Why we want to expand purl to proprietary software

OWASP SBOM Forum September 2024

It is no exaggeration to say that truly automated end-user vulnerability management, in which an end user organization automatically checks vulnerability databases to learn about new vulnerabilities that have been identified in software products they utilize, is almost impossible today.

This is because the database that provided the free vulnerability lookup services that a huge number of organizations rely on, the National Vulnerability Database (NVD), has essentially stopped performing that role. Moreover, there is no obvious replacement for it, either operationally or technically. We will discuss our reasons for saying this, and how we think this problem can be addressed, below. However, we need to first provide some background information.

The automated vulnerability management process

The OWASP SBOM Forum believes there are four primary components of a fully automated vulnerability management process:

1. Every private- and public-sector organization needs to build and continually update an inventory of software and firmware products, as well as intelligent devices, that it uses. This inventory can be updated using automated methods, but it may also be updated "by hand". In either case, the organization needs to have an up-to-date software and intelligent device inventory available, since truly effective vulnerability management is impossible without one.
2. Each organization needs to maintain, for each software product and intelligent device (collectively, each "product") in its inventory, one or more software identifiers that allow the organization to look up the product in a vulnerability database. The only such identifier currently supported by the NVD is CPE, which stands for "common platform enumeration". However, most vulnerability databases other than the NVD, and databases that are built on data downloaded from the NVD, are based on the "purl" identifier, which stands for "product URL".
3. Assuming an organization has items 1 and 2 in place, they can set up an automated process to look up every product in their inventory in the NVD and/or other vulnerability databases (there are many tools that do this). That process is designed to retrieve a list of all vulnerabilities, for each product in their inventory, that have been identified since the last time the organization searched for new vulnerabilities. Ideally, the organization will update this list daily; this is why it is so important to automate this process.
4. The end result of the above is a regularly updated and machine-readable pairing of a) products currently in the organization's inventory and b) vulnerabilities that have been identified in one or more of those products since the last time the organization searched for new vulnerabilities; we will refer to this as a "Vulnerability Action List". This list is then "fed

into" the organization's existing process for identifying and addressing[1] vulnerabilities that have recently been identified in the products it uses.

The role of vulnerability databases

The vulnerability management process described above depends on having one or more reliable vulnerability databases. While vulnerability databases today perform many roles, the most important of these, and the purpose for which they were originally developed, is to link vulnerabilities (usually, but not always, identified with a "CVE number") with the products that are affected by them.

In other words, a vulnerability database is essentially a set of statements of the form, "Product XYZ is affected by vulnerability CVE-2024-12345." Saying that a product is affected by CVE-2024-12345 is the rough equivalent of saying that an attacker might be able to compromise the product by basing their attack on CVE-2024-12345. The primary purpose of vulnerability management for an organization is to identify vulnerabilities that affect the products it uses, as well as to mitigate serious vulnerabilities.

Of course, the preferred method for mitigating a vulnerability is to apply a patch issued by the manufacturer of the device or the developer of the software or firmware. However, if the vulnerability is deemed serious and a patch is not available, the user organization should take action on its own to limit their risk, possibly including removing the vulnerable product from their network(s) altogether.

It isn't enough for the vulnerability database to identify a product using the manufacturer's official name for the product, even if that is combined with identification of the manufacturer itself. This is because proprietary (i.e., "non-open source") software products are usually known by many different names, as are the manufacturers of the products. For example, one consultant who was working with a large software developer asked employees that she worked with what company they worked for. She received over twenty different responses, each of which would be valid in a particular context. There was no single name that was self-evidently correct in all contexts. This is the case for almost all large software developers, which have absorbed hundreds or even thousands of smaller developers over time. Each of those acquisitions yielded new software products that continued to be referred to by their previous names for varying amounts of time.[2]

This is why a vulnerability database needs to utilize one or more software identifiers, which ideally will provide a machine-readable reference to a software product or intelligent device that will be

[1] Each organization will decide for itself how it will "address" vulnerabilities identified in software products it uses. This can include activities like coordinating with the software supplier regarding when important vulnerabilities will be patched, taking measures on their own to mitigate unpatched vulnerabilities, such as isolating affected devices on their own segment, etc. In other words, the fundamental contents of the Vulnerability Action List will not change from organization to organization, but how that list is used will vary widely.

[2] Of course, there are many other reasons, besides acquisitions, why a software product or company name may be referred to differently by different groups within the acquiring company.

valid regardless of the context of the reference. Without such an identifier, it is almost impossible to identify a software product with certainty.

The NVD's special role, and its special problem

The NVD has always occupied a special role in the vulnerability management ecosystem, primarily because it is owned and operated by the US government[3]. Access to it has always been free, and it is (or at least *was*) heavily used by private- and public-sector organizations worldwide. Understanding the NVD's unique role requires first understanding how vulnerabilities – at least the great majority of them which are identified using a CVE number - are identified and reported:

1. Software vulnerabilities are almost always identified by the developer of the software. Vulnerabilities identified in intelligent devices are usually reported by the manufacturer of the device.
2. Many of the developers and manufacturers are CVE Numbering Authorities (CNAs), of which there are currently around 400. CNAs are authorized by CVE.org, a non-profit organization sponsored by the US Department of Homeland Security (DHS) and the Cybersecurity and Information Security Agency (CISA), an agency of DHS. CVE.org (which used to be known just as "MITRE", and which is staffed entirely by contractors employed by the MITRE Corporation) manages the CVE program and maintains the database of CVEs.
3. Whenever a CNA identifies a new vulnerability in one of their products, they assign a CVE number to it and include that in a report to CVE.org. While that report can contain many types of information, the information types most relevant for the current purpose include:
    a. The CVE number assigned by the CNA.
    b. A textual description of the product (or products) affected by the vulnerability. This often includes the name by which the developer refers to the product. However, since even within a developer organization, one product may be referred to with multiple names, the name utilized in the textual description can never be taken as an exact identifier.
    c. For example, if a product was recently acquired by Company A along with Company B, which developed the product, the former employees of Company B may still refer to the product by the previous name, even if Company A has renamed the product. In fact, an acquired product will often be sold for years under the previous name and will only be renamed when the acquiring company decides for marketing reasons to rebrand the product.
    d. Sometimes, the CNA develops a CPE name for the product and includes that in the CVE report. However, the NVD staff members (who are mostly employees of NIST) have generally discouraged CNAs from doing this and have usually developed their own CPE name to replace the one generated by the CNA. This has made CNAs reluctant to add CPE names to the CVE reports they create.

[3] The NVD is part of the National Institute of Standards and Technology (NIST), which is part of the US Department of Commerce.

e. The most recent CVE specification allows the CNA to include a purl identifier for the product. However, it is likely that no CNAs are doing this today, since there is no provision for utilizing the purl identifier, either in the CVE.org database or in the NVD itself.

After the CNA submits their CVE report and it is accepted by CVE.org, it is entered into the CVE.org database. From there, it is transferred to the NVD, which incorporates the new CVE number, and most of the textual information from the report, into the NVD. In addition to this, until early 2024, a NIST/NVD staff member almost always created a CPE name for each product listed in the CVE report and added it to the report. Doing that linked the CPE name to the CVE number in the NVD. This step is part of the process called "enrichment".

It is this link that allows an automated search of the NVD to learn that "the product identified with (for example) CPE345678 is affected by (for example) CVE-2024-12345". If the CPE name has not been added to the CVE report, either by the NIST/NVD staff or by an "Alternate Data Provider" (ADP) like CISA or MITRE, a search on the CPE name will never indicate it is vulnerable to that CVE. Since the affected product(s) should always be listed in the text of the CVE report (as created by the CNA), only an enterprising soul who wants to read all 18,000+ CVE reports that are currently "unenriched" will be able to determine with certainty whether the product they are concerned with appears in any of those reports.

Of course, this isn't usually going to happen, although at least a few commercial vulnerability database providers have read all the unenriched CVE reports, created their own CPE names, and added them to the reports in their system. This means that a search on the CPE name created by the database provider in the provider's own database should in fact reveal whether CVE-2024-12345 applies to that product. That is good, but there can be no certainty that, when and if the NVD gets back on its feet, they will incorporate those CPE names into the NVD.[4]

In early February 2024, the NVD experienced some problem that caused it to drastically reduce the number of CPE names it added to CVE reports. As a result, there is currently a huge number of CVE reports that are in the NVD (meaning that a search on the CVE number will reveal the textual contents of the report), that do not have a CPE name associated with them[5].

The result of this is that currently an automated search for a particular product in the NVD is unlikely to reveal any CVE number to which the product is vulnerable, if that CVE was added to the

[4] This will not be true for CPE names created by CISA as part of their Vulnrichment program, since CISA is an ADP. Those CPE names will likely remain in the NVD.

[5] This blog post from July identified the backlog of "unenriched" CVE reports (i.e., CVE reports to which the NVD has not added a CPE name) at 17,000 and growing by almost 100 per day. The backlog continues to increase and is now well over 18,000. While a small number of Alternate Data Provider organizations (especially CISA) has created CPE names that have been added to CVE records, this is only slowing the growth rate of the backlog, not reversing it.

For perspective, the total number of CVEs identified since 1999 (when the first CVE was identified and released) is, as of July 2024, over 240,000. This means the current backlog of unenriched CVEs is close to 1/12 (8.33%) of all CVEs ever identified, although, to be fair, the number of new CVEs has jumped markedly this year, as mentioned in this blog post.

NVD after February 12. This means that a user that searches the NVD, looking for recent vulnerabilities that affect the software products they use, is unlikely to learn about more than a small percentage of vulnerabilities identified since February, unless they are willing to conduct a lengthy non-automated search.

Since it is safe to say that most users of the NVD (or any other vulnerability database) are more concerned about current vulnerabilities than those that were identified more than eight months ago, this means that automated searches of the NVD today yield little value, although "manual" searches (in which a user is willing to spend their time reading the text in CVE reports) will still be useful in some cases.

How do we solve this problem?

Of course, everyone hopes that the NVD will be back on its feet quickly; the NVD itself continues to put out optimistic messages about how soon that will happen. However, given the size of the backlog of unenriched CPEs, it is reasonable to doubt the NVD will be fully recovered before sometime in late 2025, if even then. It would be foolish not to at least start planning for "life without the NVD".

This is especially important, given the serious problems with the CPE identifier, which were identified in the "proposal" for fixing naming problems in the NVD that was released in 2022 by the SBOM Forum (now the OWASP SBOM Forum). Pages 4-6 of that document list six serious problems with CPE, although there are others as well.

The biggest problem with CPE is that there's no way to predict, using just the CPE specification and knowledge about the product and version being specified, exactly what CPE will be created for that product/version. This is because the CPE name includes several mandatory fields for which there is no single value, even in principle.

For example, one of the fields in CPE is "vendor". We have already discussed the fact that staff members of a large software developer don't agree on what their company is called. This is aggravated by the fact that even tiny differences in spelling are significant in CPE. For example, a CPE name that includes the vendor name "Microsoft" is different from a CPE name that includes "Microsoft, Inc", which in turn differs from the CPE name that includes "Microsoft, Inc." with a period. This list could go on and on. A search for a CPE name with any one of these variants will come up with no results if the CPE name was created using one of the other variants, even if the only difference is the presence or absence of a period.

Of course, there are many ways to get around these problems using fuzzy logic, machine learning, etc. In fact, it's safe to say that almost every major organization – such as a software developer – that regularly searches the NVD has their own bag of workarounds that make the experience less painful, although far from being automated.

However, these workarounds all violate the principle we described at the beginning of this paper: given the huge number of software products that the average medium-to-large organization utilizes and the need to check daily for newly identified vulnerabilities in all those products, it is essential

for the vulnerability management process to be as automated as possible. If identifying vulnerabilities can't be automated in any realistic sense due to ambiguities in CPE names, this means truly automated vulnerability management simply isn't possible.

If there were no realistic alternative to CPE names for identifying software products, the above situation would be quite depressing, since it seems like we're at a dead end with no way out. However, as already mentioned, the purl identifier currently offers at least a partial way out, although more work is required before it can provide a complete solution to the problem of software identification in the CVE ecosystem.

The 2022 OWASP SBOM Forum paper titled "A Proposal to Operationalize Component Identification for Vulnerability Management" describes in detail why purl is far superior to CPE as a software identifier for vulnerability management purposes. The essence of the argument is:

> "Intrinsic" software identifiers are far preferable to "extrinsic" identifiers. An intrinsic software identifier is one for which the user of a product will "intrinsically" already have all the information they need to create an identifier that exactly matches the identifier used by the supplier when they reported the vulnerability to CVE.org. By contrast, an extrinsic identifier requires a lookup to some external data source in order to have an exact match – and even that may not be possible.
>
> An example of an intrinsic identifier is the name of a chemical compound. A sodium chloride (table salt) molecule includes one atom of sodium and one of chlorine; it is written as NaCl. If one chemist wants to communicate to another chemist the exact formula of a compound they have created, they can simply write down the name. There should never be any ambiguity, unless one of the chemists simply makes a mistake.
>
> By contrast, if one person needs to know another person's Social Security number, there is no way they can guess it based on characteristics of the person, their history, etc. They need to look it up in some database (although hopefully not a public one). Therefore, Social Security number is an extrinsic identifier.
>
> CPE name is also an extrinsic identifier. Even though there is in theory a formula for a CPE name, the different components of the formula that were used in a particular CPE name (vendor name, product version string, etc.) can't be predicted with any accuracy, for reasons mentioned earlier and described on pages 4-6 of the SBOM Forum's 2022 white paper. Therefore, frequent NVD users need to utilize information from a variety of external information sources, including the NVD's "CPE Dictionary" (which isn't a dictionary at all, but simply a list of every CPE name that has ever been created by the NVD), in order to find the CPE name used to reference a particular product. Moreover, the same product and version may be referred to with different CPE names, created by different NVD staff members at different times.
>
> By contrast, purl is an intrinsic identifier. It currently applies primarily to open source software packages that are available in package managers like Maven Central and PyPI, although the OWASP SBOM Forum wants to extend that applicability to proprietary software.

If a software user has downloaded a package from the PyPI package manager and wishes to look up vulnerabilities that apply to that package in an open source vulnerability database like OSS Index (which, like almost all open source vulnerability databases, utilizes purl identifiers), they can create the purl for the package simply by knowing

1. The "scheme", which is always "pkg".
2. The name of the package manager from which the user downloaded the software. The package manager name is referred to in purl as "type". In this case, the type is "PyPI".
3. The name of the product *in that package manager*. No matter what the "same" product might be called in a different package manager, its name will never vary in PyPI.[6]
4. The "version string" for the version of concern (this is optional, but it isn't likely that many vulnerability searches won't include a version string).

The purl for version 1.11.1 of the package named "django" in PyPI is "pkg:PyPI/django@1.11.1".

Note that, absent somebody making a mistake, only one purl matches any package in a package manager. When the supplier of the package (probably the community that maintains django) reports a vulnerability for version 1.11.1 of the package, they will use the same purl that the user creates when they want to look up vulnerabilities in the package. In other words, there is no "dictionary" of purls, nor is there any need for one. An open source product in a package manager intrinsically has its own purl, just like a chemical compound intrinsically has its own chemical formula. This is why purl has literally taken over the world of open source software found in package managers.[7]

Why do we need to extend purl to proprietary software?

We have already pointed out that currently the purl identifier only "supports" open source software that is available in package managers, although it is universally used in that space. Of course, that alone justifies adding purl support to the NVD, since CPE is not a good identifier for open source software (for example, it isn't at all clear what should be entered in the CPE "vendor" field, since a free product by definition does not have a vendor. The purl spec has a vendor field, but it is optional, not required).

[6] More specifically, the name of that *version* of the package will never change in the package manager. A later version of the product might be given a different name, but the purl for an existing version of the product should never change.

[7] A lot of open source software products, notably software products written in C and C++, are not available in package managers and therefore do not have purls. The OWASP SBOM Forum hopes someday to extend purl to those products, but we feel it is currently much more important to extend purl to proprietary software.

However, the NVD also needs a reliable identifier for proprietary ("closed source") software. Even if the NVD didn't have the current huge backlog of unenriched CVE reports, it would still have the problem that CPE is not a reliable identifier for closed source software, any more than it is for open source software.

This is especially important because most public and private sector organizations worldwide strongly prefer to run their operations using proprietary software; they consider the fact that they have to pay for the software to be an advantage, since they know they can normally expect a prompt response when they contact the supplier with a problem. That can't be counted on with open source software.[8]

There are some open source software products (such as the Chrome browser) that are widely used in many organizations (and there are many more open source products that are in widespread use, but often without the knowledge of the organization that uses them. This includes a lot of open source products like Java libraries, that are components included in commercial products). However, the fact that there are so few open source software products in heavy use by private sector organizations shows they are the exceptions that prove the rule.

To summarize, few organizations will be able to run a comprehensive vulnerability management program unless there is a reliable identifier they can use to learn about vulnerabilities found in the proprietary software that they operate. Since CPE is not a reliable identifier and since purl is already widely used for open source software, it makes sense to determine whether it is possible to extend purl to proprietary software.

How can we extend purl to proprietary software?

To answer the above question, we first need to determine what is essential to make purl work. An important feature of purl is that the user obtains the software from a single location: the primary URL for the package manager that contains the software. Any software downloaded from that package manager (e.g., PyPI) will always have the same "type" in purl. This means that, if the user knows the package manager name as well as the product name and version string *in that package manager*, they should always be able to recreate the purl that the supplier used when they reported a vulnerability that applies to that product and version.[9]

What we have just described is a "controlled namespace", meaning the product names in the package manager are under the control of the operators of the package manager; the operators can

[8] Of course, there are organizations like Red Hat™ that sell support for open source software, even though the software itself is usually free.

[9] This assumes that the open source project team that developed the software reported the vulnerability to CVE.org (through a CNA, if they are not themselves a CNA), and that they included the purl for the software in the CVE report. Even though the current CVE specification supports purl, this is a recent development; there has been no training, guidance, etc. for the CNAs in how to report purls. Of course, if the NIST/NVD staff were trained in developing purls, they could add the purl when the CNA has not done this. However, that is not likely to happen soon, since the NVD does not technically support purl now, and it is unlikely that situation will change for years.

be counted on to manage the names within their namespace by removing duplicates, making sure each name is unique within the namespace, etc. Probably the most important aspect of purl is that it provides a "distributed namespace" – a set of individual namespaces linked to download locations. It doesn't matter how many namespaces (i.e., purl types) there are, as long as each namespace is controlled.

On the other hand, there is only one namespace for CPE names: the set of CPE names found in the NVD. These have (mostly) been created by NIST/NVD employees and contractors. Since the NVD doesn't have a fixed methodology for creating a CPE name (including a fixed methodology for choosing a vendor name, product name and version string), there is no way for a user to be certain to create a name that will match the one used when the vulnerability was reported.

This is why truly automated vulnerability management will never be possible if the software security community continues to rely on a database(s) that is based on CPE names, like the NVD. How can purl be extended to proprietary software, so that most organizations will someday be able to enjoy truly automated vulnerability management?

As just discussed, it seems that the key to purl's success as an open source software identifier is the fact that it is based on a "federation" of controlled namespaces, specifically package managers. Since proprietary software is not usually available in package managers, we need to identify one or more ways in which there can be a controlled namespace for proprietary software. In other words, we need to identify a way for a user of a proprietary software product to create, using information they have readily available to them, a purl that is certain to match the purl entered by the supplier of that product when they (or a CNA acting on their behalf) originally reported the vulnerability to CVE.org.

Two options for purls for proprietary software

There are probably many ways that a purl can be created for a proprietary software product, while at the same time being linked to a controlled namespace. Two of these have been identified by Steve Springett, a founding member of the OWASP SBOM Forum who is better known as creator and leader of the OWASP Dependency Track and CycloneDX projects.

1. *Purls for online "software stores"*

Steve has pointed out that, in the world of proprietary software products, the closest thing to a package manager is a "software store" like Google Play™, the Apple Store™, or the Microsoft Store™. Like a package manager, a software store provides a single download location for a variety of software from many suppliers. Also like a package manager, a software store provides a controlled namespace; the store can be counted on to make sure each product in the store has a unique name.

Of course, the primary difference between a software store and a package manager is that in a store, software is mostly for sale, while in a package manager the software is usually available for free, although it comes with a license restricting its use. However, this difference is not significant.

In both cases, the namespace is controlled. The store can be treated by purl as if it were a package manager, meaning it will have its own purl "type".

While most software products and intelligent devices are probably not sold in online stores, millions of those products are available in stores. It is impressive to think that all those products could instantly acquire a purl identifier as soon as a new type is incorporated into the purl specification (each store would require its own purl type).

However, even though treating a software store like a package manager in purl might seem to be simple, in practice there will undoubtedly be many challenges. The SBOM Forum would like to conduct at least two proofs of concept for creating a purl type for a software store, then testing many dummy transactions to test all aspects of the experience. Of course, the stores used in the proofs of concept don't have to be huge, like Google Play. In fact, it may be better to prove the concept with one or two small stores, then try it with a larger store.

Before any software store can be recruited to participate in a purl proof of concept, there will need to be a set of requirements for all software stores, such as having a mechanism to parse a purl and use the results to locate a particular product in the store.

2. Purls for software not sold in online stores

Creating a controlled namespace for software not sold in online stores is more difficult than for software sold in online stores; however, it can certainly be accomplished. The problems are not technical; in fact, there are probably many ways to do this. Instead, the problems involve designing appropriate procedures for making the required information available to software users, and then getting buy-in from many software developers for implementing those procedures.

In the SBOM Forum's 2022 white paper (referenced above), we included this section on pages 11 and 12:

> **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), 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 tag 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. 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. If CPE is the only available software identifier, a user that acquired a product before it was transferred to

a different company might never learn of vulnerabilities that were reported after it was transferred.

SWID tags were developed to be a much more comprehensive (and well-defined) identifier for software than CPE names. Based on the ISO/IEC 19770-2 specification (which was developed by NIST), SWID tags are XML files that, according to the NVD website, are "…composed of a structured set of data elements that identify the software product, characterize the product's version, the organizations and individuals that had a role in the production and distribution of the product, information about the artifacts that comprise a software product, relationships between software products, and other descriptive metadata."

SWID tags were intended to be the replacement for CPE and to be distributed with the binaries for a software product. Some software developers, including Microsoft, distributed SWID tags with all their software binaries for at least two years. However, for various reasons, SWID tags never achieved critical mass, and NIST has recently officially abandoned the idea of replacing CPE names with SWID tags.

Nevertheless, SWID tags provide a well-defined format for presenting metadata about a software product. If a software developer creates a SWID tag whenever they "ship" a new or revised software product or version of a software product, and if they make it available for inspection by an automated vulnerability management tool like OWASP Dependency Track, the tool could easily extract from the SWID tag the four or five pieces of information required to create a purl with the type "SWID". This will normally match the purl used by the developer to report any vulnerabilities to CVE.org. This is another example of a controlled namespace, this time controlled by the developer of the product.

The SWID type was added to purl while the SBOM Forum was developing its 2022 white paper. Currently, this is the specification for the SWID type in purl, although the suitability of the specification needs to be tested. Note there are fields for "software creator" (the developer of the software) as well as "tag creator", if this is not the same as the software creator.

Steve Springett has created a tool that accepts manual input of fields from the SWID tag for a product and creates the purl based on that input. Of course, in the future an automated tool like Dependency Track could import the SWID tag and output the purl.

The 2022 white paper did not address a very important problem: how a user can locate the SWID tag for a software product they use, especially if it is not a currently supported product. If it is a current product, the tag might be available in the binaries (although there will certainly be cases when the tag is not found in the binaries, or when the binaries themselves are unavailable).

However, for legacy products that did not originally include a SWID tag, the developer – or their successor company, if the original developer has been purchased or merged with another company, or if the product was sold to another company – will need to create one[10]. But how will an

[10] Five or six years ago, in anticipation of SWID becoming the standard software identifier for the NVD, the NVD generated a large number of SWID tags for existing software products. If that trove could be accessed, it might be utilized to "jump start" the effort to generate legacy purls.

end user, or more specifically the user's vulnerability management tool, locate the appropriate SWID tag?

In order not to "break" the automated vulnerability management process, the tool will need to locate and read the tag without human intervention. The tool will already have certain information that is available to the user, including the vendor name, product name and version string. Using that information, plus information that is only available on the SWID tag (specifically, the Tag ID) the tool should be able to generate the same purl that the supplier used when they reported vulnerabilities in the product.[11]

There are probably many mechanisms by which the user's tool could locate the correct SWID tag. Two of them are:

A. The supplier of the product could maintain on their website a page with a well-known name, such as "purlswid.txt". The user's tool could locate this page and search for the correct SWID tag, using whatever information the user has available about the software. Usually, by searching using the vendor name (which will normally match the name on the website), product name and version string shown on the "About (product name)" page of the software, it should be easy for the tool to find the correct SWID tag.
B. The Transparency Exchange API, under development by the CycloneDX project, will be able when complete to locate a wide variety of software supply chain artifacts, including SBOMs, VEX and VDR documents, and SWID tags.

Whatever mechanism is chosen, it cannot require a central database of SWID tags, since doing that is likely to result in problems like those currently being experienced with CPE names, and also because this immediately raises the question of who will pay for maintaining that database.

The database must be a distributed one, in which each supplier is required to maintain and make available for inspection the set of SWID tags they have created for their products. Of course, the objection will immediately arise that some suppliers will do their part and others won't, meaning there will be large gaps in the database. Also, some suppliers will create usable SWID tags and others won't. Unless there's a central database enforcing discipline on the suppliers, how can these problems be avoided?

There won't be a central database of SWID tags, but discipline can be enforced by CVE.org. When a CNA creates a purl for one of their products and adds it to a CVE report, they can also be required to provide the SWID tag for the product. CVE.org could compare the SWID tag to the purl that is based on the tag, to confirm the information matches.

Another consideration is that when these problems come up, there will be a self-correcting mechanism: If a supplier doesn't provide SWID tags for their software at all or does a poor job of it, their customers won't be able to learn about vulnerabilities in that supplier's products. A supplier

[11] In case the supplier didn't generate a Tag ID when they created the SWID tag, the tool could create a purl that utilizes all the fields that are filled in on the SWID tag. An algorithm might then need to decide whether the match with the purl in the vulnerability database is close enough to declare the user's product to be the same one referenced in the database. If it isn't possible to declare a match, the user could be warned of that fact and advised to make their own decision on whether there is a match.

that doesn't provide SWID tags for their products will ultimately face pushback from their customers and may at some point find they are at a disadvantage compared to their competitors who *do* provide SWID tags.

If it seems unlikely that customer pushback could ultimately lead to widespread adoption of particular security practices by software suppliers, consider patching. When widespread adoption of the internet led to increasing attacks based on software vulnerabilities, it took years before it became safe to assume that most software suppliers offering their products for sale would also provide free patches. Before then, many suppliers didn't provide patches at all, and many others charged for the service. It was only when software users in general came to realize the advantage of having free patches available that they demanded it of their suppliers. It is likely that adoption of purl by proprietary software suppliers will follow a similar trajectory.

3. Implementing the SWID tag option

Implementing the option for software stores will be relatively easy, since the stores are in principle no different from package managers. However, implementing the SWID tag option for proprietary software will require a number of steps. These could include:

A. A "tabletop exercise", in which multiple software suppliers test use of the current SWID type specification to determine whether purls created using that specification can successfully identify software products in practice. It is possible that this exercise will reveal that the current specification needs to include other SWID fields (there are around 80 fields in the SWID specification. The current purl SWID type specification only includes a small number of these[12]).
B. After the tabletop exercise is finished, publication of the final specification for the SWID type in purl. There needs to be a separate specification for each software store.
C. Development of proof of concept code to parse SWID tags and create a purl, which is in principle what Steve Springett's "Package URL SWID Generator" app does. Developers of tools that search vulnerability databases will be able to incorporate this code into their tools.
D. Training software suppliers to use Steve's app to generate a SWID tag for a new (or legacy) product/version.
E. Training CNAs to include purls in their CVE reports.[13]

[12] In addition, there is no reason why the specification for the SWID purl type needs to limit itself to the fields in the SWID specification. After all, the SWID tags created for purl purposes should not be used for any other purpose served by SWID tags (for example, SWID tags have been cited as a "low-cost" SBOM format). Other fields that have been suggested for the SWID purl type are download location, patch location, support location and "information URL". It will be up to the group that develops the specification to determine what fields are needed.

[13] The CNAs will need to be trained on incorporating purls for both open source products and proprietary products in their CVE reports. Note that the CVE.org database will need to be able to accommodate purl as an alternative identifier for software products, along with CPE.

F. Working with operators of vulnerability databases that are already based on purl, such as OSS Index and OSV, to accommodate the SWID type.[14]
G. "Evangelizing" software developers about the importance of creating SWID tags for both current and legacy product versions.
H. Explaining to both software users and security tool vendors (e.g., software composition analysis tool vendors) the importance of purl in general, and of the new types that accommodate proprietary software.

Version ranges

A software vulnerability seldom appears suddenly in one version of a product, then disappears in the next version. Instead, it usually remains in the product for multiple versions (or even multiple years), until it is finally patched or removed for other reasons. This means that vulnerability management is likely to work much better if it can be based on version ranges, not individual versions.

Today, CVE reports often state that a vulnerability applies to a range of versions in a software product, not just to one version or multiple individual versions. However, that assertion is strictly in text; there is no way to specify a version range in a machine readable format, since CPE does not support version ranges. This means that, if a user performs an automated search on the NVD, they will not learn that a CVE affects a range of versions, unless they also read the text of the CVE report.

Ideally, a software identifier should be able to specify a version range in a standard format that states in effect, "Versions 3.4 through 5.6 of product XYZ are affected by CVE-2024-12345. Other versions are not affected." A user tool that parses that identifier will identify every version that falls in that range, and mark each one as affected by the vulnerability. That is, the user tool will understand the above statement to mean, "Versions 3.4, 3.5, 3.6…5.5, and 5.6 are affected by CVE-2024-12345."

Currently, if a user notices a version range in a CVE report that applies to a product they utilize and they want to make sure every version within the range is noted to be affected in their vulnerability tracking system, they will have to annotate each version manually. Since a multiyear version range could easily contain hundreds of versions (including minor versions, patch versions, separate builds, etc.), this could turn into a very time-consuming process.

Recognizing this problem, a few years ago, the purl community developed a "version range specifier" called vers. It provides a simple specification for version ranges. For example, a range that includes version 1.2.3, as well as versions greater than or equal to version 2.0.0 and less than version 5.0.0 would be specified as "1.2.3|>=2.0.0|<5.0.0".

The simplicity of vers comes at a cost: It only applies to certain versioning schemes in which the elements of a range can be specified using simple arithmetic. For example, if I have version 3.2.5 of the product to which the above range applies, I can easily determine that my version falls within

[14] It is likely that existing purl databases will have to make only minimal changes to accommodate new purl types for software stores. However, that may not be the case for the new SWID purl type.

that range, whereas version 5.4.6 falls outside of the range. On the other hand, a versioning scheme that uses letters is not supported by vers, since there is no way to be certain whether "version 4.6B" falls within the above range or not. The vers specification lists versioning schemes that are supported, although it is possible that a scheme that isn't on the list, but in which versions can be compared using just addition, subtraction, and greater than/less than, would work fine with vers.

One of the main reasons why CVE.org should move to purl as the primary software identifier is that by doing so, a purl in the database could identify an entire version range, not just a single version of the product. In other words, besides doing a better job of identifying individual product/version pairs than CPE does, purl would add a new – and greatly needed - capability to what is in place today. Even though the OWASP SBOM Forum wants to focus our efforts on expanding purl so it can identify proprietary software products, we will try to include vers in whatever we create, so that in the hopefully-not-far-off future, version ranges can be fully incorporated in purl.

The way forward

One year ago, talking about moving from CPE to purl in the "CVE ecosystem" would have just been a way of showing you're intellectually curious – like the people who continually push Esperanto as the universal language. However, the fact that the NVD has so far this year assigned CPE names to fewer than one third of the total CVE reports it has received and that the backlog continues to grow, not shrink, has added urgency to a problem that the OWASP SBOM Forum called out in our 2022 white paper: CPE is a deeply flawed software identifier and can never be "fixed". We need to move away from it as quickly as possible.

While nobody expects CPE to disappear tomorrow (if ever), the fact remains that purl is a much better identifier. It has literally taken over the open source world, and now needs to be expanded to address proprietary software. Of course, that alone will take years, but we'll never finish if we don't start. This needs to be done, if we ever expect to have accurate attribution of software vulnerabilities.

Ideally, the OWASP SBOM Forum would push forward and emerge in say two years with a new purl specification that supports proprietary software in both of the ways described in this white paper: by implementing software stores as new purl types and by implementing the SWID type. Of course, that also requires that all the practices required to make both options work correctly will be in place. We would also love to see all the new purl types be usable in CVE reports – with the CNAs fully trained and supportive of these changes.

Accomplishing the above is certainly a possibility, but it will require that a lot of resources be thrown at the effort, in simultaneous workstreams, and starting immediately. This is not a realistic scenario. A realistic scenario would divide the effort into three parts:

1. Planning – scoping out the whole project, at least at a high level. This will include planning for items 2 and 3 below, although they may not be executed simultaneously.
2. Software stores – designing, testing and implementing the option in which individual software stores have separate purl types.

3. SWID tags – designing, testing and implementing the option in which software not available in stores is identified using a purl type whose contents are based on a SWID tag created by the supplier. A discovery method for the tags will need to be worked out.

Of course, item 3 will be a complex endeavor, which will involve a lot of "cat herding" and moving parts. On the other hand, the software stores in item 2 might be easy to accommodate in purl, since they resemble package managers in so many ways. Given that resources will certainly be limited, it is probably most realistic to tackle items 1 and 2 first, and later work on item 3, as resources become available. However, if it turns out there are enough resources available early on, it would certainly be possible to implement items 2 and 3 as parallel workstreams.

The first step in item 2 will be doing a proof of concept with one or two software stores. Maybe it will involve one of the "big three" stores (Apple, Microsoft and Google Play), and maybe it will involve a few smaller stores. In either case, the point will be to:

1. Design a preliminary specification for each store's purl type and test it with the store in a tabletop exercise – then modify the spec as needed. Of course, the specification for one store's purl type will not be the same as the spec for another store's purl type.
2. Submit a pull request to the purl group to implement each store's purl type.
3. Request that the store put in place whatever coding is needed to implement purl lookups using that spec, for a small test group of products.
4. Test lookups for the test group.
5. If the test works well, gradually roll out lookup capability to all products in the store.

Note what is *not* in the above list: creating a purl for each product in the store. The nice thing about purl is that, once there is a type for a controlled namespace (whether it's a package manager like Maven Central or a software store like Google Play), all the products in that namespace automatically have a purl, whether or not it has ever been explicitly identified.

The OWASP SBOM Forum is very interested in furthering the use of purl in the CVE ecosystem, since it will help address both a short-term problem (the lack of machine-readable identifiers for about 2/3 of the new CVEs identified this year) and a long-term problem (the fact that CPE is a deeply flawed identifier that can't be fixed).

However, we can't do this on our own. We want to cooperate with other organizations and interested individuals, both as participants in the project and donors to the effort. Donations can be made to OWASP and "restricted" to the SBOM Forum. OWASP is a 501(c)(3) non-profit organization. If you are interested in joining us and/or donating, please email Tom Alrich at tom@tomalrich.com.