover both source code and information on how open source is used in any commercial software or binary deployed in production or used as a library in an application.

MAP OPEN SOURCE TO KNOWN VULNERABILITIES. Public sources, such as the NVD, are a good first stop for information on publicly disclosed vulnerabilities in open source software. Keep in mind, however, that over 90 organizations contribute entries to the NVD. Not only does the NVD reflect their priorities, but there can be significant lags in data reporting, scoring, and actionability of the data in a CVE entry.

Don't rely solely on the NVD for vulnerability information. Instead, look to a secondary source that provides earlier notification of vulnerabilities affecting your codebase and, ideally, delivers security insight, technical details, and upgrade and patch guidance.

CONTINUALLY MONITOR FOR NEW SECURITY THREATS. The job of tracking vulnerabilities doesn't end when applications leave development. Organizations need to continuously monitor for new threats for as long as their applications remain in service. Importantly, any continuous monitoring strategy must take into account the composition of the software under attack, lest it become overwhelmed by the volume of vulnerability disclosures we experienced in 2018.

IDENTIFY LICENSING RISKS. Failure to comply with open source licenses can put organizations at significant risk of litigation and compromise of IP. Educate developers on open source licenses and their obligations. Involve your legal advisors in the education process and, of course, in reviewing licenses and compliance with legal obligations.

MAKE SURE OPEN SOURCE IS PART OF M&A DUE DILIGENCE. If you're engaged in an acquisition, understand that the target company is using open source—and likely not managing it well. Don't hesitate to ask questions about their open source use and management. If the software assets are a significant part of the valuation of the company, have a third party audit the code for open source

Summary: A useful summary of open source and vulnerabilities. The recommendations include:

- create and enforce open source risk policies and processes.
- perform a full inventory of their open source software.
- map open source to known vulnerabilities.
- continually monitor for new security threats.
- identify licensing risks.
- make sure open source is part of mergers & acquisition due diligence

IEEE

The IEEE Center for Secure Design intends to shift some of the focus in security from finding bugs to identifying common design flaws — all in the hope that software architects can learn from others' mistakes. The website has Tweets from 2020 whilst much of the rest of the site dates to 2019.

A range of articles addressing:

- Building Code for the Internet of Things
- Building Code for Medical Device Software Security
- Avoiding the Top 10 Software Security Design Flaws
- Design Flaws and Security Considerations for Telematics and Infotainment Systems

British Computer Society

A range of short articles relevant to open source including:

Can open source be secure? By Steve Smith, Managing Director of IT security consultancy Pentura – During an open source project's lifetime, it usually forks off into a variety of different versions, depending on what developers require of the new application or operating system. Commercial organisations can often get involved in this, forking off a version of the open source application and placing some commercial backing to the project, typically involving a more structured development approach, a licensing model and structured support services.

This offers users the best of both worlds, where they can benefit from access to the open community of applications whilst still having someone to turn to if they have problems."

The truth about open source By Paul Adams BSc MBCS, chairman of the BCS Open Source Specialist Group. …. freedom is at the heart of open source and, in fact, it comes four-fold.

Freedom 0: The freedom to run the program, for any purpose

Freedom 1: The freedom to study how the program works and adapt it to your needs

Freedom 2: The freedom to redistribute copies so you can help your neighbour

Freedom 3: The freedom to improve the program, and release your improvements to the public, so that the whole community benefits - access to the source code is a precondition to this

Time to secure software By John Colley, Managing Director, (ISC)2 EMEA Since software, like anything else that goes through a manufacturing process, is designed and developed to a blueprint, it is of paramount importance that security requirements are determined alongside the functional and business requirements.

A preliminary risk assessment at this stage serves to determine the core security necessities of the software, while a security plan should be generated as part of the design phase of the project to be revisited and adjusted as the development progresses. In fact, we have to consider security at every point in the system's development life cycle:

1. Requirements

At the requirements stage, we should be considering what the security requirements are. Are there any specific business security requirements that need to be built in? Is the data being processed particularly sensitive? Are there any specific requirements that have to be met?

2. Design

From an architectural design viewpoint, consideration must be given to how the security requirements can be designed into the system. Design should also consider how the system might be misused or what could go wrong, a perspective missed by developers that instinctively think about how to build rather than break things. We should also think about what are the main threats to the design of the system - SQL injection is a typical example of a threat to the design of the system.

3. Coding

Coding is important, and we must make sure the coding is robust and secure. There are many tools that can be used to check coding, and code inspection can also be used in the testing phase. The approach to coding should be to trust nothing and to be able to process anything:

4. Testing and deployment

Testing should check the security functionality along with the other functionality, ensuring that the system is resilient to attack. And of course testing should look for incorrect system operations, as well as correct ones. Secure deployment ensures that the software is functionally operational and secure at the same time. It means that software is deployed with defence-in-depth and that the attack surface area is not increased by improper release, change, or configuration management.

Software that works without any issues in development and test environments when deployed into a more hardened production environment often experiences hiccoughs. Post-mortem analyses in a majority of these cases reveal that the development and test environments do not simulate the production environment. Changes therefore made to the production environment should be retrofitted to the development and test environments through proper change management processes.

5. Operations and maintenance.

It doesn't matter how well the system has been designed coded and tested, if it is operated insecurely, then all of that effort has been wasted. Maintenance presents two main problems. Firstly, any changes should be designed, coded and tested with the same rigour as the initial implementation.

Secondly, it is important to have a good change management and source management system. Too many upgrades are released with errors that were corrected in previous versions resurfacing in the new version. This is because the corrections were not retrofitted back into the previous development environment or because flawed source code was deployed within a broader program rather than a corrected version.

6. Disposal

Don't forget the disposal of the system. We have seen losses to reputation where data has been left on hard drives when systems have been replaced or updated. Make sure what has been left behind is considered both from a software and a hardware viewpoint."

The Software Alliance

The Software Alliance [SA] is the leading advocate for the global software industry before governments and in the international marketplace. Its members are among the world's most innovative companies, creating software solutions that spark the economy and improve modern life.

With headquarters in Washington, DC, and operations in more than 30 countries, [SA] pioneers compliance programs that promote legal software use and advocates for public policies that foster technology innovation and drive growth in the digital economy.

The Framework for Secure Software

The Framework identifies best practices relating to both organizational processes and product capabilities across the entire software lifecycle. It is organized into six columns: Functions, Categories, Subcategories, Diagnostic Statements, Implementation Notes, and Informative References. Functions organize fundamental software security activities at their highest level, consistent with the software lifecycle. The Functions are" Secure Development, Secure Capabilities and Secure Lifecycle.

Secure Development

- Secure Coding
- Testing & Verification
- Process & Documentation
- Supply Chain
- Tool Chain
- Identity & Access Management

Secure Capabilities

- Support for Identity Management & Authentication
- Patchability
- Encryption
- Authorization and Access Controls
- Logging
- Error & Exception Handling

Secure Lifecycle

- Vulnerability Management
- Configuration
- Vulnerability Notification & Patching
- End of life

Whitesource software

A series open software security-related whitepapers including:

- THE COMPLETE GUIDE ON OPEN SOURCE SECURITY

  While not perfectly synonymous to the term of open source management, Software Composition Analysis is the industry tool aimed at helping organizations to get a handle on their open source usage. When it was initially coined, it was meant to reference the process of creating inventory reports that could provide managers visibility over the composition of the components in their software. As time progressed, software development and security managers understood that they need much more than a mere inventory of their open source components, including dependencies. They need to ensure that they are using high-quality open source components without known vulnerabilities and open source licenses that fit their organization and business model. Currently, there are three technologies of SCA tools in the market:

  • Open source code scanning
  • Continuous open source components analysis
  • Open source effective usage analysis (or components impact analysis)

  Capabilities and accuracy differ between technologies and vendors, but the key functionalities are considered to be the following:
  • Generating open source inventory reports, including all dependencies
  • Identification and alerting on vulnerable open source components
  • Identification of open source licenses to ensure compliance
  • Ability to enforce license and security policies

  Even as shifting left helps to improve your software development process, it is not sufficient on its own as your sole practice for managing your open source components. In many cases, a vulnerability is found years after the impacted version was released and it may already be in deployed products. This means that companies should not only shift their open source security, they should also “shift right” to deal with newly discovered issues. By shifting right we mean that companies need to continuously review their open source inventory of deployed products to ensure they have not become exploitable to newly discovered vulnerabilities.

  - The Importance of open source security – Describes the 2 pillars of open source security: Vulnerability Detection (Prioritizing effective vulnerabilities, The end to false positives, Comprehensive Database) and Vulnerability Remediation (Pinpointing the path, Suggested Fixes, Automated workflows)
  - Use DevOps to Minimize Application Security Risks

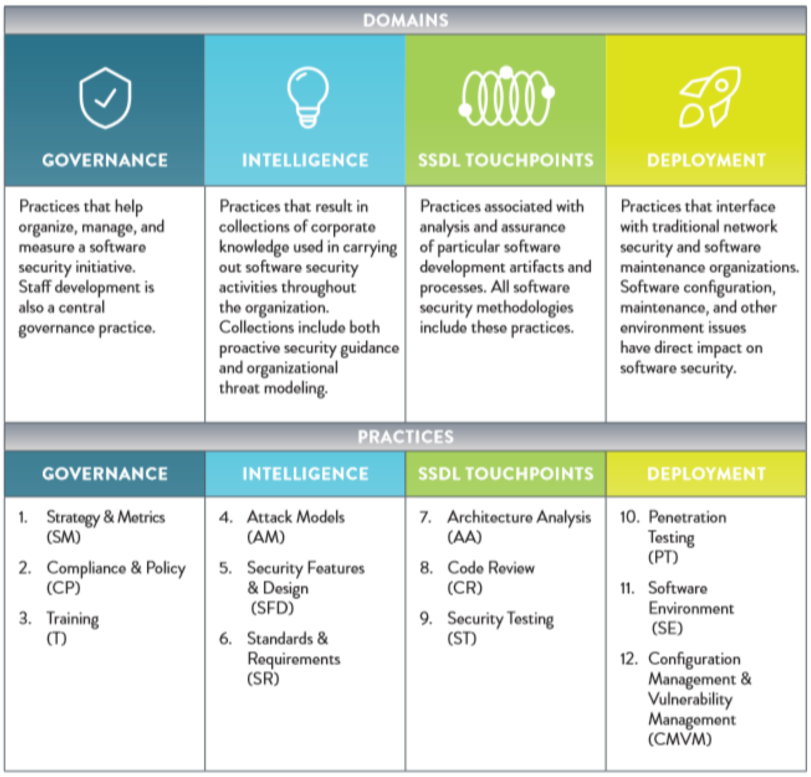
Building Security In Maturity Model (BSIMM)

The Building Security In Maturity Model (BSIMM, pronounced “bee simm”) is a study of existing software security initiatives. By quantifying the practices of many different organizations, we can describe the common ground shared by many as well as the variations that make each unique.

The most important use of the BSIMM is as a measuring stick to determine where your approach currently stands relative to other firms. You can simply note which activities you already have in place, find them in the SSF, and then build your scorecard. A direct comparison of all 119 activities is perhaps the most obvious use of the BSIMM. This can be accomplished by building your own scorecard and comparing it to the BSIMM10 Scorecard. As a summary, the BSIMM SSF comprises four domains—Governance, Intelligence, SSDL Touchpoints, and Deployment. In turn, those four domains include 12 practices, which contain the 119 BSIMM activities. A BSIMM scorecard indicates which activities were observed in an organization.

THE BSIMM10 FRAMEWORK

The BSIMM is organized as a set of 119 activities in a software security framework, represented here. The framework includes twelve practices that are organized into four domains.



DOMAINS			
GOVERNANCE	INTELLIGENCE	SSDL TOUCHPOINTS	DEPLOYMENT
Practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.	Practices that result in collections of corporate knowledge used in carrying out software security activities throughout the organization. Collections include both proactive security guidance and organizational threat modeling.	Practices associated with analysis and assurance of particular software development artifacts and processes. All software security methodologies include these practices.	Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance, and other environment issues have direct impact on software security.

PRACTICES			
GOVERNANCE	INTELLIGENCE	SSDL TOUCHPOINTS	DEPLOYMENT
1. Strategy & Metrics (SM)	4. Attack Models (AM)	7. Architecture Analysis (AA)	10. Penetration Testing (PT)
2. Compliance & Policy (CP)	5. Security Features & Design (SFD)	8. Code Review (CR)	11. Software Environment (SE)
3. Training (T)	6. Standards & Requirements (SR)	9. Security Testing (ST)	12. Configuration Management & Vulnerability Management (CMVM)

There are descriptions of each activity such as [CP2.4: 44] Include software security SLAs in all vendor contracts. Vendor contracts include an SLA to ensure that a vendor won’t jeopardize the organization’s compliance story or SSI. Each new or renewed contract contains provisions requiring the vendor to address software security and deliver a product or service compatible with the organization’s security policy (see [SR2.5 Create SLA boilerplate]). In some cases, open source licensing concerns initiate the vendor management process, which can open the door for additional software security language in the SLA. Traditional IT security requirements and a simple agreement to allow penetration testing aren’t sufficient here.

Illustrates the relative frequency of security activities surveyed across a range of industries. The model is at version 10 and illustrates the changing nature of activities over time.

Financial Services Information Sharing and Analysis Center (FS-ISAC)

The FS-ISAC published a report showing the application of a few of the techniques outlined previously including a derived version of BSIMM. The report is called Appropriate Software Security Control Types for Third Party Service and Product Providers.

Cloud Native Compute Foundation

The Cloud Native Computing Foundation (CNCF) hosts critical components of the global technology infrastructure. CNCF brings together the world's top developers, end users, and vendors and runs the largest open source developer conferences. CNCF is part of the nonprofit Linux Foundation

From 2019 Annual Report; In 2018, the CNCF began performing and open sourcing security audits for its projects to improve the security of our ecosystem. The goal was to audit several projects and gather feedback from the CNCF community as to whether the pilot program was useful. The first projects to undergo this process were Kubernetes, CoreDNS, and Envoy. In 2019, CNCF invested in security audits for Vitess, Jaeger, Fluentd, Linkerd, Falco, Harbor, gRPC, Helm, and Kubernetes, totaling approximately half a million dollars. These first public audits identified a variety of security issues, ranging from general weaknesses to critical vulnerabilities. Project maintainers for CoreDNS, Envoy, and Prometheus have addressed the identified vulnerabilities and added documentation to help users, thus improving the security of these projects.

With funds provided by the CNCF community to conduct the Kubernetes security audit, the Security Audit Working Group was formed to lead the process of finding a reputable third-party vendor. The group created an open request for proposals. The group took responsibility for evaluating the proposals and recommending the vendor best suited to complete a security assessment against Kubernetes, bearing in mind the project's high complexity and broad scope.

This audit process was partially inspired by the Core Infrastructure Initiative (CII) Best Practices Badge program that all CNCF projects are required to complete. Provided by the Linux Foundation, this badge offers a clear and easy-to-understand way for open source projects to show that they follow security best practices. Adopters of open source software can use the badge to quickly assess which open source projects are following best practices, and as a result, are more likely to produce higher-quality, secure software.

Findings from the Kubernetes audit conducted over a few months revealed:

1. Key security policies may not be applied, leading to a false sense of security.
2. Insecure TLS is in use by default.
3. Credentials are exposed in environment variables and command-line arguments.
4. Names of secrets are leaked in logs.
5. Kubernetes lacked certificate revocation.
6. seccomp is not enabled by default.

By open sourcing security audits and processes, the working group hopes to inspire other projects to undertake similar efforts in their respective open source communities. Full findings and recommendations from the audits are listed here.

CNCF also provide an overview of the cloud native landscape.

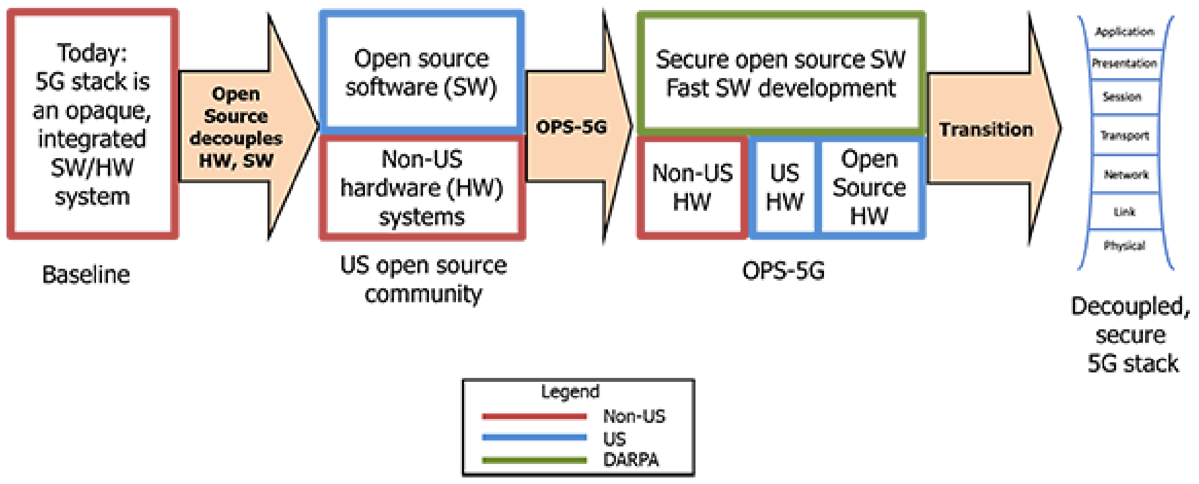
CNCF have an interesting tool – FOSSA. FOSSA claims to manage 'Visibility of 3rd party code', 'prioritise problematic dependencies', automatically compile compliance reports' (e.g. Bill of Materials) and 'Streamline license & vulnerability remediation'.

Center for Internet Security (CIS Controls v7.1)

The CIS Controls Implementation Groups (IGs) are self-assessed categories for organizations based on relevant cybersecurity attributes. Each IG identifies a subset of the CIS Controls that the community has broadly assessed to be reasonable for an organization with a similar risk profile and resources to strive to implement.   A large corporation with thousands of employees may be labeled IG3. There a range of recommended security controls and of particular interest is: CIS Control 18: Application Software Security Manage the security life cycle of all in-house developed and acquired software in order to prevent, detect, and correct security weaknesses. This presumes a full life cycle involvement that may not be possible for open source code but nonetheless has a set of useful controls.

DARPA

DARPA have begun (in 2020) a 4 year project called OPS-5G. DARPA created the Open, Programmable, Secure 5G (OPS-5G) program to tackle many of the security challenges facing future wireless networks. OPS-5G will explore the development of a portable, standards-compliant network stack for 5G mobile networks that is open source, and secure by design. The program seeks to enable a “plug-and-play” approach to various network software and hardware components, which reduces reliance on untrusted technology sources. The goal of OPS-5G is to develop open source software and systems that can enable more secure 5G as well as future generations of networks beyond 5G.


Today: 5G stack is an opaque, integrated SW/HW system
Baseline
Open Source decouples HW, SW
Open source software (SW)
Non-US hardware (HW) systems
US open source community
OPS-5G
Secure open source SW Fast SW development
Non-US HW
US HW
Open Source HW
OPS-5G
Transition
Application
Presentation
Session
Transport
Network
Link
Physical
Decoupled, secure 5G stack
Legend
Non-US
US
DARPA

Alliance for Telecommunications Industry Solutions

An interesting (more telco-focused) risk assessment methodology called Architectural Risk Analysis.

ATIS is developing an overall industry cybersecurity framework focused on the needs of the ICT industry. The work started by documenting the baseline of the current cybersecurity landscape, including existing ATIS initiatives and NIST/U.S. government cybersecurity frameworks and guidelines. From there it analyzed the expected threat landscape over the next three years.

Two reports have been published: 1) an Architectural Risk Analysis (ARA) Process for Security; and 2) an overview of IoT/M2M Cybersecurity activities and progress.

Cloud iNfrastructure Telco Task Force (CNTT)

CNTT was incubated in early 2019 through a partnership between GSMA and the Linux Foundation as a global open source taskforce comprised of industry-leading CSPs and NFVI/VNF suppliers. CNTT provides standardized infrastructures for both virtual machine-based and cloud native network functions, making it possible to deploy multiple network functions without having to create new infrastructures for each. This standardization enables providers to shorten deployment and onboarding from weeks and months to hours and days, reducing costs and accelerating digital transformation. Verification makes it possible to immediately determine whether a vendor's infrastructure is compatible with target network functions. This ability to "mix and match" components in a single infrastructure increases interoperability and enables more complex functionality.

All of this had led to a growing awareness of the need to develop more open models and validation mechanisms to bring the most value to telco operators as well as vendors, by agreeing on a standard set of infrastructure profiles to use for the underlying infrastructure to support VNF applications across the industry and telecom community at large. To achieve this goal, the cloud environment needs to be fully abstracted via APIs and other mechanisms to the VNFs so that both developers of the VNF applications and the operators managing the environments can benefit from the flexibility that the disaggregation of the underlying infrastructure offers.

The next step after the Reference Model has been identified and developed is to take the general model, which is purposely designed to be able to be applied to a number of technologies, and apply it to a discrete number of concrete and ultimately deployable Reference Architecture platforms. The intention is to choose the reference architectures carefully so that there will only be a small set of architectures that meets the specific requirements for supporting NFV and Telecom specific applications. Per the principles laid out in the Reference Model documentation, the Reference Architectures need to meet the following criteria as much as is practical:

Initially should be based on widely established technology and systems used in the Telecom Industry. This will help ensure a faster adoption rate because the operators are already familiar with the technology and might even have systems in production. Another advantage to this approach is a project faster development cycle.

Subsequent architectures should be based on either additional established or promising emerging technologies that are chosen by the community members.

Functional Scope

In terms of the functional scope of the CNTT documentation, in order to target the project goals as described above, we are focused on:

- Functional capabilities of the cloud infrastructure and the infrastructure management
- Functional interfaces between infrastructure and infrastructure management
- Functional interfaces between workloads and workload management

Due to the close alignment with ETSI GS NFV 002, those ETSI interfaces that are considered relevant (with notes where required) are included in the figure below.

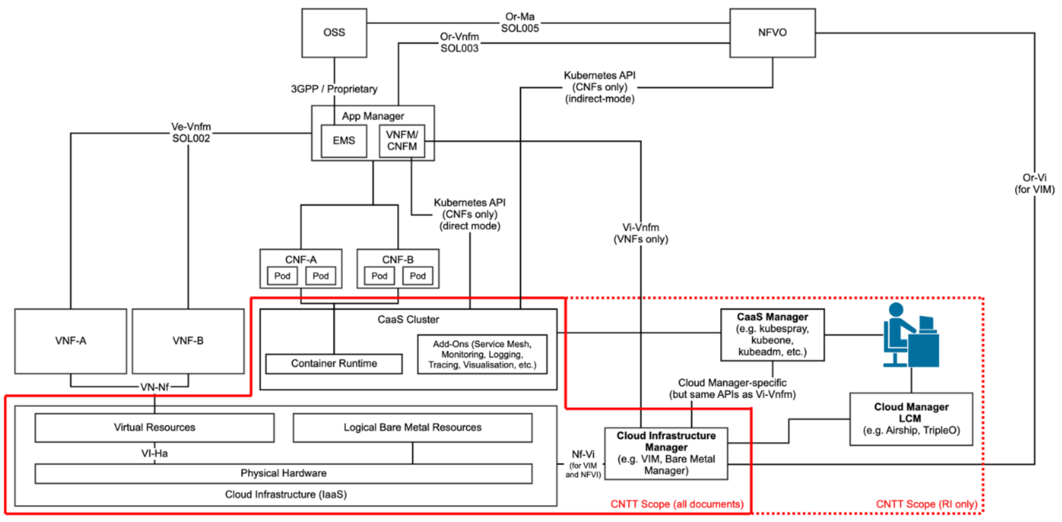


Figure 2: Functional Scope of CNTT

Conclusion & Next Steps

This initial GSMA report has summarised research undertaken for the open source software security project as part of the broader open networking initiative. The security concepts identified in the report and summarised in the table below are likely to form the basis of any set of good security controls especially within an open source software environment.

Concept / Organisation	NIST	NTIA	Safecode	OWASP	OASIS	LNF	NCSC (UK)	SSA	SynOpSys	BCS	BSA	Whitesource	BSIMM	ENISA	CNCF	DARPA	ATIS	CNTT
Zero Trust	🟥													🟥		🟥		
SBOM / Configuration Management		🟥	🟥						🟥		🟥	🟥	🟥		🟥			
Source Code Analysis	🟥			🟥		🟥			🟥			🟥			🟥			
Lifecycle approach	🟥	🟥	🟥	🟥				🟥	🟥	🟥	🟥		🟥	🟥				
Threat / Risk Model	🟥		🟥	🟥			🟥	🟥	🟥	🟥			🟥	🟥			🟥	
System and Component Approach			🟥				🟥											
Approved' Code / Re-use	🟥		🟥	🟥	🟥					🟥								🟥
Secure Coding / Avoid vulnerabilities			🟥	🟥		🟥	🟥	🟥			🟥		🟥	🟥	🟥			
Security Requirements (&Test)	🟥			🟥										🟥	🟥			
Open source licensing						🟥			🟥									
Cyber secure design				🟥			🟥									🟥		
Cloud / Virtualisation Security							🟥								🟥			
Secure by Default							🟥						🟥					
Toolchain	🟥						🟥					🟥						
Dev(Sec)Ops				🟥										🟥				

There is good overall coverage across the range of sources, however, many are written as if there is full influence over the development lifecycle (which may not be the case for open source software where some of the early stages (including Coding) may be delivered by others). This aspect may warrant further focus to identify, understand and manage the residual associated security risks.

The next stage of activity will seek to combine these security considerations with the output from a parallel project seeking mobile network operator feedback on their considerations for open source software security to identify deployment scenarios and associated security considerations. A later stage of activity will consider broader system security considerations particularly in the infrastructure area. This work has potential to identify a set of 'proof points' of testing and compliance against these security requirements. The longer-term aim is to build a set of best practice considerations for operators to consider when dealing with open source software. It is intended to complete this sequence of activity, however, it is recognised that this cycle may need further development as deployment scenarios and security controls evolve.

This report and Open Source Software Security project is undertaken as part of the broader GSMA Open Networking initiative where work is ongoing to define Minimum Viable Products and Use Cases. The Open Source Software Security project informs these models and provides high level considerations for security.