SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity

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

Table of Contents

1 Introduction

Usage of third-party components (TPCs) has become the de-facto standard in software development. These TPCs include both open-source software (OSS) and commercial off-the-shelf (COTS) components. According to a survey by Black Duck software[1], "78 percent of respondents said their companies run part or all of its operations on OSS and 66 percent said their company creates software for customers built on open-source. This statistic has nearly doubled since 2010 […]."

TPCs, used as pre-made building blocks, enable faster time to market and lower development costs by providing out-of-the box functionality of common functions, allowing developers to focus on product-specific customizations and features. While these TPCs are often treated as black boxes and are less scrutinized than comparable internally developed components, they are not without risk. Users inherit the security vulnerabilities of the components they incorporate. Historically, the selection and usage of TPCs has been an engineering decision, purely based on functionality. Given the increasing trend in usage of third-party components, security must be a consideration in the selection and usage of TPCs[2].

The number of vulnerabilities reported against TPCs, both OSS and proprietary COTS software, should serve as a strong testament that managing security risks due to the use of third-party components is an important duty for their users. Some good examples include Heartbleed (CVE-2014-0160[3]), which was disclosed in 2014, and more recently, a security flaw in the GNU C Library (CVE-2015-7547[4]) that was discovered by researchers in 2015. These vulnerabilities triggered analysis and remediation activities on an unprecedented scale that sent the software industry into a "patching frenzy."

To attackers, the fact that TPCs are widely used in software development is an introduction and invitation to an unexplored land of opportunities. The current state of uncontrolled TPC usage must be replaced by a disciplined analysis and consideration of security risk.

Disclaimer: This white paper focuses only on security risks inherent in the use of third-party components. Any other risks such as legal or regulatory risks, intellectual property, business risks, OSS vs. COTS quality or due diligence are out of scope for this white paper.

1.1 Methodology and Scope

The supply chain of components in software development is extremely varied and complex. There are many different use cases and considerations for TPCs, and there are several open-source and commercial tools offering capabilities to assist with the management of TPCs (see section 6.2). This white paper is the culmination of a yearlong research project within SAFECode that included surveys and industry research to identify best practices in TPC management. At the time of this research, no single body of work comprehensively addressed the issues with the usage of TPCs in product development.

This white paper provides a blueprint for how to identify, assess and manage the security risks associated with the use of third-party components. The white paper helps to understand these security risks and

[1] https://www.blackducksoftware.com/about/news-events/releases/seventy-eight-percent-of-companies-run-on-open-source-yet-many-lack-formal-policies-to-manage-legal-operational-and-security-risk
[2] https://www.computer.org/cms/CYBSI/docs/Top-10-Flaws.pdf
[3] https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160
[4] https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2015-7547

provides recommendations to help manage them. Through this white paper, SAFECode aims to share our collective knowledge regarding the challenges and recommended solutions for dealing with TPCs. The SAFECode publication "Principles for Software Assurance Assessment"[5] more broadly addresses software security assurance of commercial technology providers and thus includes use cases that are different from the one in this white paper. The table below summarizes these use cases, clarifying which paper covers which use case.

Use Case	Publication
An organization is evaluating/acquiring COTS software applications to install/use within a corporate network or environment. This is often done through a procurement process.	"Principles for Software Assurance Assessment"[5]
An organization is evaluating/acquiring open-source software applications to install/use within a corporate network. This is often done through a procurement process.	"An Assurance-Based Approach to Minimizing Risks in the Software Supply Chain"[6]
A development team in an organization is selecting or using open-source or COTS components for inclusion in products.	This white paper
An organization is contracting out custom development of a component or product.	Not in scope

For trainings on this and other topics, please refer to https://training.safecode.org.

2 Challenges in Using Third-party Components

This section introduces the importance of a well-established third-party component management life cycle by taking an example use case from the life of Bob, a software developer. This example use case highlights challenges faced by Bob in the absence of a TPC management life cycle which are discussed further in the rest of this section. Recommended solutions for managing third-party components that address these challenges are discussed in section 3.

2.1 Example Use Case

Bob's team is developing a product that incorporates a variety of third-party components (TPCs) to provide functionality the product needs. Bob's company does not have any requirements or restrictions on using third-party components. The integration of TPCs is quickly completed and the product is released and sold to customers. Not long after the production release, the internet is abuzz with reports of a new critical security vulnerability. The vulnerability was discovered in an open-source component and has

[5] http://www.safecode.org/publication/SAFECode_Principles_for_Software_Assurance_Assessment.pdf
[6] http://www.safecode.org/publication/SAFECode_Software_Integrity_Controls0610.pdf

been assigned an official Common Vulnerabilities and Exposures (CVE) number by the MITRE Corporation[7]. Bob's teammate Juanita forwards the CVE to Bob and asks him whether his product is affected. Bob encounters his first problem: ***"What third-party components are included in my product?"***

Bob mines his product documentation and code to attempt to determine what TPCs are included/used in the product. After much effort, a list of TPCs is generated. Now Bob encounters his next problem: ***"Is the product affected by the CVE?"*** The CVE lists the affected component as "Strawberry Lane" version 3.4. Bob's list of components does not contain Strawberry Lane version 3.4 but it does include StrwLn version 3.4. Bob wonders whether this is the same component. He does additional research and determines that it is, in fact, the same component, despite the different name. Bob must also evaluate whether or not the product uses the specific TPC functionality that is vulnerable. After extensive design and code reviews, Bob determines that the product is affected.

While Bob is evaluating the impact of the vulnerability, the product is compromised by attackers who succeed in accessing the data stored in the database server. Root cause analysis points to a serious vulnerability in a different TPC that has been known publicly for many years. Fixing these issues while in production results in financial loss, service outage, reputational exposure and decrease in customers' faith. Bob's company is responsible for any vulnerabilities in the product even though the vulnerability is in a third-party component and not the custom code that Bob added. Bob's team struggles with the question, "***What should we do to maintain the TPCs within our product?"***

The Quality Assurance & Product Security Incident Response teams at Bob's company insist on a full bill of materials (BOM) to identify all TPCs in the product and to make sure that other components are not vulnerable. During this analysis, Bob discovers that the internal product documentation only lists a few TPCs being used. His further analysis reveals that one of the TPCs in use actually depends on other TPCs internally. Frustrated, he realizes quickly that he does not know which TPCs are actually used by his product. The manual process of TPC discovery becomes labor-intensive, time-consuming and error-prone. Once the list of TPCs is ready, Bob now realizes that a few of these TPCs have reached end of life (EOL) and are therefore no longer supported. Further, he finds that these end-of-life components have unpatched security vulnerabilities that cannot be fixed because the vendors no longer support these components. The organization is now attempting to understand, ***"What TPCs should we use and what is the security risk associated with them?"***

This entire painful and costly experience has left Bob's company wondering, ***"How should we manage TPCs overall?"***

While this scenario focused on one particular product, in reality an organization may have tens, hundreds or thousands of products with code of varying age and complexity and a massive number of distinct TPCs. How to identify, assess, maintain and manage TPCs overall is a critical challenge facing software development organizations today.

2.2 What TPCs Are Included in a Product?

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

[7] https://cve.mitre.org/

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.

2.2.1 Naming of Components

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. For instance, Apache Xerces is often short for Apache Xerces/J, which is also xercesImpl.jar. Similarly, some components with the same name are available from different sources with different characteristics: For example, 7-zip is available for Windows in 32-bit x86 as well as in 64-bit x64 and for Linux, too. Finally, different TPCs could be identified by the same name, and a product may have multiple versions of a single TPC. The root cause of this challenge is a lack of unique identifiers or names for third-party components. These naming inconsistencies originate both outside and inside a company. Outside a company there is no standard naming convention for TPCs, and names vary significantly across software suppliers. Inside a company, the absence of a company-wide naming convention results in different development teams naming the same TPC differently.

2.2.2 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. An example of multi-level component dependencies is illustrated in Figure 1 in a dependency graph of an NPM (Node Package Manager) project that relies on the Express framework.

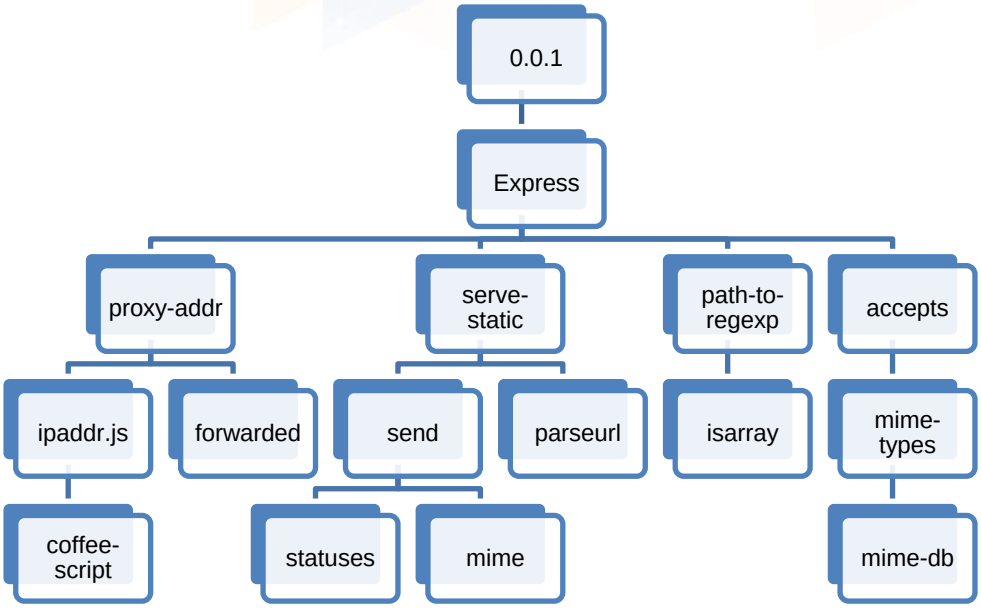

Figure 1: A small part of the dependency graph for the "0.0.1" NPM Module

Collecting and mapping software dependency information is often a product-team-driven effort, relying on individual product development teams to supply dependency information from build systems, references in source code or system architecture documentation. Individual product teams may use varying conventions in identifying, collecting and describing their product dependencies. These partly manual efforts are prone to error, and they produce data that is often unreliable and unusable in managing components and component risk at a portfolio or organizational level.

Criteria for discovery of these components-of-components may differ among product teams. What defines a component? How many levels deep should these references be followed? Who is collecting the data and what is the collector's background, level of understanding, and business objective? How should the data be represented? Even when organizational standards are developed for identifying and describing components, the standards may be unworkable, as the interpretation of these standards, and how these standards are proceduralized, will differ wherever there is human involvement.

Heterogeneous environments present additional challenges, as TPC dependencies are incorporated differently by language, framework and platform. Consider the import statement, often employed to use TPCs, which varies by programming languages, such as "import" in Java, but "include" in PHP and Ruby. Frameworks/platforms maintain the status quo: For example, maven maintains dependencies in a POM file, whereas Bower uses a JSON file for this. A company that uses several of these languages and frameworks then needs a tool that understands these differences to be able to find included TPCs.

2.3 Is the Product Affected by the Vulnerable Third-party Component?

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.

2.3.1 Naming of Components

Section 2.2.1 described the issue of TPCs not having unique identifiers and the impact this has on determining the product BOM. This same problem makes it very difficult to match the components in the BOM with the TPC listed on the CVE to determine whether the affected TPC is included in the product.

2.3.2 Dependencies

Section 2.2.2 described the issue of dependencies within TPCs. This same problem makes it very difficult to determine whether the affected TPC is included in the product. If the product BOM only includes the first or top-level set of TPCs, then vulnerabilities in nested components may go undetected for a period of time.

2.3.3 CVE Reports

CVEs listed in the National Vulnerability Database exist in varying levels of quality and completeness. The contents of the CVE may not be sufficient for a team to quickly determine whether or not its usage of a component is affected. In general, a detailed analysis of the CVE and additional research may be needed.

2.4 What TPCs Should We Use and What Are the Security Risks Associated with Them?

Product teams and developers often select third-party components purely based on the functionality they deliver, without considering the security, supportability and maintainability of these components. 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. Consider the following examples of third-party components that could carry high levels of security risk:

- A component created by a graduate student or intern, designed without any consideration of software security. Once delivered, the component is immediately abandoned by the author and never touched again.
- Source code made available on GitHub, but with build instructions pointing to outdated compilers, and with a last update over four years ago
- Hobby code written by a single author as an experiment or while learning to program, left available on the author's personal website

While these components may deliver desired functionality, they may bring in an unacceptable level of risk to the organization, especially if a product has been integrated into a critical business function or infrastructure. Selecting components that were developed with security in mind rather than choosing solely based on functionality will help lower an organization's security risk. Organizations that integrate components should understand the challenges and implement a balanced approach to risk management.

2.5 What Should We Do To Maintain the TPCs Within Our Product?

Maintaining used or incorporated TPCs is an important task during the entire product life cycle until the product finally reaches end of life and end of support. Maintenance includes responding to vulnerabilities discovered in TPCs used in the product. This task, however, is not an easy one and appropriate preparation is required. Without having plans and a strategy to address the following issues, risks may arise regarding:

- Keeping track of security weaknesses and vulnerabilities
- Monitoring and updating
- Unused TPCs
- TPCs reaching their end of life or end of support

3 Managing Third-party Components

Addressing the challenges associated with using third-party components requires a robust process that is integrated into the organization's software development life cycle (SDLC). Management of TPCs should begin as early as possible in the SDLC. Organizations should define and adopt a process for managing the security risk of TPCs that fits into an organization's existing SDLC. Section 3.1 provides a high-level overview of the proposed TPC management process and its relation to the software development life cycle. Section 3.2 describes the key ingredients of a TPC management process and discusses each step in detail. Section 3.3 revisits the use case.

3.1 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.

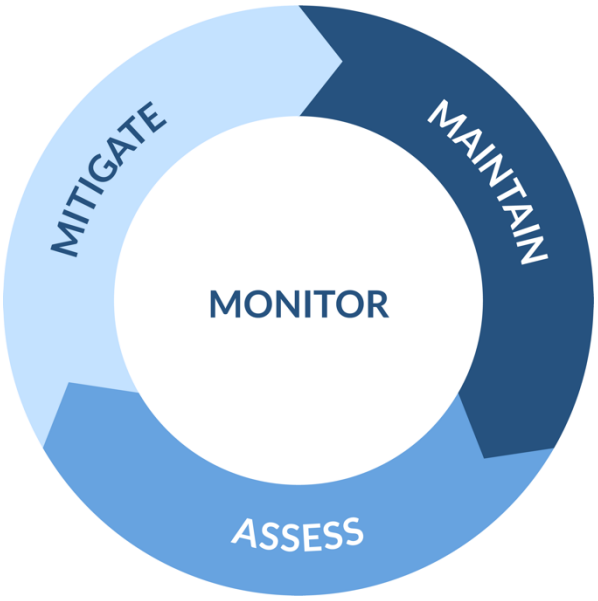

Figure 2: The high-level steps of TPC management are Maintain, Assess, and Mitigate. The Monitor step is a central aspect of TPC management and thus valid throughout the entire life cycle.

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. For new code, this could be as simple as keeping a list of third-party components of interest, including their versions. For legacy code, identifying the baseline set of TPCs included can be quite challenging, particularly for large codebases. There are many ways to generate and maintain a list of TPCs used, none of which is perfect. Section 3.2.1 explores some of the options available for

discovering and maintaining the list of TPCs. Regardless of what method is chosen, you cannot manage what you do not know. It is imperative that organizations adopt a process and method for identifying TPCs used.

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. This score could be binary -- acceptable/unacceptable -- or numeric for more advanced TPC management programs that may allocate weights to various aspects critical to an organization's business.

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.

3.1.1 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.

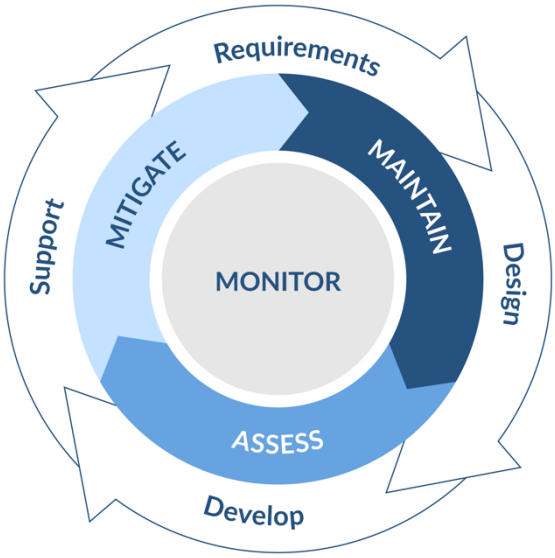

Figure 3: The TPC life cycle and the Software Development Life Cycle go hand in hand. This figure shows the repetitive nature of the both life cycles. Figure 4 shows how individual stages in these life cycles correlate.

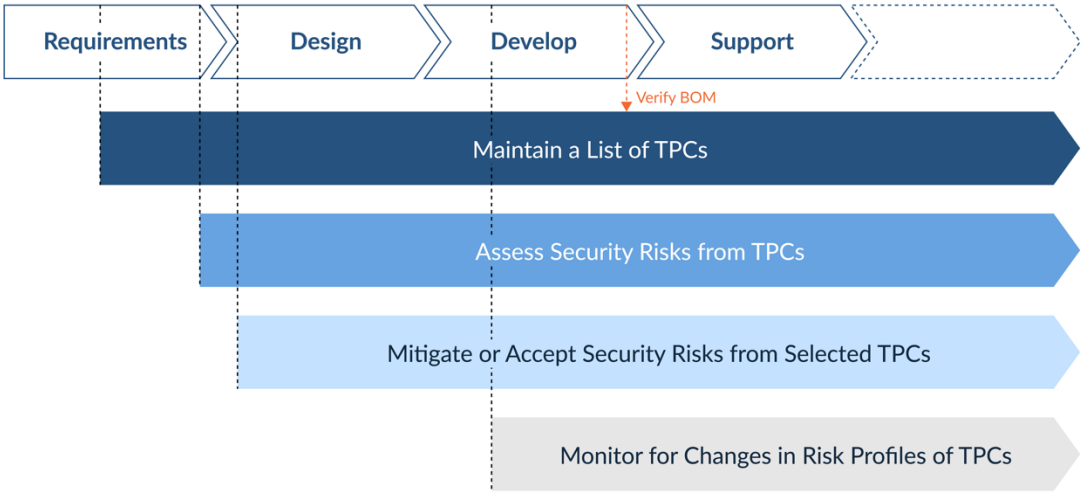

Figure 4: TPC life cycle steps should start early and occur at distinct points of the SDLC.

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. Different parts of an application may be in any of the four SDLC phases. Hence, maintaining this list of TPCs is a continuous process that could systematically yield a bill of materials for the application.

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 missed during the develop phase and that the BOM reflects reality. This verification entails the use of manual/semi-automated means (some tools are listed in Table 2) to find TPCs in use and compare them against the manually created BOM from previous phases.

3.2 Key Ingredients of a TPC Management Process

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.

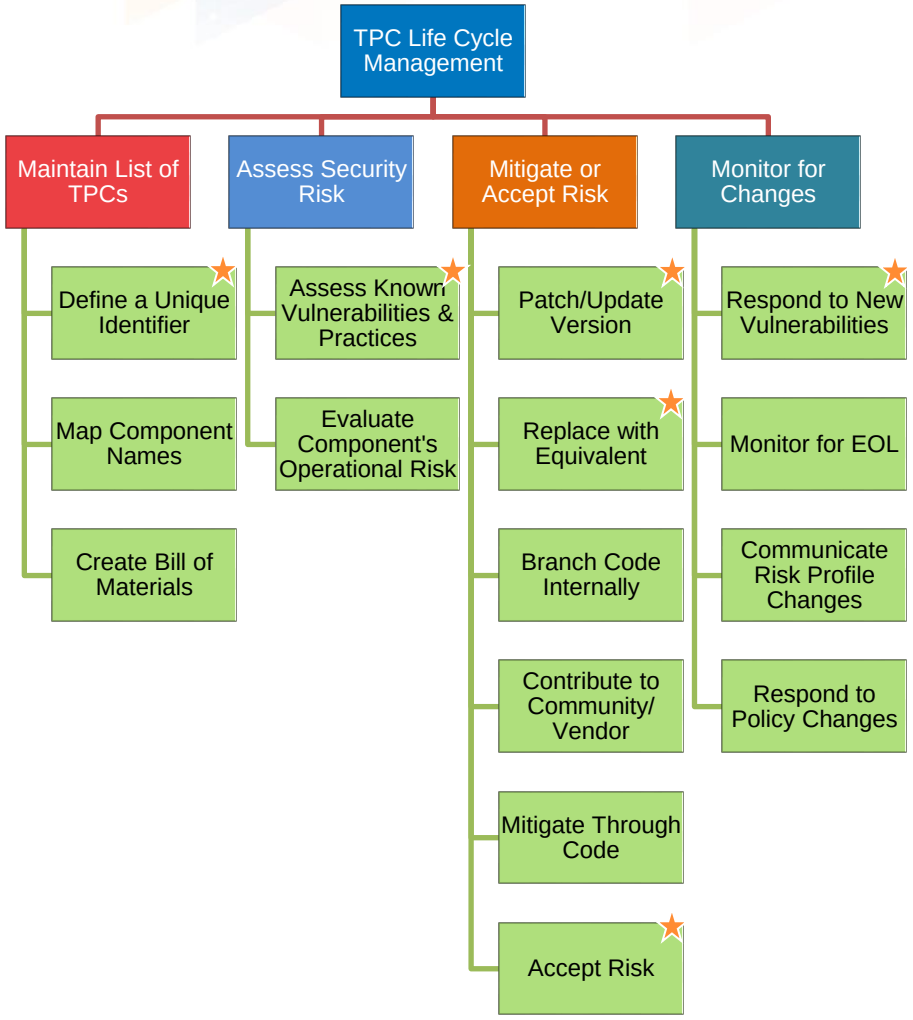

Figure 5: This chart depicts all TPC life cycle management steps of the TPC management process. Green boxes with stars are considered the bare minimum.

3.2.1 Maintain List of TPCs (MAINTAIN)

MAINTAIN1: Define a Unique Identifier. Defining a unique identifier primarily addresses the naming challenge described in section 2.2.1 and helps in the creation and maintenance of a BOM. Naming issues with third-party components should be approached in two different ways: (1) defining a unique identifier for company-wide adoption, and (2) mapping components to their appropriate identifiers.

In order to be able to assign identifiers to third-party components without collisions, a unique identifier solution has to be defined as the basis. For instance, such an identifier can be a simple integer value, a globally unique identifier (GUID) as used in computer software, or an existing naming standard or convention. Another key consideration in deciding upon this identifier should be its compatibility with other naming standards, in order to use the identifiers for querying different internal or external databases and in case a need arises to convert TPCs to a standard specified naming scheme. Homebrewed solutions that rely on a single identifier used only internally should be employed with caution, as these might complicate the compatibility of TPC names with other standards. For a company-wide approach, project-specific names should be disallowed in favor of using names that are valid across the company.

The following are some standards and conventions that are worthy of mention:

- Common Platform Enumeration (CPE) provides a standard machine-readable format for encoding names of IT products and platforms (https://cpe.mitre.org/about/).
- ISO/IEC 19770-2:2009 establishes specifications for tagging software to optimize its identification and management (http://www.iso.org/iso/catalogue_detail.htm?csnumber=53670).
- The Software Package Data Exchange (SPDX) specification is a standard format for communicating the components, licenses and copyrights associated with a software package (http://spdx.org/).
- Software Identification (SWID) Tags record unique information about an installed software application, including its name, edition, version, whether it is part of a bundle and more (http://tagvault.org/swid-tags/).
- The FS-ISAC working group recommends a naming convention[8] to facilitate the controls for third-party applications delivered to financial services institutions.
- The Maven "Project Object Model" (POM) is an XML representation of a Maven project and provides corresponding artifact naming: <groupid>:<artifactid>[:<version>] (https://maven.apache.org/guides/mini/guide-naming-conventions.html).

The CPE naming scheme, for instance, is suitable for the assignment of unique identifiers, and from a security standpoint it is definitely wise to use CPE. However, keep in mind that there are cases where no CPE name might exist, such as when there is not yet a CVE entry in the National Institute of Standards and Technology (NIST) database. A common approach in this case is to create a so-called inferred CPE name[9] based on the CPE naming specification[10] and hope that this inferred CPE name will become an official CPE name in the NIST dictionary for exactly the same TPC in the future. In contrast to that approach and especially to make the unique identifier solution reliable and deterministic, the SAFECode TPC Working Group proposes a different approach, which is depicted in Figure 6: a unique identifier for each TPC is at the center of this approach, and all alternative properties such as corresponding CPE name are associated with this identifier.

The identifier solution should encompass characteristics that represent third-party components in general and that make it possible to clearly and uniquely identify each TPC. At a minimum, this should include the name of the vendor, author or manufacturer, product or component name and version. It is further highly recommended that the TPC identifier solution include additional attributes or tags to a component entry, especially those that can be used to query external databases, such as NIST's National Vulnerability Database (NVD), for vulnerability lookup (see Figure 6). Each attribute or tag should be easily searchable, to find and identify existing TPCs in the database and prevent duplicates. It should be possible to add as much information as is available to a third-party component entry in this database. An all-embracing solution might even need to go further to cover additional characteristics such as minor version, architecture, origin and other aspects.

[8] https://www.fsisac.com/article/appropriate-software-security-control-types-third-party-service-and-product-providers
[9] https://www.tenable.com/blog/common-platform-enumeration-cpe-with-nessus
[10] https://cpe.mitre.org/specification/

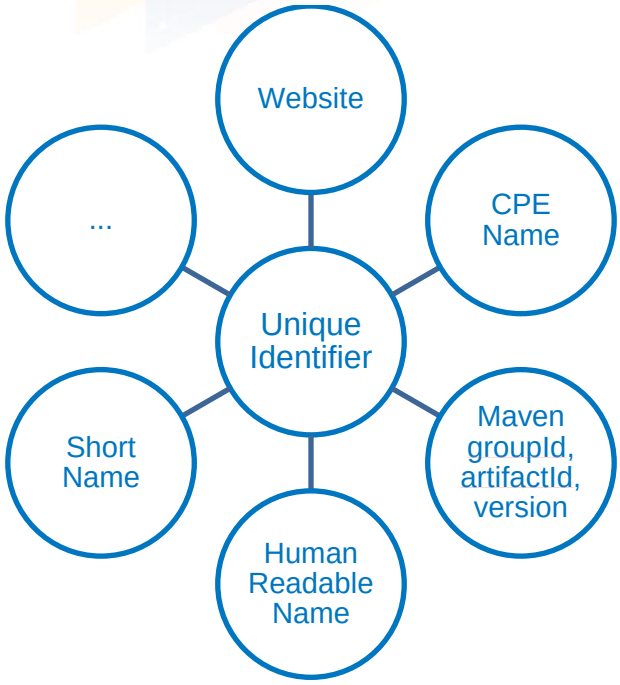

Figure 6: The recommended identifier solution should have a unique identifier and include as many attributes or tags as are available, to make each TPC uniquely identifiable.

MAINTAIN2: Map Component Names. When a suitable identifier solution is in place, effort must be spent on assigning correct identifiers to the third-party components in a product in order to create a valid bill of materials. The name-mapping aspect requires a system to map all names by which a TPC could be known to a single unique identifier (essentially to add tags/attributes to the unique identifier as shown in Figure 6). The goal of this mapping is to benefit from useful data that may be associated with each non-unique name of a TPC, such as a new vulnerability found, patch available, etc. Intuitively the naming system facilitates interaction with all such data sources and mapping their useful data to the associated unique TPC identifier. For TPC name data originating from a naming standard outside a company, such as the CPE database, for instance, strict control of correctness is important. Since external sources like the CPE database are constantly being updated and names are deprecated and replaced, name mappings have to be kept up to date. Otherwise, all advantages of using an external naming standard vanish.

After establishing a naming convention for software products and their TPCs, the next step in developing a program to address software supply-chain risk is to identify and catalog the TPCs used in an organization's own products.

MAINTAIN3: Create a Bill of Materials. Organizations should adopt a top-down, uniform approach to component identification and collection. It is recommended that you establish a company-wide standard that clearly articulates the requirements for dependency identification and management. Processes need to be established and implemented to ensure conformance to the standard. Using a technical and automated solution for identifying and collecting software component information is advisable. This technical solution should be at the core of the TPC management process, with supporting procedures developed where product languages, frameworks and platforms fall beyond the scope of the standard technical solution. These supporting procedures should additionally be developed and maintained by the

TPC process owner and should be extended as required to support the organization's software products. Measurements of process success should be established, and compliance with the component management standard should be the responsibility of application owners. The objective is to produce a usable consistent set of data on TPC use across all of an organization's software products.

Table 1 summarizes the advantages and disadvantages of different approaches that can be applied to create and maintain a bill of materials.

Table 1: Overview of different methods to create and maintain a bill of materials and their pros and cons

Method	Pros	Cons
Manually generate and maintain a list based on what developers attest to including	• Free	• Low accuracy: developers may not list all the TPCs in use, especially for legacy or inherited code. • Cost to maintain: manual effort by all developers is required to maintain the list.
Use automated scanning tools	• Highly accurate, for languages scanned • Highly efficient	• Cost to purchase • May require multiple tools: all scanning tools are limited in the languages and frameworks they support. Large codebases incorporating many languages will probably require multiple tools for complete coverage.
Combination of manually generated list and scanning tools (in the event that the organization does not possess scanning tools that cover all languages)	• Combination of pros of manual generation and using automated scanning tools	• Combination of cons of manual generation and using automated scanning tools

Organizations should adopt appropriate means to find TPCs in their codebases (some example tools are listed in Table 2 in the appendix). As a TPC may itself depend on other TPCs, this analysis can be done recursively until no new TPCs are found. Such recursive dependency exploration could yield a large number of dependencies, such as can be seen in Figure 1. Initially organizations may decide to limit the depth of this exploration to focus on immediate dependencies. Limiting exploration to immediate dependencies may not identify high-risk TPCs that may be deeply nested. Ideally, the exploration should be done until no new dependencies are found, and then policies should be defined to identify an actionable subset of TPCs. For example, if we have a vulnerable TPC one level deep in a single air-gapped product, it could be marked low priority. However, if we have a vulnerable component that is five levels deep, used in 90% of products, then it should be a high priority to resolve this.

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 third-party components from the approved list are used. Components in this store should fulfill certain security

requirements, which have been validated beforehand according to a predefined policy but which should also be revalidated on a regular basis.

Having a centralized store also enables large, distributed teams to track which products or services use particular components. This way, if a vulnerability is discovered in a component, the affected teams could be promptly notified. This might include keeping third-party components manageable by limiting the number of the same or very similar components, such as by having a set of strategic or recommended components in place. Organizations that use third-party components must strike a balance between the benefits (e.g., time to market, specialized functionality, re-use of commodity components, etc.) and the risks, of which security is only one aspect. Selecting components that do have a positive reputation regarding security is a big help toward achieving the benefits and not incurring as much risk.

3.2.2 Assess Security Risk (ASSESS)

When TPCs are identified, they must be evaluated and assessed for security risk. The security risk of a component is dependent on multiple factors, including the maturity of the vendor's secure development practices as well as the context in which the component is used. The SAFECode "Principles for Software Assurance Assessment"[11] paper describes a tiered approach for assessing risk based on transparency and maturity of the supplier's practices.

Security risk is not influenced solely by vulnerabilities but also by a variety of other attributes. A component that is not actively maintained is unlikely to be patched quickly following the discovery of a vulnerability. The security risk associated with using a third-party component is highly context dependent. For example, an authentication library used to protect a critical business function should undergo greater scrutiny than a calendar widget used on the organization's intranet home page. If a vulnerability is identified, the impact of that vulnerability on the product or service that uses the component should be considered. Determining the impact is an essential part of the assessment. A proper impact assessment will reduce cost, by allowing an organization to focus its efforts on vulnerabilities that will have significant impact. For example, if a component has a vulnerability on a code path that a product does not exercise, it might be considered lower priority. A strong understanding of the attack surface of a product or service will go a long way toward producing a meaningful risk assessment.

ASSESS1: Assess Known Vulnerabilities & Practices. Most software suppliers have established mechanisms for handling and reporting security weaknesses and vulnerabilities, ranging from private support forums to public issue trackers to security-specific email addresses. Organizations that use third-party components should understand the processes for disclosure and response to security vulnerabilities to limit the risk of such vulnerabilities to customers.

Known Vulnerabilities

For known vulnerabilities, the National Vulnerability Database and other sources such as proprietary databases to which the organization may have access, project histories, public reports, etc., can be queried for vulnerabilities associated with the component in question. It is important that the organization query using all common names for the component, due to the lack of industry standard naming conventions.

Note: Publicly known vulnerabilities, especially their quantity, are an imperfect indicator of the security risk of a component. It can be tempting to see public vulnerabilities as a sign that a component has poor

[11] http://www.safecode.org/publication/SAFECode_Principles_for_Software_Assurance_Assessment.pdf (see section 1.1)

security, but a healthy cycle of responding to reported vulnerabilities can in fact show that the supplier has a strong focus on ensuring that the component is secure. However, recurrence of the same types of vulnerabilities may point to deep-rooted security issues. Conversely, having no publicly reported vulnerabilities is not necessarily a good thing; it could be that no one is looking for or openly communicating vulnerabilities, and there are hidden vulnerabilities yet to be discovered. If the source code of a third-party component is available, static source code analysis tools can be used to check for security weaknesses or vulnerabilities.

The following should be considered:

- How many publicly known security vulnerabilities have not been remediated in the latest version or version in use?
- Do any of the publicly known open vulnerabilities in the latest version not contain mitigating steps to reduce or eliminate the risk?
- How quickly are publicly known vulnerabilities corrected, once identified?
- Are the same types of publicly known vulnerabilities reoccurring over time?

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[12], CVE[13], Common Attack Pattern Enumeration and Classification[14], etc.)?
- Does the community/supplier routinely disclose vulnerabilities and prepare customers for patch deployment?

[12] https://cwe.mitre.org
[13] https://cve.mitre.org (see section 2.1)
[14] https://capec.mitre.org

- 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?

A bug notification process can be as simple as a web form, a public bug tracking database or an email address published for the purpose of reporting security findings. Providers should have a published, coordinated, responsible disclosure policy that promotes a non-punitive response to responsible feedback and coordination with security researchers. Bug bounties are becoming increasingly popular. Though many providers struggle to respond well to a high volume of responders to their bounty programs, these programs are generally a sign that a provider has some confidence in its security program and some capacity to address findings, and it is likely doing a fair amount of internal security testing of its products.

Robust secure development practices are more likely to yield secure products. When choosing a component, consider the secure development practices that went into creating that component. Evaluate the community/supplier's published practices:

- Do the supplier's organization-level and software security policies include requirements that aim to produce a high-quality, repeatable result?
- Does the supplier implement a secure development process that includes activities for requirements definition, design, implementation, and test phases?
- Does the supplier include secure coding standards in the software security policy?
- Was the component developed under an established security process, such as a secure development life cycle?
- What security assurances are available for the component (e.g., security assessments, results of automated security testing, history of responding to security vulnerabilities)?
- What processes are used to validate code changes prior to release (e.g., code reviews, automated security testing, etc.)?
- Is there a documented test plan and are test suites used to test the component?
- Does the community/supplier advertise its application security controls (formal security requirements)?
- Does the community/supplier perform automated static code reviews to identify security defects introduced during coding?

ASSESS2: Evaluate the Component's Operational Risk. A well-maintained component is more likely to have vulnerabilities identified and remediated quickly. Organizations that use third-party components need to understand the responsibilities of their software suppliers regarding short- and long-term maintenance, and should be prepared to perform the necessary updates to remain on a stable, supported component version.

In general, components that have been actively used for a long time tend to be more mature, complete and stable. They tend to be actively maintained and often have mature, rapid processes for fixing vulnerabilities when discovered. Stability of the TPC is an important attribute that influences the security risk of using the component. The age and maintenance cycle of a TPC should not be used as an indicator of the likelihood of vulnerabilities, but rather as an indicator of the likelihood of a rapid response from the

supporting organization when a vulnerability is discovered. TPCs that are not actively maintained are less likely to be evaluated or patched for a new vulnerability quickly.

Abandoned components, i.e., those which have reached their end of life, can present particular challenges, since the organizations using such components are often forced to choose among (a) finding an alternative component, (b) maintaining the component themselves, if possible, or (c) continuing to use the component. Option (c) is especially risky, since security vulnerabilities could be discovered at any time, and abandoned but popular components can be a rich target for attackers.

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?

3.2.3 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. If a patch or update is available, it should be applied. There are two different forms of this scenario: (a) installing a newer version of a TPC or (b) installing a patch. By installing a newer or the latest version, a TPC is replaced as a whole. It is important to note that by moving to a newer version, there is more effort required, as performance, features or interfaces can change. However, in applying a patch, only parts of a TPC are changed during the patching process. TPC users are advised to adopt a regular patching process to keep up with planned and unplanned patches/updates. This approach could minimize the need to patch reactively, and it scales better when TPC usage is high.

MITIGATE2: Replace with an Equivalent. Often there are many TPCs that offer similar functionality. The team may choose to adopt a different TPC (that satisfies security risk assessment criteria) to replace a vulnerable component. As an example, consider a library that is used for reading compressed files. Assume this library is affected by a critical vulnerability and that it poses a huge risk to the product that uses that library, because potentially malicious files from the internet are opened. If the vulnerability

remains unpatched because the supplier no longer supports the component or is just unwilling to provide a patch, the library can and should be replaced by an alternative that fulfills the same requirements; that is, opening compressed files. Of course, this is only feasible if the vulnerable library is not too tightly integrated into the product and the new library satisfies equal or better security risk criteria.

MITIGATE3: Branch Code Internally. Depending on the TPC license conditions for use, the team may take the code in-house and remediate the vulnerability. This is not a preferred method, as now the team has taken on additional technical debt and is responsible for maintaining the code. This option should only be pursued if the community/vendor that maintains the TPC is unwilling or unable to remediate the vulnerability in the necessary timeframe.

MITIGATE4: Contribute to Community/Vendor. The availability of open-source components is made possible through the contributions of a diverse set of developers willing to share their creations with the broader development community. Thanks to their hard work and generosity, teams are able to increase productivity through efficient reuse of existing components. Anyone consuming open-source components should seek to contribute back to the community. Contributing to the identification and remediation of security vulnerabilities is one opportunity to give back. The Core Infrastructure Initiative[15], for example, "is a multi-million dollar project to fund and support critical elements of the global information infrastructure" organized by The Linux Foundation and supported by various companies. Projects receive funds from the initiative "to assist the project in improving its security, enabling outside reviews, and improving responsiveness to patch requests."[15]

MITIGATE5: Mitigate Through Code. Depending on the vulnerability, sometimes additional code can be added to the organization's product that effectively wraps the TPC call/usage and mitigates the vulnerability.

MITIGATE6: Accept Risk. There are circumstances under which an organization may choose to accept the risk and not mitigate the vulnerability. For example, if the vulnerability exists in a code path of the TPC that the team is not actively using, the vulnerability may be low risk for this usage. The team may choose not to patch instantly, but instead to accept the risk for a period of time until the next scheduled update/release. Organizations should adopt a process and set of criteria for accepting risk of TPC vulnerabilities.

3.2.4 Monitor for TPC Changes (MONITOR)

MONITOR1: Respond to New Vulnerabilities. All product development organizations must establish a process for discovering and responding to new vulnerabilities. Many large organizations establish a Product Security Incident Response Team (PSIRT) capability. PSIRT is considered a standard industry best practice. PSIRT teams typically have methods to monitor and receive vulnerability reports for both the organization's products and its TPCs.

Whether or not the organization has a PSIRT function, the product team must monitor all the third-party components used in its products for reported security vulnerabilities and patch these as necessary to mitigate known vulnerabilities. Unfortunately, this can be quite painful and time-consuming if done manually by monitoring CVEs or the vendor/community that maintains the components. Options for monitoring include:

[15] https://www.coreinfrastructure.org/

- Monitoring public vulnerability data stores (such as the NIST CVE database) for notices of vulnerabilities that affect the component
- Registering for the appropriate supplier-specific notification mechanisms (GitHub, supplier announcement mailing list, etc.)

Commercial and open-source tools are available that can scan a codebase (primarily source code but also possible for binaries, to some extent), identify TPCs used and alert the user when a used component has a CVE filed against it. These tools enable rapid detection and notification of vulnerabilities. If an organization is using a lot of TPCs, it might want to consider investing in one of these tools to reduce the pain of monitoring and the risk of not discovering a known vulnerability until too late.

MONITOR2: Monitor for End of Life (EOL) and Usage. Users of TPCs should establish internal guidance for 1) how to determine whether a TPC has reached its EOL and 2) what to do when a TPC reaches its EOL. If a TPC has reached its EOL, it is not being maintained or updated to address vulnerabilities or weaknesses. Sometimes, the entire TPC reaches EOL (meaning all versions); sometimes just a particular version or branch of the TPC reaches its EOL, and consumers/users must update to use the current supported branch. Unused TPCs should be identified and removed accordingly, while appropriate plans have to be in place for how to treat used TPCs. The guidance and handling of a situation may be dependent on many factors, such as the security risk for the product using the TPC, the contract maintenance period for the product using the TPC, and the alternatives available to replace the functionality provided by the TPC.

Monitoring for EOL can be very difficult for an organization, especially when TPCs from different commercial suppliers and open-source communities are included in the organization's products. Commercial vendors usually publish roadmaps and accurate end-of-life information on their websites. Open source communities, however, may not formally plan and publish roadmaps with future releases or EOL information. It can be very difficult to determine whether an open-source component is still active or has reached its end of life. For example, for many open-source projects hosted on SourceForge or GitHub, it is unclear whether a component is still supported. Projects that do not have active maintainers or have long periods of inactivity (lack of code commits) may be indicators of a possible EOL state. Unfortunately, there is no single set of rules to determine EOL status that can be applied consistently and correctly to every TPC. Organizations must establish their own criteria for determining when a component is at risk of reaching its EOL and monitor TPCs against those criteria.

There are commercial software tools available that identify TPC components in software and provide reports on the components (see some example tools in Table 2). These tools can also search for known vulnerabilities affecting these components and flag components that have not been updated since a predetermined time. Some of these tools allow the user to define those thresholds to flag components as being stale, and possibly as having reached their EOL.

In general, if a TPC has not been updated in over a year, it is perceived by many (including several commercial software packages that assess risks of TPCs) to be effectively unmaintained and at EOL. While this is not always actually the case, it is reasonable to at least flag such a TPC for further investigation of its status. Once it has been determined that a TPC has effectively reached EOL, the user of the TPC must determine how to mitigate the risk associated with an orphaned TPC. The steps outlined in section 3.2.3 can be applied for the handling of EOL components. The most common step is to replace the TPC with a suitable equivalent.

MONITOR3: Communicate Risk Profile Changes. Some organizations have centralized teams that monitor TPCs for vulnerabilities. Some organizations require each product team to monitor its own TPCs for vulnerabilities. In both scenarios, it is important that the presence of a new vulnerability be communicated throughout the organization to ensure that everyone whose products are affected by a vulnerability is aware of the vulnerability and responds appropriately. Every organization must establish a process and requirements for monitoring and communicating TPC vulnerabilities.

MONITOR4: Respond to Policy Changes. New vulnerabilities in TPCs are discovered every day. In trying to manage the security posture of products, organizations should frequently revisit and modify internal processes, policies and best practices regarding TPC usage and management. A TPC that could have acceptable security risk one day might be banned from use the next day. Product teams must understand and be prepared to respond to policy changes that affect their usage of TPCs.

3.3 Closing the Example Use Case

In contrast to Bob's company in the example use case introduced in section 2.1, John's organization considered the question, "***How should we manage TPCs overall?"*** and established a robust TPC management process and toolset. This process includes procedures and criteria for vetting the security risk of TPCs, a company-wide naming standard for TPCs, and a repository to track what TPCs are used and in what products. This investment by the organization results in a rapid and efficient process for dealing with the risks associated with TPCs.

3.3.1 Selecting TPCs

John is developing a product and needs TPCs that can provide certain capabilities. His developers have compiled a list of TPCs they would like to use. John and his team are faced with the question, "***What TPCs should we use and what is the security risk associated with them?***" John and his team review the list of approved TPCs that have been vetted for security risk in the organization's TPC repository. Many of the components his team wishes to use are already on the list and have been approved for usage. Some of the TPCs John's team wants to use are not on the list, but alternative, already approved TPCs are listed. John asks his team to review the alternative TPCs and determine whether they are sufficient for usage in his product. The team determines that some of the alternatives can be used, but there are three TPCs they need that are not listed. The team engages with the security risk assessment process for the vetting of those three TPCs.

3.3.2 Monitoring TPCs

John's organization established a process and automation for monitoring the TPCs used in their products. A new CVE has recently been filed against a TPC used in many of the organization's products. All product owners are notified of the new CVE shortly after its publication.

3.3.3 Responding to New Vulnerabilities

John selects a web server and a database server, the same ones that Bob selected in section 2, to include in his product. When the product is in production, the TPC management program in John's company warns him about a new publicly known security vulnerability in the chosen version of the web server TPC. John consults his list of TPCs used in the product to determine "***What third-party components are included in my product?***" Because John's organization has established a robust process and toolset for tracking TPCs used, John is able to quickly determine and verify that his product is using the specific component and version affected by the vulnerability. John promptly completes

remediation, and his company is able to issue a timely response to customers about the patches. All other affected teams in John's organization are able to quickly and efficiently identify and remediate as well.

3.3.4 Maintaining the TPCs in the Product

John's organization also considered the question, "***What should we do to maintain the TPCs within our product?***" and decided to establish a process for a quarterly review of the status of TPCs used within each product. They use both tools and manual processes to collect information about the components they use. This helps John greatly in prioritizing patch application, as well as starting investigations to replace components that have reached EOL. In reviewing the frequency of updates to his product's TPC components, John realizes that some of his components have effectively reached EOL, as they have not had a new release in more than two years and there are no longer any active maintainers. John updates the central TPC repository with his assessment that some components appear to be at EOL so that other teams can benefit from his analysis. John makes plans for future releases to gradually replace these EOL components. One EOL component is widely used in many products the organization owns and is critical to those products' success. In reviewing the impact, the organization decides to give back to the community and to staff maintainers for this component, allowing the organization to continue to use it with confidence and also enabling the addition of features that would benefit its products.

John's organization proactively defined a plan, process, criteria and supporting toolset to implement a robust TPC management life cycle. The example above was simplistic and focused primarily on the impact to a single development team. The return on investment in this TPC management life cycle scales up with both the number of development teams and number of independent TPCs. Regardless of the size and volume of an organization's development scope, every organization must consider how it will manage TPCs and identify processes and tools to address overall TPC management.

4 Future Considerations

Risks and challenges, as well as the corresponding countermeasures that are described above, need to be addressed not by one single company or entity but by different parties in a cooperative effort. This section of the white paper discusses what the SAFECode TPC Working Group believes can be done in joint efforts to improve the overall situation and to reduce risks inherent in the use of third-party components.

4.1 Crowdsourcing of Naming and Name Mapping

Sections 3.2.1 (MAINTAIN1) and 3.2.2 (MAINTAIN2) addressed the issues of uniquely naming TPCs and associating meaningful characteristics with such unique names (e.g., replacing alternative names with unique names). Given the volume of TPCs out there, any single organization implementing those solutions would face the problem of scale. Prioritization schemes such as dealing first with more frequently used TPCs, etc., would help. In general, such problems may be better dealt with in a collaborative (think crowdsourced) effort. In such an effort, individual organizations can contribute unique names (applying a standard naming scheme) and provide meaningful attributes for a small set of TPCs, and would benefit from similar contributions from others. The resulting repository/knowledgebase would enable effective implementation of unique naming as well as name-mapping solutions discussed previously.

4.2 Crowdsourcing of an End-of-life Repository

There is currently no central database available providing dates that tell when a third-party component will reach the end of life; i.e., when it will no longer be supported by its vendor and security vulnerabilities will remain unfixed. Today, this information has to be collected in a time-consuming manner by every organization by itself from different websites on the internet. The SAFECode TPC Working Group recommends setting up a crowdsourced database that contains third-party components with end-of-life information and corresponding dates.

4.3 Crowdsourcing of a Vulnerability Source Listing

Every supplier of a third-party component has its own way of providing information on security vulnerabilities and patches. With the increasing use of third-party components, an organization would have to monitor all these sources or, alternatively, use a vulnerability provider to perform this task. However, there exists no central and all-embracing list with references to where security and vulnerability information can be found for a vendor or component. A crowdsourced vulnerability source listing would be very helpful to both TPC-using organizations and vendors.

5 Summary

Software developers commonly use and incorporate third-party components, including open-source software, commercial off-the-shelf and proprietary software, in their code and in their products. This practice has become the de-facto standard in software and product development today. While the benefits are clear, with time to market and cost savings leading the way, the industry as a whole has only recently recognized the inherent security risks of TPC usage and the necessary steps to mitigate these risks. Third-party components cannot be blindly used as-is without responsible, disciplined consideration and evaluation. As this document has attempted to point out, these components can be used responsibly with proper processes, tooling and infrastructure necessary to assess and manage the security risk of such components.

The third-party component management life cycle described in this paper provides an approach for practitioners, software developers, managers and everybody who is using or intends to use third-party components. This paper proposes a lightweight and easy-to-use TPC life cycle that can easily be aligned to an existing software development life cycle to tackle security risks due to the use of third-party components. For those who are new to this field and would like to start quickly, Figure 5, the TPC Life Cycle Management Steps, provides quick starter elements that should be considered a minimum for meaningful TPC life cycle management.

SAFECode and the TPC Working Group behind this work perceive this paper as a compilation of best practices and recommendations for anyone using or intending to use third-party components. Though much of the information presented is a result of innovation happening internally within individual software companies, SAFECode believes that this industry collaboration has amplified these efforts and contributed positively to advancing the state of the art across the industry.

To continue this positive trend, SAFECode encourages software providers not only to consider, tailor and adopt the ideas outlined in this paper, but also to continue to contribute to a broader industry dialogue on advancing processes for addressing security risks due to the use of third-party components. For its part, SAFECode will continue to review and update this paper based on the experiences of our members and the feedback from the industry and other experts. To this end, we encourage your comments and contributions. To comment on this paper, please write to feedback@safecode.org. To contribute, please join SAFECode or visit www.safecode.org.

5.1 Acknowledgements

5.1.1 Contributors

- Prithvi Bisht, Adobe
- Mike Heim, Boeing
- Manuel Ifland, Siemens
- Michael Scovetta, Microsoft
- Tania Skinner, Intel

5.1.2 Reviewers

- Nazira Carlage, Dell EMC
- Brian Glas, Microsoft

- Shaun Gilmore, Microsoft
- Steve Lipner, SAFECode
- Matt MacNeil, Dell EMC
- Jim Manico, Manicode Security
- Nick Ozmore, Veracode
- Kristen Pascale, Dell EMC
- Joe Jarzombek, Synopsys

SAFECode also acknowledges the efforts of its Technical Director, Tom Brennan, in the development of this paper.

5.2 About SAFECode

The Software Assurance Forum for Excellence in Code (SAFECode) is a non-profit organization exclusively dedicated to increasing trust in information and communications technology products and services through the advancement of effective software assurance methods. SAFECode is a global, industry-led effort to identify and promote best practices for developing and delivering more secure and reliable software, hardware and services. Its charter members include Adobe Systems Incorporated, CA Technologies, Dell EMC, Intel Corp., Microsoft Corp., Siemens AG, and Symantec Corp. Associate members include Autodesk, Boeing, Huawei, NetApp, Security Compass, Synopsys, Veracode, and VMWare. For more information, please visit www.safecode.org.

6 Appendix

6.1 TPC Provider's Security Mindedness/Posture Assessment

In addition to assessing a third-party component itself, a similar process should be established to assess the security posture of a TPC provider. The following list provides examples of other publications on assessing third parties:

- SAFECode's "Principles for Software Assurance Assessment" provides a framework for examining the secure development process of commercial technology providers (http://www.safecode.org/publication/SAFECode_Principles_for_Software_Assurance_Assessment.pdf).
- Google's Vendor Security Assessment Questionnaire (VSAQ) is an interactive questionnaire application that supports security reviews by facilitating not only the collection of information, but also the redisplay of collected data in templated form (https://github.com/google/vsaq).
- FS-ISAC Third Party Software Security Working Group, "Appropriate Software Security Control Types for Third Party Service and Product Providers," Version 2.3/October, 2015 (https://www.fsisac.com/article/appropriate-software-security-control-types-third-party-service-and-product-providers)
- Software Assurance Maturity Model (SAMM) is an open framework to help organizations formulate and implement a strategy for software security that is tailored to the specific risks facing the organization (http://www.opensamm.org/).
- Building Security In Maturity Model for vendors (vBSIMM) is concerned with measuring large numbers of vendors in order to assess secure software development lifecycle maturity and control risk (https://www.bsimm.com/about/bsimm-for-vendors/).

6.2 Related Work

There is an existing and growing ecosystem of open-source and commercial tools and services that address various challenges associated with third-party component management. This section is not intended to be a comprehensive analysis of the space, but rather only to touch on the types of tool and services available. Furthermore, this section is not intended to give recommendations on which tool, service or provider one should choose, or to endorse a certain company, tool, service or provider. Instead, the SAFECode TPC Working Group highly recommends that the reader use these resources to obtain a picture of the tool landscape, in order to choose the most suitable solution for a certain environment and use cases.

6.2.1 Identification of Third-party Components and Dependency Management

Dependency management solutions focus on software products that allow the identification of third-party components that a product depends on. There are a few open-source tools, but the majority of tools on the market seem to be commercial. The listed tools, especially the commercial ones, usually not only identify OSS and COTS components, but also provide component management and cover checking open-source license obligations.

Table 2: Tools for identification of TPCs and dependency management

Name	Vendor	Type	Description	Reference
Binary Analysis Tool (BAT)	Tjaldur Software Governance Solutions	Comm.	Looks inside binary code to find compliance issues and reduce uncertainty when deploying free and open-source software	http://www.binaryanalysis.org/
BinDiff	Google	OSS	Performs fingerprinting and static flow analysis, which allows comparing binaries in order to check whether two binaries are identical. Can be used to identify libraries an application depends on	https://www.zynamics.com/bindiff.html
DependencyCheck	OWASP	OSS	Identifies project dependencies and also checks for known vulnerabilities (CVE). Supports Java, .NET, Ruby, Node.js and Python	https://www.owasp.org/index.php/OWASP_Dependency_Check
Hub	Black Duck	Comm.	Creates inventories of used open-source software. Maps to known vulnerabilities	https://www.blackducksoftware.com/products/hub
Nexus	Sonatype	Comm.	Provides a continuous component management system to support development teams in choosing healthy components and not using risky components	http://www.sonatype.com/products-sonatype
Palamida Enterprise	Palamida	Comm.	Scans open-source and binary files to detect open-source software and other third-party code in development projects	http://www.palamida.com/products/enterprise
Protecode	Synopsys	Comm.	Finds known vulnerabilities and license violations in the software supply chain before they become legal liabilities or business risks	http://www.synopsys.com/software
Protex	Black Duck	Comm.	Automatically scans, identifies and inventories open-source software	https://www.blackducksoftware.com/products/protex
Victims database	Red Hat	OSS	Maps JAR file SHA-512 hashes to CVE IDs	https://victi.ms/

6.2.2 Vulnerability Databases

Security vulnerabilities affecting third-party components as well as corresponding patches should be monitored and addressed accordingly. For their own products, of course, many vendors supply security vulnerability information as well as information on patches, either publicly or to paying customers directly. These range from security bulletins from large software companies to issues tagged “security” on an open-source project’s web page. Unfortunately, these web pages are usually in different formats and would have to be monitored constantly. However, there are a few free or commercial vulnerability information sources that provide uniform vulnerability and patch information. Table 3 provides a non-comprehensive list of free and commercial vulnerability information databases.

Nearly all databases report in a vulnerability-centric manner; i.e., a single CVE/vulnerability per published advisory. Flexera was found to report on a patch basis, where multiple vulnerabilities or CVE numbers are included in a single advisory.

Table 3: A few examples of free and commercial vulnerability information databases

Name	Type	Description	Reference
NIST National Vulnerability Database	Free – vulnerability-centric	U.S. government repository of standards-based vulnerability management data	https://nvd.nist.gov/
Risk Based Security VulnDB	Commercial – vulnerability-centric	Vulnerability intelligence through a continuously updated data feed	https://www.riskbasedsecurity.com/vulndb/
Flexera Software Vulnerability Intelligence Manager (VIM)	Commercial – patch-centric	Vulnerability intelligence and tools to support software vulnerability management	http://www.flexerasoftware.com/enterprise/products/software-vulnerability-management/vulnerability-intelligence-manager/
Symantec DeepSight	Commercial – vulnerability-centric	Threat intelligence and information on vulnerabilities	https://www.symantec.com/services/cyber-security-services/deepsight-intelligence