A Proposal to Operationalize Component Identification for Vulnerability Management

The SBOM Forum
September 13, 2022

Executive Summary

One of the biggest problems in software security today, and a huge inhibitor of the automation of production and use of software bills of materials (SBOMs), is the “naming problem” – that is, the fact that a single software product is known by different names in different ecosystems, and there is no “canonical” name that will work everywhere. This problem is most acute in the National Vulnerability Database (NVD), the most widely used vulnerability database in the world.

An informal group of software security professionals – we call ourselves the “SBOM Forum” – has developed this proposal, which describes a small set of steps that can be taken to alleviate a large portion of this problem (our solution addresses both software naming and the related problem of hardware naming). Most importantly, all the technologies involved in our proposal are in wide use today and are available either free or at low cost.

Introduction

One of the biggest challenges in software security today is known as the “naming problem”. Briefly stated, this refers to the problem that there is no universally agreed-on name for any open source software component or product, and there is often a lot of uncertainty about the name of a proprietary component or product. For example, the name by which an open source product is known in one package manager or registry is often different from its name in a different package manager or registry. Conversely, in different package managers or registries, the same name and version string might not refer to the exact same product. And the names of proprietary products will often be confused due to mergers and acquisitions.

These considerations are especially important because securing software usually involves identifying vulnerabilities in databases organized by product name and version string. Even though serious vulnerabilities may have been identified for a product, a user may never find them if they don’t know the exact name by which the component or product is identified in the vulnerability database.

The primary locus of this challenge is the National Vulnerability Database[1] (NVD), the most widely used software and hardware vulnerability database in the world. Software products are identified in the NVD with a CPE[2] (Common Platform Enumeration) name; CPE names are assigned by NIST (the National Institute of Standards and Technology, part of the US Department of Commerce). The many problems with CPE names (see below) make it difficult to locate a product in the NVD.

The primary impact of this problem is that it is currently impossible to reliably automate many processes required for software security. An important example of this is the processes required for suppliers to produce software bills of materials (SBOMs) and for software consumer organizations to utilize SBOMs to secure software deployed on their networks. These processes will be challenging to fully automate until the naming problem is wholly or substantially solved; currently, the problem is a long way from being solved in any meaningful way.

The individuals named below are all involved with the SBOM ecosystem. We have all been impacted by the naming problem and we know the cost in productivity of having to constantly deal with it. We have been discussing the problem weekly for more than five months and have – to our surprise, given our initial skepticism - identified a set of steps that in principle can go a long way toward solving the naming problem. Even more surprisingly, implementing these steps will not be technically difficult, since our solution relies entirely on technologies that are already well understood and in widespread use by private industry and governments.

This document describes the problem we want to solve and the existing technologies (naming systems) that compose our solution. Because CPEs are also used to name hardware products in the NVD and are subject to some of the same problems as CPEs for software, we have also

[1] https://nvd.nist.gov/

[2] Common Platform Enumeration - Wikipedia

proposed supplementing hardware CPEs with two existing hardware identifier classes, currently in use worldwide.

Nothing we are proposing will require any proprietary technologies or specialized services. Every piece of our solution is currently in widespread use and available either as open source or as a low-cost proprietary solution. Because the naming problem affects software security in many ways, the benefits from addressing the problem will flow well beyond SBOM uses.

The Challenge

As mentioned above, the NVD plays an important role in cataloging hardware and software components and the vulnerabilities that are applicable to them. The NVD uses the CPE identifier to identify hardware and software components, each identity consisting primarily of the vendor, product, and version string. CPEs are not native to the hardware or software industries, and are only employed by the security community due to the importance of the NVD.

Bills of Materials (BOM) consumers often want to look up their component inventory in one or more databases of known vulnerabilities ("vulnerability intelligence" sources). In the case of the NVD, doing this usually requires knowing the CPE name for the component. However, for various reasons (see below) it is often impossible to find a CPE for a particular component, whether open source or proprietary. Oracle Corporation estimates they can identify CPEs for no more than 20% of the components in their software products. Should this situation not change, it will significantly inhibit the hoped-for widespread distribution and use of SBOMs, both by federal agencies in response to Executive Order (EO) 14028[3] and by private sector organizations.

Below are six problems that result from the NVD's reliance on CPE as the sole identifier. Every one of these problems would be substantially solved if our proposal were adopted by the NVD (NIST) and by CVE.org, which manages the "CVE" vulnerability identifiers. Note that our proposal does not require discontinuing or changing the current CPE system, but merely supplementing it.

1. Vulnerabilities are identified in the NVD with a CVE number, e.g. "CVE-2022-12345". A CPE is typically not created for a software product until a CVE is determined to be applicable to the product. However, many software suppliers have never identified a CVE that applies to their products, so they have never created a CPE for them. This is almost certainly not because the products have never had vulnerabilities, but because the suppliers, for whatever reason, have not submitted any vulnerability reports for those products for inclusion in the National Vulnerability Database.

   The worst part of this problem is that the result of an NVD search will be the same in both cases - the case where a vulnerability has never been identified in a product and

---

3 https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

the case where the supplier has never felt inclined to report a vulnerability, even if they may have identified some. The search will yield “There are 0 matching records” in both cases. Someone conducting a search won’t know which case applies and may believe the product has no vulnerabilities, when in fact the supplier has simply never reported one.

2. There is no error checking when a new CPE name is entered in the NVD. Therefore, if the CPE name that was originally created for the product does not properly follow the specification is entered, a user who later searches for the same product and enters a properly-specified CPE will receive an error message. Unfortunately, it is the same error message that they would receive if the original CPE name were properly specified but there are no CVEs reported against it: “There are 0 matching records”.

   In other words, when a user receives this message, they might interpret this to mean that there *is* a valid CPE for the product they’re seeking, but a vulnerability (CVE) has never been identified for that product - i.e. it has a clean bill of health. However, in reality it would mean the CPE name was created improperly. In fact, there might be a large number of CVEs attached to the off-spec CPE, but without knowing that name, the user will not be able to learn about those CVEs.

   Another explanation for the “There are 0 matching records” error message is that the user had misspelled the CPE name in the search bar. Again, the user would have no way of knowing whether this was the reason for the message, or whether the message means the product has no reported vulnerabilities.

   It is to avoid problems like this that most organizations that use the NVD employ advanced search techniques based on AI or fuzzy logic[4]. While that can greatly reduce the number of unsuccessful searches, having to resort to this makes it impossible to conduct truly automated searches. Considering that an average-sized organization might easily need to conduct tens of thousands of NVD searches per day and a service provider doing this on behalf of hundreds of customers would need to conduct some large multiple of that number, the magnitude of this problem should be apparent.

3. When a product or supplier name has changed since a proprietary product was originally developed (usually because of a merger or acquisition), the CPE name for the product may change as well. Thus, a user of the original product may not be able to learn about new vulnerabilities identified in it, unless they know the name of the current supplier as well as the current name for the product. Instead, this user will also receive the “There are 0 matching records” message.

4. The same holds true for supplier or product names that can be written in different ways, such as “Microsoft(™)” and “Microsoft(™) Inc.”, or “Microsoft(™) Word” and “Microsoft Office(™) Word”, etc. There is no easy way to distinguish the correct supplier or product name among a large number of query responses.

[4] Or, in the case of at least one third-party service provider, a “small army” of CPE-resolvers.

5. Sometimes, a single product will have many CPE names in the NVD because they have been entered by different people, each making a different mistake. For this reason, it will be hard to decide which name is correct. Even worse, there may be no “correct” name, since each of the names may have CVEs entered for it. This is the case with OpenSSL (e.g. “OpenSSL” vs “OpenSSL_Framework”) in the NVD now. Because there is no CPE name that contains all of the OpenSSL vulnerabilities, the user needs to find vulnerabilities associated with each variation of the product's name. But how could they ever be sure they had identified all the CPEs that have ever been entered for OpenSSL?

6. Often, a vulnerability will appear in one module of a library. However, because CPE names are not assigned on the basis of an individual module, the user may not know which module is vulnerable, unless they read the full CVE report. Thus, if the vulnerable module is not installed in a software product they use but other modules of the library are installed (meaning the library itself is listed as a component in an SBOM), the user may unnecessarily patch the vulnerability or perform other unnecessary mitigations.

What is needed is to be able to name software and hardware components in a BOM with an identifier that, when entered in the NVD, will

1. Almost always match to the correct product, if the product is listed.
2. Almost never match to an incorrect product.
3. Not require that the identifier already exist in the NVD. This is almost always required today, in order for the user to get a correct response. However, if the user searches on a CPE name that doesn’t exist in the NVD, the error message they receive, “There are 0 matching records”, is the same one they would receive if the CPE does exist, yet it has no reported vulnerabilities.
4. Never yield a result that might be interpreted to mean the product was found but there are no applicable vulnerabilities, when in fact one of the following is the case:
    a. The wrong identifier was entered in the search bar; or
    b. An off-spec identifier (CPE) was initially created for the product, so the product cannot be found by searching on an identifier created according to the spec; or
    c. The name and or/supplier of the product has changed due to a merger or acquisition. Thus, the identifier entered by a user of the original product doesn’t match the current CPE name.
5. Identify the vulnerable module in a library rather than just the entire library, so that, if that module isn’t installed in a product but other modules are installed, users will not patch or perform other mitigations that are not necessary.
6. When a supplier and/or product name has changed for a product, allow there to be separate identifiers - and thus separate locations to report CVEs - for the different supplier or product names; thus the different supplier/product names will be treated as separate products.

The two principles behind our solution

1. Intrinsic vs. extrinsic identifiers

Our solution is based on two principles. The first of these is the distinction between intrinsic and extrinsic identifiers. As described in an excellent article[5], extrinsic identifiers "use a register to keep the correspondence between the identifier and the object", meaning that what binds the identifier to what it identifies is an entry in a central register. The only way to learn what an extrinsic identifier refers to is to make a query to the register; the identifier itself carries no information about its object.

A paradigmatic example of an extrinsic identifier is Social Security numbers. These are maintained in a single registry (presumably duplicated for resiliency purposes) by the Social Security Administration. When a baby is born or an immigrant is given permission to work in the US, their number is assigned to them. The only way the person "behind" a Social Security number can be identified is by making a query to the central registry (which of course is not normally permitted) or by hacking into the registry.

By contrast, intrinsic identifiers "are intimately bound to the designated object". They don't need a register; the object itself provides all the information needed to create a unique identifier. What intrinsic identifiers need is an agreed-on standard for how that information will be represented. Once a standard is agreed on, anyone who has knowledge of the object can create an identifier that will be recognizable to anyone in the world, as long as both creator and user of the identifier follow the same standard.

An example of an intrinsic identifier is the name of a simple chemical compound. As the article states, "We learned in high school that we do not need a register that attributes different identifiers to all possible chemical compounds. It's enough to learn once and for all the standard nomenclature[6], which ensures that a spoken or written chemical name leaves no ambiguity concerning which chemical compound the name refers to. For example, the formula for table salt is written NaCl, and read *sodium chloride*."

In other words, simply knowing information about the makeup of a simple chemical compound is sufficient to create a name for the compound, which will be understandable by chemists that speak any language - as long as you follow the standard when you create the name. NaCl refers to table salt, no matter where you are or what language you speak[7].

An example of an *extrinsic* identifier that is very pertinent to our proposal is a CPE name. CPE names are assigned by a central authority (NIST) and stored in a register. Whenever a user

---

[5] https://www.softwareheritage.org/2020/07/09/intrinsic-vs-extrinsic-identifiers/

[6] https://en.wikipedia.org/wiki/Chemical_nomenclature

[7] More complex compounds may require a CAS Registry Number, which is a centralized database.

searches for a CPE name in the National Vulnerability Database, the register is searched to determine which entry the name refers to.

Only CVE Numbering Authorities (CNA) authorized by NIST may create CPE names; currently, there are around 200 CNAs, mostly software suppliers or cybersecurity service providers. Organizations that wish to report a CVE (vulnerability), but do not themselves have a CNA on staff and are unable to identify an appropriate CNA to do this, may submit their requests to the MITRE Corporation “CNA of Last Resort”, which will identify an appropriate CNA to process the request.

In most cases, a software supplier applies to NIST for a CPE name for a product. When the name is assigned, the supplier can report vulnerabilities (each of which has a CVE name, assigned by CVE.org[8]) that apply to the product. Even though software products are named in many different ways as they are being developed and distributed, the product name used in the CPE is based on a specification[9] that has no inherent connection to the product itself.

The solution we are proposing for the software naming problem is based on an intrinsic identifier called purl[10], a contraction of “package URL”. As in the case of a simple chemical compound, knowing certain publicly available attributes of a software product enables anyone to construct the correct purl for the product. Moreover, anyone else who knows the same attributes will be able to construct exactly the same purl, and therefore find the product in a vulnerability database without having to query any central name registry (which of course does not exist for purl, since it is not needed).

Purl was originally developed to solve a specific problem: A software product will have different names, depending on the programming language, package manager, packaging convention, tool, API or database in which it is found.[11] Before purl was developed, if someone familiar with, for example, a specific package manager (essentially, a distribution point for software) wanted to talk about a specific open source product with someone familiar with a different package manager, the first person would need to learn the name of that product in the second package manager, assuming it could be found there. This would always be hard, because – of course – there is no common name to refer to.

This situation is analogous to the case in which an English speaker wishes to discuss avocados with someone who speaks both Swahili and English. The English speaker doesn’t know the Swahili word for “avocado”. To find that word, the English speaker uses an English/Swahili dictionary to learn that the Swahili word for avocado is *parachichi*. Neither avocado nor

---

[8] https://www.cve.org/About/Overview

[9] https://cpe.mitre.org/specification/

[10] https://github.com/package-url/purl-spec

[11] The reader does not need to understand what all these items are in order to understand the principle behind purl. In principle, nothing would be different if each of these items were a language, so the reader might think of the items simply as different languages.

parachichi has any connection to an actual avocado – they are simply names. They are extrinsic identifiers, and there is no way to know that they refer to the same thing without a central register, which in this case is the dictionary.

However, what if the two speakers didn't have ready access to an English/Swahili dictionary, either in hard copy or online? Most likely, the English speaker would find a picture of an avocado and show it to the Swahili speaker. The latter would smile broadly and say they now understand exactly what an avocado is. In fact, even if a dictionary were available, it would probably be easier for the English speaker to show the picture to the Swahili speaker. The picture is an intrinsic identifier; it is based on an attribute of the thing identified – in this case, what the thing looks like. Even if there were no English/Swahili dictionaries in existence, the picture would be a perfectly acceptable (in fact, preferable) identifier.

2. Native vs. non-native identifiers

The second principle behind our solution is native vs. non-native identifiers. A native identifier is one that is already in use in the ecosystem that serves as the namespace for the identifier. An example of this is - once again - names of simple chemical compounds. These names have been in use for at least 200 years and are understood by all chemists; moreover, they are embedded in literally millions of documents.

Most importantly, these names are derived from an activity that chemists perform all the time: creating and analyzing compounds. If a chemist sees the name "dihydrogen oxide" and isn't sure what that is, they can always use electrolysis to break this down into the two elements that make it up: hydrogen and oxygen. That is, the chemist will discover (probably to their great embarrassment) that the compound is water.

Purl identifiers are based entirely on the names in use in specific package manager ecosystems like maven; by its very nature, a purl is native to the ecosystem it is derived from. Therefore, purls are both intrinsic and native to the software ecosystem; moreover, they are already in widespread use[12]. For this reason, we believe that purl should be the basis of our proposal regarding naming software.

However, since CPE names can apply both to software and hardware products, the proposal described in this document needs to address both. Because only software products are found in package managers, purl will not work for hardware. Our group has determined that the existing GTIN and GMN standards[13] (described below) are native to the intelligent device ecosystem (for example, both UPC codes and RFID tags are included in GTIN and GMN). Even though they

[12] The Sonatype OSS Index database (https://ossindex.sonatype.org/) is currently used over 200 million times a month to search for vulnerabilities for software components, by a single open source product (https://dependencytrack.org/) . The database is built on purl.

[13] These are collectively referred to as the GS1 standards: https://www.gs1.org/standards.

require a registry and thus are extrinsic identifiers, the fact that they are native means they are superior to CPE numbers, which are neither intrinsic nor native to the hardware ecosystem.

Description of our solution: software

As discussed above, the software part of our proposed solution is based on purl.

How purl works

Modern development languages created in the last two decades benefit from the use of package managers, which describe the use of third-party and open source components used by an application. Package managers automatically resolve dependencies (i.e. components) at build time. Every component available through any package manager already has a corresponding Package URL. Purls are decentralized and are a natural extension of package managers. They provide a standardized way to represent component identity, regardless of from which ecosystem the component is derived.

Multiple sources of vulnerability intelligence already support purl for vulnerability lookup. This allows software suppliers to create BOMs with data originating from their build, and it allows consumers to analyze BOMs using any source of vulnerability intelligence supporting purl. The creation, consumption, and analysis of BOMs can in theory be entirely automated with this approach. However, the naming problem currently excludes fully automated use of the NVD, because the NVD doesn't support purl.

A purl can contain seven components, but only the following are required:

1. "Scheme", which is always "pkg".
2. "Type", which refers to the package ecosystem. Examples are "maven", "npm", "nuget", and "gem".
3. "Name". This is the name of the product in the package ecosystem.[14]

For example, if "foo" is a package found in the maven package manager, its purl is pkg:maven/foo.

Of course, since the first required item never changes, there are only two pieces of new information required to create a purl. Someone that wants to refer to an open source product as it's found in for example the maven build framework, simply needs to create a purl that contains "maven" and the name of the product as found in maven. Unless the person has made a mistake, the purl they create with those two pieces of information will allow anyone to locate the same product in maven, or to identify vulnerabilities for it in the NVD.

[14] The user can think of "type" as a language in which the product is named. If this were the case, the purl for avocado would be pkg:English/avocado. The purl for parachichi would be pkg:Swahili/parachichi.

Note that purl doesn't give precedence to any type. If something is defined in a type (in this case, products contained in maven), it is as important as in any other type. In other words, the fact that there might be 20 types with different names for the same product is not a problem: each has its own purl. Purl includes mechanisms like a "generic" type and parameters such as "repository_url" and "vcs_url" that enable it to handle many special cases not otherwise handled.

Even though each of 20 purls might refer to what is commonly understood to be the same product, it is important to keep in mind that there is no canonical name for the product. However, this actually solves a big problem: There will often be slight differences between instances of the same "product" when they reside in different package managers. Should these be called different products, or simply different "versions" of one product? Purl answers this question in favor of the first option: since the version in each package manager has a different purl, they are different products. Even when the code is exactly the same in different package managers, each version still has a different name and they are still different products (of course, the code could diverge at any time). NVD can then list all the relevant purls for a given CVE.

Because purl identifies packages, not just products, while CPE identifies only products, each module of a library will have its own purl (since each module corresponds to at least one unique package). Because of this, in the common case where a vulnerability appears in just one module of a library, it will be reported against the purl for that specific module in the NVD, once our proposal is implemented. An SBOM for a product that contains that module will list the purl of the vulnerable module, and a tool that looks up components found in an SBOM in the NVD will identify the appropriate CVE as applicable to the product. Just as importantly, if the vulnerable module is *not* present in the product, but one or more other modules from the library are present, the NVD search will not identify the CVE as applicable to the product.

Purl is not a perfect solution to the problems listed above; we know of no perfect solution. There are many cases where our solution will not be workable, and we will try to address them in the future. At the same time, we don't want to let the perfect be the enemy of the good, and we believe the advantages of using purl far outweigh the disadvantages. Any supplier that is uncomfortable with using purl will still be able to report product vulnerabilities using the CPE name.

New purl types

For our solution, we have requested that two new "types" be added to the purl specification. These will extend purl to offer at least partial coverage of proprietary components and legacy open source components, respectively.

SWID

Because proprietary software products are not often found in package managers, there needs to be some way to uniquely identify them. One widely-used means of identification for

proprietary software is Software Identification tags (SWID tags)[15], which are the basis of the ISO/IEC 19770-2 standard. SWID tags are intended to be prepared by the supplier and provided with the software binaries.

Purl supports SWID by extracting the most important identity attributes from a SWID document into a single purl universal resource indicator (URI), consisting of the software creator, tag creator, product, version, and tagID.

Because a SWID tag can be created for any version of a software product and because there is no central database of SWID tags, this means that, when a product's name and/or its supplier name changes, vulnerabilities will be reportable against both the "before" and "after" versions, as long as both versions have a SWID tag.[16] Thus, a user of a previous version of the product - from before it was acquired by the current owner - would be able to learn about vulnerabilities that apply to their version, which is often difficult or impossible today.

Software Heritage IDs

Another type of software that is not well-represented currently in purl is legacy software. Once a software product is no longer available in any package manager, it will not be possible to create a guaranteed unique purl for it. Therefore, legacy source software products need to be represented in purl in a different way than other open source products.

Software Heritage[17] is a non-profit organization that archives source code. To locate a particular product among the more than 12 billion source files already archived, the Software Heritage ID (SWHID) format was developed. Because SWHIDs contain a unique hash of the file, the hash value can be used as the name of the product in the purl identifier. The purl will consist of "SWH" as the type and the hash value as the name.

Our solution: hardware

We propose use of the following two types of hardware identifiers. Just like purl identifiers will supplement CPEs as identifiers for software in the NVD, GTIN and GMN will supplement CPEs as identifiers for hardware.[18]

---

[15] https://nvd.nist.gov/products/swid

[16] It is hoped that, once this proposal is adopted, SWID tags will be created for all new proprietary products. It is also hoped that some organizations will create SWID tags for legacy commercial products as well. This has already been done on an experimental basis for a large number of legacy products in the NVD.

[17] https://www.softwareheritage.org/

[18] It is important to note that our solution does not require that any current CPEs be removed from the NVD, or even that new CPEs stop being assigned. Our goal is that purls (for software) and GTINs/GMNs (for hardware) be made available as supplemental identifiers in the NVD. In other words, for a software or hardware product to be listed in the NVD (and for CVEs to be reportable for it), a CPE will no longer need to be assigned to it. Instead, users will be able to locate it using one of the above identifiers, and see any

Global Trade Identification Number (GTIN)

GTIN is an identification standard for all trade items developed by the international non-profit GS1. These identifiers are specifically designed to be unique to individual trade items and provide a simple way to uniquely identify trade items across multiple databases and organizational boundaries. GTIN is a superset of other standards, including those in the United States, Europe, and Japan, and thus may be 8, 12, 13, or 14 digits in length, depending on the specific GTIN format and encoding used.

GTIN consists of the following formats and encodings:

- GTIN-8 / EAN/UCC-8
- GTIN-12 / UPC-A
- GTIN-13 / EAN/UCC-13
- GTIN / EAN/UCC-14 or ITF-14

The UPC code affixed to the back or bottom of various products is a manufacturer's way of uniquely identifying the product to consumers. By leveraging GTIN in this way, consumers of hardware devices can look up that product's identifier in any GTIN database by scanning the UPC code and retrieving metadata about the device. If GTIN identifiers are included in the NVD, scanning the UPC code will allow consumers, using a simple app, to identify vulnerabilities affecting their hardware device. They could do this without having to manually look up the vendor, product, and version information in the CPE dictionary. This method would also allow end users to proactively identify vulnerabilities against hardware devices which may not have previously had a CVE, and thus a CPE.

Global Model Number (GMN)

GMN is a standard from GS1 which allows manufacturers to describe products and product families. The GMN and GTIN share the same Company Prefix (GCP) in the specification. However, a single GMN may have several GTINs. This allows databases to track a model number along with all the variants of each model number, without the need to track individual GTINs. In the European Union, the GMN supports the implementation of the requirements of the Basic UDI-DI under the EU Unique Device Identification (UDI) regulations on medical devices.

If extended to the NVD, support for GMN would allow CNAs to describe vulnerabilities against a model number, without the requirement to identify every possible GTIN that corresponds to one GMN.

One big advantage of using GTIN and GMN is that they are already in wide use in purchasing systems worldwide; no new identifier will need to be introduced into these systems.

CVEs that reference that identifier. However, as long as a supplier and its customers are happy with identifying their products using CPEs, they can continue doing that indefinitely.

Implementing this proposal

We believe that implementing this proposal will be relatively easy for software, and more time-consuming than difficult for hardware. This is because literally every naming technology mentioned in the proposal is already in widespread use by industry. Our proposal will place no additional burden on either software or hardware suppliers, with the possible exception of proprietary software suppliers that are not currently creating SWID tags for their products.

Conclusion

The “naming problem” is a serious impediment to software security in general and software bills of materials (SBOMs) in particular. Until significant progress has been made on this problem, the full benefits of SBOMs and related activities will be largely unachievable. Automated production or analysis of SBOMs, for example, may well be impractical. In this document, we have proposed what is in principle a “solution” to the naming problem, although it is inevitable that there will be some obstacles to full implementation. It is our hope that the importance of these obstacles will continually diminish over time.

We, the participants in the “SBOM Forum”, welcome any and all comments or questions on either our proposal or our characterization of the problem we want to solve. Please send any comments to tom@tomalrich.com.

Contributors

- Tom Alrich (Tom Alrich LLC)
- William Bartholomew (SPDX)
- Chris Blask (Cybeats)
- Bryan Cowan (Fortress Information Security)
- Cassie Crossley (Schneider Electric)
- Isaac Dangana (Red Alert Labs)
- Stefan Fleckenstein (MaibornWolff GmbH)
- Walter Haydock (Deploy Securely)
- Ed Heierman (Abbott Laboratories)
- JC Herz (Ion Channel)
- Derek Kruszewski (aDolus Technology Inc.)
- Bruce Lowenthal (Oracle Corporation)
- Tom Pace (NetRise)
- Dmitry Raidman (Cybeats)
- Melissa Rhodes (Medtronic)
- Steve Springett (OWASP, ServiceNow)
- Kate Stewart (SPDX, Linux Foundation)
- Tony Turner (OWASP, Fortress Information Security)
- Nicholas Vidovich (Finite State)
- Timothy Walsh, CISSP (Mayo Clinic)
- David A. Wheeler (Linux Foundation)
- Shivaram Umapathy (Beniva Consulting Group Inc., OWASP)
- Curtis Yanko (Grammatech)